

OPSAWG
Internet-Draft
Intended status: Informational
Expires: October 25, 2021

B. Claise
Huawei
J. Quilbeuf
Independent
D. Lopez
Telefonica I+D
D. Voyer
Bell Canada
T. Arumugam
Cisco Systems, Inc.
April 23, 2021

Service Assurance for Intent-based Networking Architecture
draft-claise-opsawg-service-assurance-architecture-05

Abstract

This document describes an architecture for Service Assurance for Intent-based Networking (SAIN). This architecture aims at assuring that service instances are correctly running. As services rely on multiple sub-services by the underlying network devices, getting the assurance of a healthy service is only possible with a holistic view of network devices. This architecture not only helps to correlate the service degradation with the network root cause but also the impacted services when a network component fails or degrades.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 25, 2021.

Copyright Notice

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

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

Table of Contents

1. Terminology	2
2. Introduction	5
3. Architecture	6
3.1. Decomposing a Service Instance Configuration into an Assurance Graph	9
3.2. Intent and Assurance Graph	10
3.3. Subservices	11
3.4. Building the Expression Graph from the Assurance Graph	11
3.5. Building the Expression from a Subservice	12
3.6. Open Interfaces with YANG Modules	12
3.7. Handling Maintenance Windows	13
3.8. Flexible Architecture	14
3.9. Timing	15
3.10. New Assurance Graph Generation	15
4. Security Considerations	16
5. IANA Considerations	16
6. Contributors	16
7. Open Issues	16
8. References	16
8.1. Normative References	16
8.2. Informative References	17
Appendix A. Changes between revisions	18
Acknowledgements	18
Authors' Addresses	19

1. Terminology

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.

SAIN Agent: Component that communicates with a device, a set of devices, or another agent to build an expression graph from a received assurance graph and perform the corresponding computation.

Assurance Graph: DAG representing the assurance case for one or several service instances. The nodes (also known as vertices in the context of DAG) are the service instances themselves and the subservices, the edges indicate a dependency relations.

SAIN collector: Component that fetches or receives the computer-consumable output of the agent(s) and displays it in a user friendly form or process it locally.

DAG: Directed Acyclic Graph.

ECMP: Equal Cost Multiple Paths

Expression Graph: Generic term for a DAG representing a computation in SAIN. More specific terms are:

- o **Subservice Expressions:** expression graph representing all the computations to execute for a subservice.
- o **Service Expressions:** expression graph representing all the computations to execute for a service instance, i.e. including the computations for all dependent subservices.
- o **Global Computation Graph:** expression graph representing all the computations to execute for all services instances (i.e. all computations performed).

Dependency: The directed relationship between subservice instances in the assurance graph.

Informational Dependency: Type of dependency whose score does not impact the score of its parent subservice or service instance(s) in the assurance graph. However, the symptoms should be taken into account in the parent service instance or subservice instance(s), for informational reasons.

Impacting Dependency: Type of dependency whose score impacts the score of its parent subservice or service instance(s) in the assurance graph. The symptoms are taken into account in the parent service instance or subservice instance(s), as the impacting reasons.

Metric: Information retrieved from a network device.

Metric Engine: Maps metrics to a list of candidate metric implementations depending on the target model.

Metric Implementation: Actual way of retrieving a metric from a device.

Network Service YANG Module: describes the characteristics of service, as agreed upon with consumers of that service [RFC8199].

Service Instance: A specific instance of a service.

Service configuration orchestrator: Quoting RFC8199, "Network Service YANG Modules describe the characteristics of a service, as agreed upon with consumers of that service. That is, a service module does not expose the detailed configuration parameters of all participating network elements and features but describes an abstract model that allows instances of the service to be decomposed into instance data according to the Network Element YANG Modules of the participating network elements. The service-to-element decomposition is a separate process; the details depend on how the network operator chooses to realize the service. For the purpose of this document, the term "orchestrator" is used to describe a system implementing such a process."

SAIN Orchestrator: Component of SAIN in charge of fetching the configuration specific to each service instance and converting it into an assurance graph.

Health status: Score and symptoms indicating whether a service instance or a subservice is healthy. A non-maximal score MUST always be explained by one or more symptoms.

Health score: Integer ranging from 0 to 100 indicating the health of a subservice. A score of 0 means that the subservice is broken, a score of 100 means that the subservice is perfectly operational.

Subservice: Part of an assurance graph that assures a specific feature or subpart of the network system.

Symptom: Reason explaining why a service instance or a subservice is not completely healthy.

2. Introduction

Network Service YANG Modules [RFC8199] describe the configuration, state data, operations, and notifications of abstract representations of services implemented on one or multiple network elements.

Quoting RFC8199: "Network Service YANG Modules describe the characteristics of a service, as agreed upon with consumers of that service. That is, a service module does not expose the detailed configuration parameters of all participating network elements and features but describes an abstract model that allows instances of the service to be decomposed into instance data according to the Network Element YANG Modules of the participating network elements. The service-to-element decomposition is a separate process; the details depend on how the network operator chooses to realize the service. For the purpose of this document, the term "orchestrator" is used to describe a system implementing such a process."

In other words, service configuration orchestrators deploy Network Service YANG Modules through the configuration of Network Element YANG Modules. Network configuration is based on those YANG data models, with protocol/encoding such as NETCONF/XML [RFC6241], RESTCONF/JSON [RFC8040], gNMI/gRPC/protobuf, etc. Knowing that a configuration is applied doesn't imply that the service is running correctly (for example the service might be degraded because of a failure in the network), the network operator must monitor the service operational data at the same time as the configuration. The industry has been standardizing on telemetry to push network element performance information.

A network administrator needs to monitor her network and services as a whole, independently of the use cases or the management protocols. With different protocols come different data models, and different ways to model the same type of information. When network administrators deal with multiple protocols, the network management must perform the difficult and time-consuming job of mapping data models: the model used for configuration with the model used for monitoring. This problem is compounded by a large, disparate set of data sources (MIB modules, YANG models [RFC7950], IPFIX information elements [RFC7011], syslog plain text [RFC3164], TACACS+ [RFC8907], RADIUS [RFC2865], etc.). In order to avoid this data model mapping, the industry converged on model-driven telemetry to stream the service operational data, reusing the YANG models used for configuration. Model-driven telemetry greatly facilitates the notion of closed-loop automation whereby events from the network drive remediation changes back into the network.

However, it proves difficult for network operators to correlate the service degradation with the network root cause. For example, why does my L3VPN fail to connect? Why is this specific service slow? The reverse, i.e. which services are impacted when a network component fails or degrades, is even more interesting for the operators. For example, which service(s) is(are) impacted when this specific optic dBm begins to degrade? Which application is impacted by this ECMP imbalance? Is that issue actually impacting any other customers?

Intent-based approaches are often declarative, starting from a statement of the "The service works correctly" and trying to enforce it. Such approaches are mainly suited for greenfield deployments.

Instead of approaching intent from a declarative way, this framework focuses on already defined services and tries to infer the meaning of "The service works correctly". To do so, the framework works from an assurance graph, deduced from the service definition and from the network configuration. This assurance graph is decomposed into components, which are then assured independently. The root of the assurance graph represents the service to assure, and its children represent components identified as its direct dependencies; each component can have dependencies as well. The SAIN architecture maintains the correct assurance graph when services are modified or when the network conditions change.

When a service is degraded, the framework will highlight where in the assurance service graph to look, as opposed to going hop by hop to troubleshoot the issue. Not only can this framework help to correlate service degradation with network root cause/symptoms, but it can deduce from the assurance graph the number and type of services impacted by a component degradation/failure. This added value informs the operational team where to focus its attention for maximum return.

This architecture provides the building blocks to assure both physical and virtual entities and is flexible with respect to services and subservices, of (distributed) graphs, and of components (Section 3.8).

3. Architecture

SAIN aims at assuring that service instances are correctly running. The goal of SAIN is to assure that service instances are operating correctly and if not, to pinpoint what is wrong. More precisely, SAIN computes a score for each service instance and outputs symptoms explaining that score, especially why the score is not maximal. The score augmented with the symptoms is called the health status.

The SAIN architecture is a generic architecture, applicable to multiple environments. Obviously wireline but also wireless, including 5G, virtual infrastructure manager (VIM), and even virtual functions. Thanks to the distributed graph design principle, graphs from different environments/orchestrator can be combined together.

As an example of a service, let us consider a point-to-point L2VPN connection (i.e. pseudowire). Such a service would take as parameters the two ends of the connection (device, interface or subinterface, and address of the other end) and configure both devices (and maybe more) so that a L2VPN connection is established between the two devices. Examples of symptoms might be "Interface has high error rate" or "Interface flapping", or "Device almost out of memory".

To compute the health status of such as service, the service is decomposed into an assurance graph formed by subservices linked through dependencies. Each subservice is then turned into an expression graph that details how to fetch metrics from the devices and compute the health status of the subservice. The subservice expressions are combined according to the dependencies between the subservices in order to obtain the expression graph which computes the health status of the service.

The overall architecture of our solution is presented in Figure 1. Based on the service configuration, the SAIN orchestrator deduces the assurance graph. It then sends to the SAIN agents the assurance graph along some other configuration options. The SAIN agents are responsible for building the expression graph and computing the health statuses in a distributed manner. The collector is in charge of collecting and displaying the current inferred health status of the service instances and subservices. Finally, the automation loop is closed by having the SAIN Collector providing feedback to the network orchestrator.

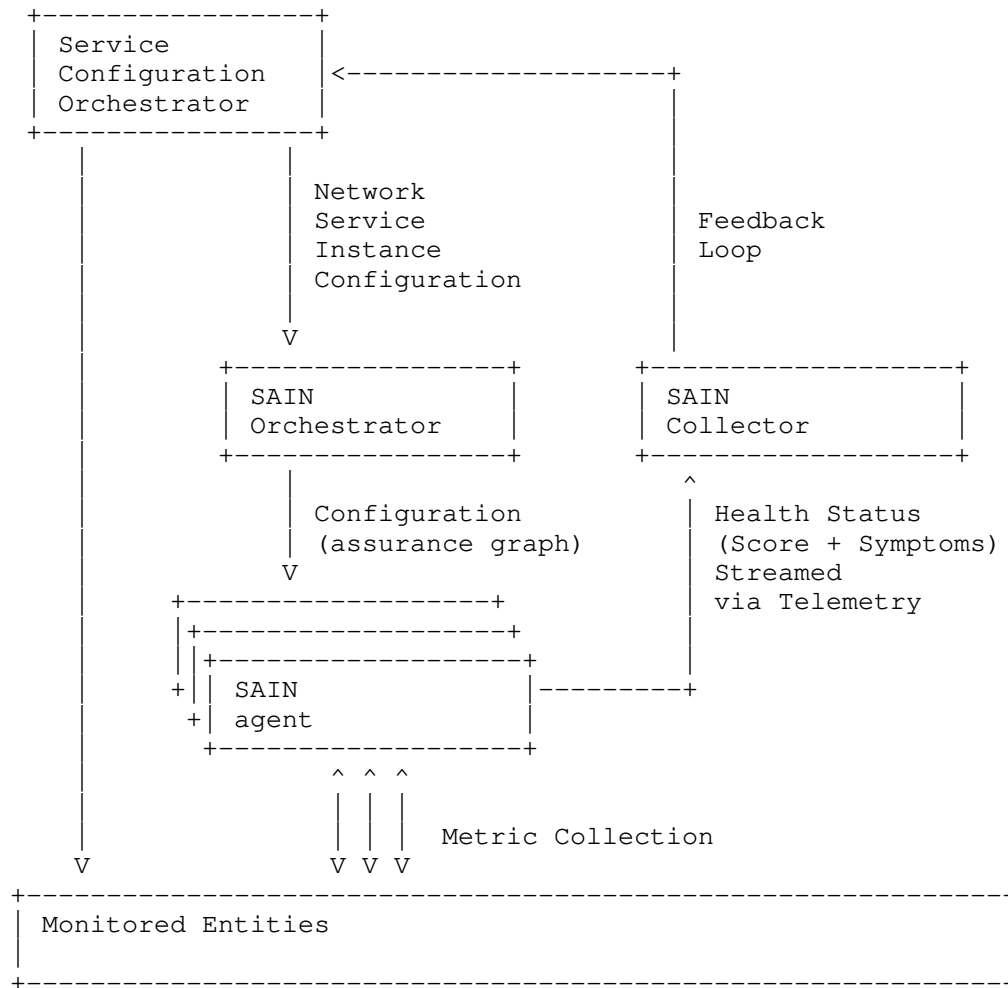


Figure 1: SAIN Architecture

In order to produce the score assigned to a service instance, the architecture performs the following tasks:

- o Analyze the configuration pushed to the network device(s) for configuring the service instance and decide: which information is needed from the device(s), such a piece of information being called a metric, which operations to apply to the metrics for computing the health status.

- o Stream (via telemetry [RFC8641]) operational and config metric values when possible, else continuously poll.
- o Continuously compute the health status of the service instances, based on the metric values.

3.1. Decomposing a Service Instance Configuration into an Assurance Graph

In order to structure the assurance of a service instance, the service instance is decomposed into so-called subservice instances. Each subservice instance focuses on a specific feature or subpart of the network system.

The decomposition into subservices is an important function of this architecture, for the following reasons.

- o TThe result of this decomposition provides a relational picture of a service instance, that can be represented as a graph (called assurance graph) to the operator.
- o Subservices provide a scope for particular expertise and thereby enable contribution from external experts. For instance, the subservice dealing with the optics health should be reviewed and extended by an expert in optical interfaces.
- o Subservices that are common to several service instances are reused for reducing the amount of computation needed.

The assurance graph of a service instance is a DAG representing the structure of the assurance case for the service instance. The nodes of this graph are service instances or subservice instances. Each edge of this graph indicates a dependency between the two nodes at its extremities: the service or subservice at the source of the edge depends on the service or subservice at the destination of the edge.

Figure 2 depicts a simplistic example of the assurance graph for a tunnel service. The node at the top is the service instance, the nodes below are its dependencies. In the example, the tunnel service instance depends on the peer1 and peer2 tunnel interfaces, which in turn depend on the respective physical interfaces, which finally depend on the respective peer1 and peer2 devices. The tunnel service instance also depends on the IP connectivity that depends on the IS-IS routing protocol.

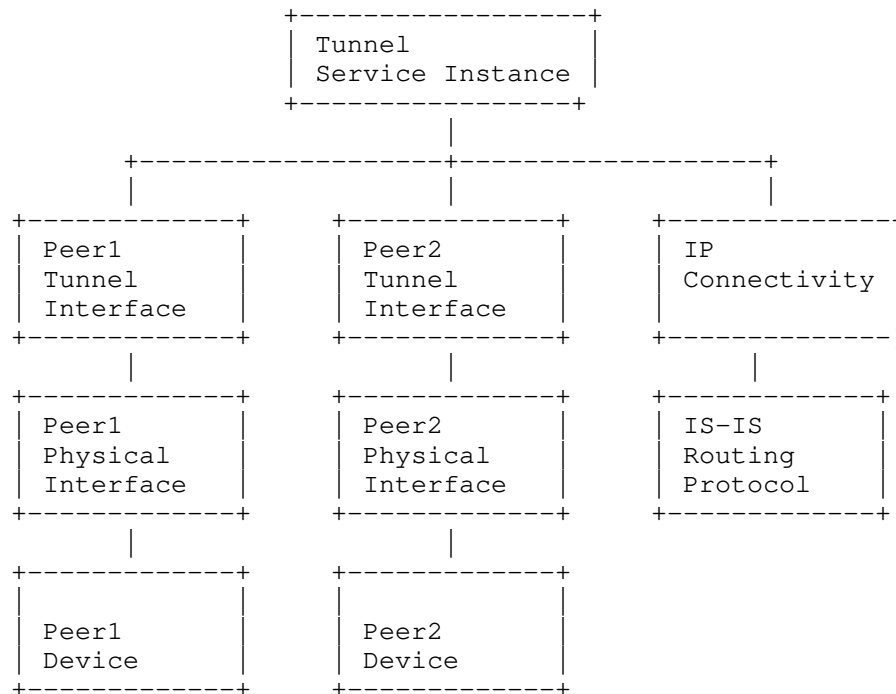


Figure 2: Assurance Graph Example

Depicting the assurance graph helps the operator to understand (and assert) the decomposition. The assurance graph shall be maintained during normal operation with addition, modification and removal of service instances. A change in the network configuration or topology shall be reflected in the assurance graph. As a first example, a change of routing protocol from IS-IS to OSPF would change the assurance graph accordingly. As a second example, assuming that ECMP is in place for the source router for that specific tunnel; in that case, multiple interfaces must now be monitored, on top of the monitoring the ECMP health itself.

3.2. Intent and Assurance Graph

The SAIN orchestrator analyzes the configuration of a service instance to:

- o Try to capture the intent of the service instance, i.e. what is the service instance trying to achieve,
- o Decompose the service instance into subservices representing the network features on which the service instance relies.

The SAIN orchestrator must be able to analyze configuration from various devices and produce the assurance graph.

To schematize what a SAIN orchestrator does, assume that the configuration for a service instance touches 2 devices and configure on each device a virtual tunnel interface. Then:

- o Capturing the intent would start by detecting that the service instance is actually a tunnel between the two devices, and stating that this tunnel must be functional. This is the current state of SAIN, however it does not completely capture the intent which might additionally include, for instance, on the latency and bandwidth requirements of this tunnel.
- o Decomposing the service instance into subservices would result in the assurance graph depicted in Figure 2, for instance.

In order for SAIN to be applied, the configuration necessary for each service instance should be identifiable and thus should come from a "service-aware" source. While the Figure 1 makes a distinction between the SAIN orchestrator and a different component providing the service instance configuration, in practice those two components are mostly likely combined. The internals of the orchestrator are currently out of scope of this document.

3.3. Subservices

A subservice corresponds to subpart or a feature of the network system that is needed for a service instance to function properly. In the context of SAIN, subservice is actually a shortcut for subservice assurance, that is the method for assuring that a subservice behaves correctly.

Subservices, just as with services, have high-level parameters that specify the type and specific instance to be assured. For example, assuring a device requires the specific deviceId as parameter. For example, assuring an interface requires the specific combination of deviceId and interfaceId.

A subservice is also characterized by a list of metrics to fetch and a list of computations to apply to these metrics in order to infer a health status.

3.4. Building the Expression Graph from the Assurance Graph

From the assurance graph is derived a so-called global computation graph. First, each subservice instance is transformed into a set of subservice expressions that take metrics and constants as input (i.e.

sources of the DAG) and produce the status of the subservice, based on some heuristics. Then for each service instance, the service expressions are constructed by combining the subservice expressions of its dependencies. The way service expressions are combined depends on the dependency types (impacting or informational). Finally, the global computation graph is built by combining the service expressions. In other words, the global computation graph encodes all the operations needed to produce health statuses from the collected metrics.

Subservices shall be device independent. To justify this, let's consider the interface operational status. Depending on the device capabilities, this status can be collected by an industry-accepted YANG module (IETF, Openconfig), by a vendor-specific YANG module, or even by a MIB module. If the subservice was dependent on the mechanism to collect the operational status, then we would need multiple subservice definitions in order to support all different mechanisms. This also implies that, while waiting for all the metrics to be available via standard YANG modules, SAIN agents might have to retrieve metric values via non-standard YANG models, via MIB modules, Command Line Interface (CLI), etc., effectively implementing a normalization layer between data models and information models.

In order to keep subservices independent from metric collection method, or, expressed differently, to support multiple combinations of platforms, OSes, and even vendors, the framework introduces the concept of "metric engine". The metric engine maps each device-independent metric used in the subservices to a list of device-specific metric implementations that precisely define how to fetch values for that metric. The mapping is parameterized by the characteristics (model, OS version, etc.) of the device from which the metrics are fetched.

3.5. Building the Expression from a Subservice

Additionally, to the list of metrics, each subservice defines a list of expressions to apply on the metrics in order to compute the health status of the subservice. The definition or the standardization of those expressions (also known as heuristic) is currently out of scope of this standardization.

3.6. Open Interfaces with YANG Modules

The interfaces between the architecture components are open thanks to the YANG modules specified in YANG Modules for Service Assurance [I-D.claise-opsawg-service-assurance-yang]; they specify objects for assuring network services based on their decomposition into so-called subservices, according to the SAIN architecture.

This module is intended for the following use cases:

- o Assurance graph configuration:
 - * Subservices: configure a set of subservices to assure, by specifying their types and parameters.
 - * Dependencies: configure the dependencies between the subservices, along with their types.
- o Assurance telemetry: export the health status of the subservices, along with the observed symptoms.

3.7. Handling Maintenance Windows

Whenever network components are under maintenance, the operator want to inhibit the emission of symptoms from those components. A typical use case is device maintenance, during which the device is not supposed to be operational. As such, symptoms related to the device health should be ignored, as well as symptoms related to the device-specific subservices, such as the interfaces, as their state changes is probably the consequence of the maintenance.

To configure network components as "under maintenance" in the SAIN architecture, the ietf-service-assurance model proposed in [I-D.claise-opsawg-service-assurance-yang] specifies an "under-maintenance" flag per service or subservice instance. When this flag is set and only when this flag is set, the companion field "maintenance-contact" must be set to a string that identifies the person or process who requested the maintenance. Any symptom produced by a service or subservice under maintenance, or by one of its dependencies MUST NOT be reported. A service or subservice under maintenance MAY propagate a symptom "Under Maintenance" towards services or subservices that depend on it.

We illustrate this mechanism on three independent examples based on the assurance graph depicted in Figure 2:

- o Device maintenance, for instance upgrading the device OS. The operator sets the "under-maintenance" flag for the subservice "Peer1" device. This inhibits the emission of symptoms from "Peer1 Physical Interface", "Peer1 Tunnel Interface" and "Tunnel Service Instance". All other subservices are unaffected.
- o Interface maintenance, for instance replacing a broken optic. The operator sets the "under-maintenance" flag for the subservice "Peer1 Physical Interface". This inhibits the emission of

symptoms from "Peer 1 Tunnel Interface" and "Tunnel Service Instance". All other subservices are unaffected.

- o Routing protocol maintenance, for instance modifying parameters or redistribution. The operator sets the "under-maintenance" flag for the subservice "IS-IS Routing Protocol". This inhibits the emission of symptoms from "IP connectivity" and "Tunnel Service Instance". All other subservices are unaffected.

3.8. Flexible Architecture

The SAIN architecture is flexible in terms of components. While the SAIN architecture in Figure 1 makes a distinction between two components, the SAIN configuration orchestrator and the SAIN orchestrator, in practice those two components are mostly likely combined. Similarly, the SAIN agents are displayed in Figure 1 as being separate components. Practically, the SAIN agents could be either independent components or directly integrated in monitored entities. A practical example is an agent in a router.

The SAIN architecture is also flexible in terms of services and subservices. Most examples in this document deal with the notion of Network Service YANG modules, with well known service such as L2VPN or tunnels. However, the concepts of services is general enough to cross into different domains. One of them is the domain of service management on network elements, with also requires its own assurance. Examples includes a DHCP server on a linux server, a data plane, an IPFIX export, etc. The notion of "service" is generic in this architecture. Indeed, a configured service can itself be a service for someone else. Exactly like an DHCP server/ data plane/IPFIX export can be considered as services for a device, exactly like an routing instance can be considered as a service for a L3VPN, exactly like a tunnel can considered as a service for an application in the cloud. The assurance graph is created to be flexible and open, regardless of the subservice types, locations, or domains.

The SAIN architecture is also flexible in terms of distributed graphs. As shown in Figure 1, our architecture comprises several agents. Each agent is responsible for handling a subgraph of the assurance graph. The collector is responsible for fetching the subgraphs from the different agents and gluing them together. As an example, in the graph from Figure 2, the subservices relative to Peer 1 might be handled by a different agent than the subservices relative to Peer 2 and the Connectivity and IS-IS subservices might be handled by yet another agent. The agents will export their partial graph and the collector will stitch them together as dependencies of the service instance.

And finally, the SAIN architecture is flexible in terms of what it monitors. Most, if not all examples, in this document refer to physical components but this is not a constrain. Indeed, the assurance of virtual components would follow the same principles and an assurance graph composed of virtualized components (or a mix of virtualized and physical ones) is well possible within this architecture.

3.9. Timing

The SAIN architecture requires the Network Time Protocol (NTP) [RFC5905] between all elements: monitored entities, SAIN agents, Service Configuration Orchestrator, the SAIN Collector, as well as the SAIN Orchestrator. This guarantees the correlations of all symptoms in the system, correlated with the right assurance graph version.

The SAIN agent might have to remove some symptoms for specific subservice symptoms, because there are outdated and not relevant any longer, or simply because the SAIN agent needs to free up some space. Regardless of the reason, it's important for a SAIN collector (re-)connecting to a SAIN agent to understand the effect of this garbage collection. Therefore, the SAIN agent contains a YANG object specifying the date and time at which the symptoms history starts for the subservice instances.

3.10. New Assurance Graph Generation

The assurance graph will change along the time, because services and subservices come and go (changing the dependencies between subservices), or simply because a subservice is now under maintenance. Therefore an assurance graph version must be maintained, along with the date and time of its last generation. The date and time of a particular subservice instance (again dependencies or under maintenance) might be kept. From a client point of view, an assurance graph change is triggered by the value of the assurance-graph-version and assurance-graph-last-change YANG leaves. At that point in time, the client (collector) follows the following process:

- o Keep the previous assurance-graph-last-change value (let's call it time T)
- o Run through all subservice instance and process the subservice instances for which the last-change is newer than the time T
- o Keep the new assurance-graph-last-change as the new referenced date and time

4. Security Considerations

The SAIN architecture helps operators to reduce the mean time to detect and mean time to repair. As such, it should not cause any security threats. However, the SAIN agents must be secure: a compromised SAIN agents could be sending wrong root causes or symptoms to the management systems.

Except for the configuration of telemetry, the agents do not need "write access" to the devices they monitor. This configuration is applied with a YANG module, whose protection is covered by Secure Shell (SSH) [RFC6242] for NETCONF or TLS [RFC8446] for RESTCONF.

The data collected by SAIN could potentially be compromising to the network or provide more insight into how the network is designed. Considering the data that SAIN requires (including CLI access in some cases), one should weigh data access concerns with the impact that reduced visibility will have on being able to rapidly identify root causes.

If a closed loop system relies on this architecture then the well known issue of those system also applies, i.e., a lying device or compromised agent could trigger partial reconfiguration of the service or network. The SAIN architecture neither augments or reduces this risk.

5. IANA Considerations

This document includes no request to IANA.

6. Contributors

- o Youssef El Fathi
- o Eric Vyncke

7. Open Issues

Refer to the Intent-based Networking NMRG documents

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [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>>.

8.2. Informative References

- [I-D.claise-opsawg-service-assurance-yang] Claise, B. and J. Quilbeuf, "Service Assurance for Intent-based Networking Architecture", February 2020.
- [RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <<https://www.rfc-editor.org/info/rfc2865>>.
- [RFC3164] Lonvick, C., "The BSD Syslog Protocol", RFC 3164, DOI 10.17487/RFC3164, August 2001, <<https://www.rfc-editor.org/info/rfc3164>>.
- [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>>.
- [RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <<https://www.rfc-editor.org/info/rfc7011>>.
- [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>>.
- [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>>.

- [RFC8199] Bogdanovic, D., Claise, B., and C. Moberg, "YANG Module Classification", RFC 8199, DOI 10.17487/RFC8199, July 2017, <<https://www.rfc-editor.org/info/rfc8199>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <<https://www.rfc-editor.org/info/rfc8641>>.
- [RFC8907] Dahm, T., Ota, A., Medway Gash, D., Carrel, D., and L. Grant, "The Terminal Access Controller Access-Control System Plus (TACACS+) Protocol", RFC 8907, DOI 10.17487/RFC8907, September 2020, <<https://www.rfc-editor.org/info/rfc8907>>.

Appendix A. Changes between revisions

v02 - v03

- o Timing Concepts
- o New Assurance Graph Generation

v01 - v02

- o Handling maintenance windows
- o Flexible architecture better explained
- o Improved the terminology
- o Notion of mapping information model to data model, while waiting for YANG to be everywhere
- o Started a security considerations section

v00 - v01

- o Terminology clarifications
- o Figure 1 improved

Acknowledgements

The authors would like to thank Stephane Litkowski, Charles Eckel, Rob Wilton, Vladimir Vassiliev, Gustavo Albuquerque, Stefan Vallin, and Eric Vyncke for their reviews and feedback.

Authors' Addresses

Benoit Claise
Huawei

Email: benoit.claise@huawei.com

Jean Quilbeuf
Independent

Email: jean@quilbeuf.net

Diego R. Lopez
Telefonica I+D
Don Ramon de la Cruz, 82
Madrid 28006
Spain

Email: diego.r.lopez@telefonica.com

Dan Voyer
Bell Canada
Canada

Email: daniel.voyer@bell.ca

Thangam Arumugam
Cisco Systems, Inc.
Milpitas (California)
United States of America

Email: tarumuga@cisco.com

OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: October 25, 2021

B. Claise
Huawei
J. Quilbeuf
Independent
P. Lucente
NTT
P. Fasano
TIM S.p.A
T. Arumugam
Cisco Systems, Inc.
April 23, 2021

YANG Modules for Service Assurance
draft-claise-opsawg-service-assurance-yang-07

Abstract

This document proposes YANG modules for the Service Assurance for Intent-based Networking Architecture.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 25, 2021.

Copyright Notice

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

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

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

Table of Contents

1. Terminology	3
2. Introduction	3
3. YANG Models Overview	3
4. Base ietf-service-assurance YANG module	4
4.1. Tree View	4
4.2. Concepts	5
4.3. YANG Module	6
5. Subservice Extension: ietf-service-assurance-device YANG module	13
5.1. Tree View	13
5.2. Complete Tree View	13
5.3. Concepts	14
5.4. YANG Module	15
6. Subservice Extension: ietf-service-assurance-interface YANG module	16
6.1. Tree View	16
6.2. Complete Tree View	17
6.3. Concepts	18
6.4. YANG Module	18
7. Vendor-specific Subservice Extension: example-service-assurance-device-acme YANG module	19
7.1. Tree View	19
7.2. Complete Tree View	20
7.3. Concepts	21
7.4. YANG Module	22
8. Security Considerations	23
9. IANA Considerations	24
9.1. The IETF XML Registry	24
9.2. The YANG Module Names Registry	25
10. Open Issues	25
11. References	25
11.1. Normative References	25
11.2. Informative References	26
Appendix A. Changes between revisions	26
Acknowledgements	27
Authors' Addresses	27

1. Terminology

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.

The terms used in this document are defined in draft-claise-opsawg-service-assurance-architecture IETF draft.

2. Introduction

The "Service Assurance for Intent-based Networking Architecture" draft-claise-opsawg-service-assurance-architecture, specifies the framework and all of its components for service assurance. This document complements the architecture by providing open interfaces between components. More specifically, the goal is to provide YANG modules for the purpose of service assurance in a format that is:

- o machine readable
- o vendor independent
- o augmentable

3. YANG Models Overview

The main YANG module, ietf-service-assurance, defines objects for assuring network services based on their decomposition into so-called subservices. The subservices are hierarchically organised by dependencies. The subservices, along with the dependencies, constitute an assurance graph. This module should be supported by an agent, able to interact with the devices in order to produce a health status and symptoms for each subservice in the assurance graph. This module is intended for the following use cases:

- o Assurance graph configuration:
 - * Subservices: configure a set of subservices to assure, by specifying their types and parameters.
 - * Dependencies: configure the dependencies between the subservices, along with their type.
- o Assurance telemetry: export the health status of the subservices, along with the observed symptoms.

The second YANG module, `ietf-service-assurance-device`, extends the `ietf-service-assurance` module to add support for the subservice `DeviceHealthy`. Additional subservice types might be added the same way.

The third YANG module, `example-service-assurance-device-acme`, extends the `ietf-service-assurance-device` module as an example to add support for the subservice `DeviceHealthy`, with specifics for the fictional ACME Corporation. Additional vendor-specific parameters might be added the same way.

4. Base `ietf-service-assurance` YANG module

4.1. Tree View

The following tree diagram [RFC8340] provides an overview of the `ietf-service-assurance` data model.

```

module: ietf-service-assurance
+--ro assurance-graph-version      yang:counter32
+--ro assurance-graph-last-change  yang:date-and-time
+--rw subservices
  +--rw subservice* [type id]
    +--rw type                    identityref
    +--rw id                      string
    +--ro last-change?            yang:date-and-time
    +--ro label?                  string
    +--rw under-maintenance?      boolean
    +--rw maintenance-contact     string
    +--rw (parameter)?
      +--:(service-instance-parameter)
        +--rw service-instance-parameter
          +--rw service?          string
          +--rw instance-name?    string
    +--ro health-score?           uint8
    +--ro symptoms-history-start? yang:date-and-time
    +--rw symptoms
      +--ro symptom* [start-date-time id]
        +--ro id                  string
        +--ro health-score-weight? uint8
        +--ro description?        string
        +--ro start-date-time     yang:date-and-time
        +--ro stop-date-time?     yang:date-and-time
    +--rw dependencies
      +--rw dependency* [type id]
        +--rw type                -> /subservices/subservice/type
        +--rw id                  -> /subservices/subservice[type=current()]/
        +--rw dependency-type?    identityref
  ..../type]/id

```

4.2. Concepts

The ietf-service-assurance YANG model assumes an identified number of subservices, to be assured independently. A subservice is a feature or a subpart of the network system that a given service instance might depend on. Example of subservices include:

- o DeviceHealthy: whether a device is healthy, and if not, what are the symptoms. Potential symptoms are "CPU overloaded", "Out of RAM", or "Out of TCAM".
- o ConnectivityHealthy: given two IP addresses owned by two devices, what is the quality of the connection between them. Potential symptoms are "No route available" or "ECMP Imbalance".

The first example is a subservice representing a subpart of the network system, while the second is a subservice representing a feature of the network. In both cases, these subservices might depend on other subservices, for instance, the connectivity might depend on a subservice representing the routing mechanism and on a subservice representing ECMP.

The symptoms are listed for each subservice. Each symptom is specified by a unique id and contains a health-score-weight (the impact to the health score incurred by this symptom), a label (text describing what the symptom is), and dates and times at which the symptom was detected and stopped being detected. While the unique id is sufficient as an unique key list, the start-date-time second key help sorting and retrieving relevant symptoms.

The assurance of a given service instance can be obtained by composing the assurance of the subservices that it depends on, via the dependency relations.

In order to declare a subservice MUST provide:

- o A type: identity inheriting of the base identity for subservice,
- o An id: string uniquely identifying the subservice among those with the same identity,
- o Some parameters, which should be specified in an augmenting model, as described in the next sections.

The type and id uniquely identify a given subservice. They are used to indicate the dependencies. Dependencies have types as well. Two types are specified in the model:

- o **Impacting:** such a dependency indicates an impact on the health of the dependent,
- o **Informational:** such a dependency might explain why the dependent has issues but does not impact its health.

To illustrate the difference between "impacting" and "informational", consider the subservice InterfaceHealthy, representing a network interface. If the device to which the network interface belongs goes down, the network interface will transition to a down state as well. Therefore, the dependency of InterfaceHealthy towards DeviceHealthy is "impacting". On the other hand, as a the dependency towards the ECMPLoad subservice, which checks that the load between ECMP remains stable throughout time, is only "informational". Indeed, services might be perfectly healthy even if the load distribution between ECMP changed. However, such an instability might be a relevant symptom for diagnosing the root cause of a problem.

Service instances **MUST** be modeled as a particular type of subservice with two parameters, a type and an instance name. The type is the name of the service defined in the network orchestrator, for instance "point-to-point-l2vpn". The instance name is the name assigned to the particular instance that we are assuring, for instance the name of the customer using that instance.

The "under-maintenance" and "maintenance-contact" flags inhibit the emission of symptoms for that subservice and subservices that depend on them. See Section 3.7 of [draft-claise-opsawg-service-assurance-architecture] for a more detailed discussion.

By specifying service instances and their dependencies in terms of subservices, one defines the whole assurance to apply for them. An assurance agent supporting this model should then produce telemetry in return with, for each subservice: a health-status indicating how healthy the subservice is and when the subservice is not healthy, a list of symptoms explaining why the subservice is not healthy.

4.3. YANG Module

```
<CODE BEGINS> file "ietf-service-assurance@2020-01-13.yang"

module ietf-service-assurance {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-service-assurance";
  prefix service-assurance;

  import ietf-yang-types {
```

```
    prefix yang;
}
```

```
organization
```

```
    "IETF NETCONF (Network Configuration) Working Group";
```

```
contact
```

```
    "WG Web:    <https://datatracker.ietf.org/wg/netconf/>
```

```
    WG List:    <mailto:netconf@ietf.org>
```

```
    Author:     Benoit Claise <mailto:bclaise@cisco.com>
```

```
    Author:     Jean Quilbeuf <mailto:jquilbeu@cisco.com>";
```

```
description
```

```
    "This module defines objects for assuring network services based on
    their decomposition into so-called subservices, according to the SAIN
    (Service Assurance for Intent-based Networking) architecture.
```

The subservices hierarchically organised by dependencies constitute an assurance graph. This module should be supported by an assurance agent, able to interact with the devices in order to produce a health status and symptoms for each subservice in the assurance graph.

This module is intended for the following use cases:

- * Assurance graph configuration:
 - * subservices: configure a set of subservices to assure, by specifying their types and parameters.
 - * dependencies: configure the dependencies between the subservices, along with their type.
- * Assurance telemetry: export the health status of the subservices, along with the observed symptoms.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

Copyright (c)2020 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.

TO DO:

- Better type (IETF or OC) for device-id, interface-id, etc.
- Have a YANG module for IETF and one for OC?";

```
revision 2020-01-13 {
  description
    "Added the maintenance window concept.";
  reference
    "RFC xxxx: Title to be completed";
}

revision 2019-11-16 {
  description
    "Initial revision.";
  reference
    "RFC xxxx: Title to be completed";
}

identity subservice-idty {
  description
    "Root identity for all subservice types.";
}

identity service-instance-idty {
  base subservice-idty;
  description
    "Identity representing a service instance.";
}

identity dependency-type {
  description
    "Base identity for representing dependency types.";
}

identity informational-dependency {
  base dependency-type;
  description
    "Indicates that symptoms of the dependency might be of interest for the
    dependent, but the status of the dependency should not have any
    impact on the dependent.";
}

identity impacting-dependency {
  base dependency-type;
  description
    "Indicates that the status of the dependency directly impacts the status
    of the dependent.";
}
```

```
grouping symptom {
  description
    "Contains the list of symptoms for a specific subservice.";
  leaf id {
    type string;
    description
      "A unique identifier for the symptom.";
  }
  leaf health-score-weight {
    type uint8 {
      range "0 .. 100";
    }
    description
      "The weight to the health score incurred by this symptom. The higher the
       value, the more of an impact this symptom has. If a subservice health
       score is not 100, there must be at least one symptom with a health
       score weight larger than 0.";
  }
  leaf description {
    type string;
    description
      "Description of the symptom, i.e. text describing what the symptom is, to
       be computer-consumable and be displayed on a human interface. ";
  }
  leaf start-date-time {
    type yang:date-and-time;
    description
      "Date and time at which the symptom was detected.";
  }
  leaf stop-date-time {
    type yang:date-and-time;
    description
      "Date and time at which the symptom stopped being detected.";
  }
}

grouping subservice-dependency {
  description
    "Represent a dependency to another subservice.";
  leaf type {
    type leafref {
      path "/subservices/subservice/type";
    }
    description
      "The type of the subservice to refer to (e.g. DeviceHealthy).";
  }
  leaf id {
    type leafref {
```

```

        path "/subservices/subservice[type=current()/../type]/id";
    }
    description
        "The identifier of the subservice to refer to.";
}
leaf dependency-type {
    type identityref {
        base dependency-type;
    }
    description
        "Represents the type of dependency (i.e. informational, impacting).";
}
// augment here if more info are needed (i.e. a percentage) depending on the
// dependency type.
}

leaf assurance-graph-version {
    type yang:counter32;
    mandatory true;
    config false;
    description
        "The assurance graph version, which increases by 1 for each new version, af
        ter the changes
        (dependencies and/or maintenance windows parameters) are applied to the su
        bservice(s).";
}
leaf assurance-graph-last-change {
    type yang:date-and-time;
    mandatory true;
    config false;
    description
        "Date and time at which the assurance graph last changed after the changes
        (dependencies
        and/or maintenance windows parameters) are applied to the subservice(s). T
        hese date and time
        must be more recent or equal compared to the more recent value of any chan
        ged subservices
        last-change";
}
container subservices {
    description
        "Root container for the subservices.";
    list subservice {
        key "type id";
        description
            "List of subservice configured.";
        leaf type {
            type identityref {
                base subservice-idty;
            }
            description
                "Name of the subservice, e.g. DeviceHealthy.";
        }
        leaf id {

```



```

    type string;
    description
        "Unique identifier of the subservice instance, for each type.";
}
leaf last-change {
    type yang:date-and-time;
    config false;
    description
        "Date and time at which the assurance graph for this subservice
        instance last changed, i.e. dependencies and/or maintenance windows pa
rameters.";
}
leaf label {
    type string;
    config false;
    description
        "Label of the subservice, i.e. text describing what the subservice is t
o
        be displayed on a human interface.";
}
leaf under-maintenance {
    type boolean;
    default false;
    description
        "An optional flag indicating whether this particular subservice is unde
r
        maintenance. Under this circumstance, the subservice symptoms and the
        symptoms of its dependencies in the assurance graph should not be taken
        into account. Instead, the subservice should send a 'Under Maintenance'
        single symptom.

        The operator changing the under-maintenance value must set the
        maintenance-contact variable.

        When the subservice is not under maintenance any longer, the
        under-maintenance flag must return to its default value and
        the under-maintenance-owner variable deleted.";
}
leaf maintenance-contact {
    when "../under-maintenance = 'true'";
    type string;
    mandatory true;
    description
        "A string used to model an administratively assigned name of the
        resource that changed the under-maintenance value to 'true'.

        It is suggested that this name contain one or more of the following:
        IP address, management station name, network manager's name, location,
        or phone number. In some cases the agent itself will be the owner of
        an entry. In these cases, this string shall be set to a string
        starting with 'monitor'.";

```

```

    }
    choice parameter {
        description
            "Specify the required parameters per subservice type.";
        container service-instance-parameter {
            when "derived-from-or-self(..../type, 'service-assurance:service-instance
-idty')";
            description
                "Specify the parameters of a service instance.";
            leaf service {
                type string;
                description "Name of the service.";
            }
            leaf instance-name {
                type string;
                description "Name of the instance for that service.";
            }
        }
        // Other modules can augment their own cases into here
    }
    leaf health-score {
        type uint8 {
            range "0 .. 100";
        }
        config false;
        description
            "Score value of the subservice health. A value of 100 means that
            subservice is healthy. A value of 0 means that the subservice is
            broken. A value between 0 and 100 means that the subservice is
            degraded.";
    }
    leaf symptoms-history-start {
        type yang:date-and-time;
        config false;
        description
            "Date and time at which the symptoms history starts for this
            subservice instance, either because the subservice instance
            started at that date and time or because the symptoms before that
            were removed due to a garbage collection process.";
    }
    container symptoms {
        description
            "Symptoms for the subservice.";
        list symptom {
            key "start-date-time id";
            config false;
            description
                "List of symptoms the subservice. While the start-date-time key is no
t
                necessary per se, this would get the entries sorted by start-date-tim
e

```

```

        for easy consumption.";
        uses symptom;
    }
}
container dependencies {
    description
        "configure the dependencies between the subservices, along with their t
ypes.";
    list dependency {
        key "type id";
        description
            "List of soft dependencies of the subservice.";
        uses subservice-dependency;
    }
}
}
}
}

```

<CODE ENDS>

5. Subservice Extension: ietf-service-assurance-device YANG module

5.1. Tree View

The following tree diagram [RFC8340] provides an overview of the ietf-service-assurance-device data model.

```

module: ietf-service-assurance-device
  augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter:
    +--rw device-idty
    +--rw device?   string

```

5.2. Complete Tree View

The following tree diagram [RFC8340] provides an overview of the ietf-service-assurance and ietf-service-assurance-device data models.

```

module: ietf-service-assurance
  +--ro assurance-graph-version          yang:counter32
  +--ro assurance-graph-last-change      yang:date-and-time
  +--rw subservices
    +--rw subservice* [type id]
      +--rw type                          identityref
      +--rw id                            string
      +--ro last-change?                  yang:date-and-time
      +--ro label?                        string
      +--rw under-maintenance?            boolean
      +--rw maintenance-contact           string
      +--rw (parameter)?
        +--:(service-instance-parameter)
          +--rw service-instance-parameter
            +--rw service?                 string
            +--rw instance-name?           string
        +--:(service-assurance-device:device-idty)
          +--rw service-assurance-device:device-idty
            +--rw service-assurance-device:device? string
      +--ro health-score?                  uint8
      +--ro symptoms-history-start?         yang:date-and-time
      +--rw symptoms
        +--ro symptom* [start-date-time id]
          +--ro id                         string
          +--ro health-score-weight?        uint8
          +--ro description?                 string
          +--ro start-date-time              yang:date-and-time
          +--ro stop-date-time?              yang:date-and-time
      +--rw dependencies
        +--rw dependency* [type id]
          +--rw type                       -> /subservices/subservice/type
          +--rw id                         -> /subservices/subservice[type=current()]/
      +--rw dependency-type?               identityref
  ..../type]/id

```

5.3. Concepts

As the number of subservices will grow over time, the YANG module is designed to be extensible. A new subservice type requires the precise specifications of its type and expected parameters. Let us illustrate the example of the new DeviceHealthy subservice type. As the name implies, it monitors and reports the device health, along with some symptoms in case of degradation.

For our DeviceHealthy subservice definition, the new device-idty is specified, as an inheritance from the base identity for subservices. This indicates to the assurance agent that we are now assuring the health of a device.

The typical parameter for the configuration of the DeviceHealthy subservice is the name of the device that we want to assure. By augmenting the parameter choice from ietf-service-assurance YANG module for the case of the device-idty subservice type, this new parameter is specified.

5.4. YANG Module

```
<CODE BEGINS> file "ietf-service-assurance-device@2020-01-13.yang"

module ietf-service-assurance-device {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-service-assurance-device";
  prefix service-assurance-device;

  import ietf-service-assurance {
    prefix "service-assurance";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";
  contact
    "WG Web: <https://datatracker.ietf.org/wg/netconf/>
    WG List: <mailto:netconf@ietf.org>
    Author: Benoit Claise <mailto:bclaise@cisco.com>
    Author: Jean Quilbeuf <mailto:jquilbeu@cisco.com>";
  description
    "This module extends the ietf-service-assurance module to add
    support for the subservice DeviceHealthy.

    Checks whether a network device is healthy.

    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 (RFC 2119)
    (RFC 8174) when, and only when, they appear in all
    capitals, as shown here.

    Copyright (c) 2020 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD License
    set forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (https://trustee.ietf.org/license-info)."
```

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices. ";

```
revision 2020-01-13 {
  description
    "Added the maintenance window concept.";
  reference
    "RFC xxxx: Title to be completed";
}

revision 2019-11-16 {
  description
    "Initial revision.";
  reference
    "RFC xxxx: Title to be completed";
}

identity device-idty {
  base service-assurance:subservice-idty;
  description "Network Device is healthy.";
}

augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter {
  description
    "Specify the required parameters for a new subservice type";
  container device-idty{
    when "derived-from-or-self(..../service-assurance:type, 'device-idty')";
    description
      "Specify the required parameters for the device-idty subservice type";
  }

  leaf device {
    type string;
    description "The device to monitor.";
  }
}
}

<CODE ENDS>
```

6. Subservice Extension: ietf-service-assurance-interface YANG module

6.1. Tree View

The following tree diagram [RFC8340] provides an overview of the ietf-service-assurance-interface data model.

```

module: ietf-service-assurance-interface
  augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter:
    +--rw device?      string
    +--rw interface?   string

```

6.2. Complete Tree View

The following tree diagram [RFC8340] provides an overview of the `ietf-service-assurance`, `ietf-service-assurance-device`, and `ietf-service-assurance-interface` data models.

```

module: ietf-service-assurance
  +--ro assurance-graph-version      yang:counter32
  +--ro assurance-graph-last-change  yang:date-and-time
  +--rw subservices
    +--rw subservice* [type id]
      +--rw type                                identityref
      +--rw id                                  string
      +--ro last-change?                     yang:date-and-time
      +--ro label?                           string
      +--rw under-maintenance?               boolean
      +--rw maintenance-contact              string
      +--rw (parameter)?
        +--:(service-instance-parameter)
          +--rw service-instance-parameter
            +--rw service?                    string
            +--rw instance-name?              string
        +--:(service-assurance-device:device-idty)
          +--rw service-assurance-device:device-idty
            +--rw service-assurance-device:device? string
        +--:(service-assurance-interface:device)
          +--rw service-assurance-interface:device? string
        +--:(service-assurance-interface:interface)
          +--rw service-assurance-interface:interface? string
      +--ro health-score?                   uint8
      +--ro symptoms-history-start?          yang:date-and-time
      +--rw symptoms
        +--rw symptom* [start-date-time id]
          +--ro id                            string
          +--ro health-score-weight?          uint8
          +--ro description?                  string
          +--ro start-date-time               yang:date-and-time
          +--ro stop-date-time?               yang:date-and-time
      +--rw dependencies
        +--rw dependency* [type id]
          +--rw type                          -> /subservices/subservice/type
          +--rw id                            -> /subservices/subservice[type=current()]/
          +--rw dependency-type?              identityref
    ../type]/id

```

6.3. Concepts

For our InterfaceHealthy subservice definition, the new interface-idty is specified, as an inheritance from the base identity for subservices. This indicates to the assurance agent that we are now assuring the health of an interface.

The typical parameters for the configuration of the InterfaceHealthy subservice are the name of the device and, on that specific device, a specific interface. By augmenting the parameter choice from ietf-service-assurance YANG module for the case of the interface-idty subservice type, those two new parameter are specified.

6.4. YANG Module

```
<CODE BEGINS> file "ietf-service-assurance-interface@2020-01-13.yang"

module ietf-service-assurance-interface {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-service-assurance-interface";
  prefix service-assurance-interface;

  import ietf-service-assurance {
    prefix "service-assurance";
  }

  organization
    "IETF OPSAWG Working Group";
  contact
    "WG Web:    <https://datatracker.ietf.org/wg/opsawg/>
    WG List:    <mailto:opsawg@ietf.org>
    Author:     Benoit Claise <mailto:bclaise@cisco.com>
    Author:     Jean Quilbeuf <mailto:jquilbeu@cisco.com>";
  description
    "This module extends the ietf-service-assurance module to add
    support for the subservice InterfaceHealthy.

    Checks whether an interface is healthy.

    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 (RFC 2119)
    (RFC 8174) when, and only when, they appear in all
    capitals, as shown here.

    Copyright (c) 2020 IETF Trust and the persons identified as
    authors of the code. All rights reserved."
```

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices. ";

```
revision 2020-01-13 {
  description
    "Initial revision.";
  reference
    "RFC xxxx: Title to be completed";
}

identity interface-idty {
  base service-assurance:subservice-idty;
  description "Checks whether an interface is healthy.";
}

augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter {
  when "derived-from-or-self(service-assurance:type, 'interface-idty')";
  description
    "Specify the required parameters for the interface-idty subservice type"
;

  leaf device {
    type string;
    description "Device supporting the interface.";
  }
  leaf interface {
    type string;
    description "Name of the interface.";
  }
}
```

<CODE ENDS>

7. Vendor-specific Subservice Extension: example-service-assurance-device-acme YANG module

7.1. Tree View

The following tree diagram [RFC8340] provides an overview of the example-service-assurance-device-acme data model.

```
module: example-service-assurance-device-acme
  augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter:
    +--rw acme-device-idty
      +--rw device?          string
      +--rw acme-specific-parameter?  string
```

7.2. Complete Tree View

The following tree diagram [RFC8340] provides an overview of the `ietf-service-assurance`, `ietf-service-assurance-device`, and `example-service-assurance-device-acme` data models.

```

module: ietf-service-assurance
  +--ro assurance-graph-version      yang:counter32
  +--ro assurance-graph-last-change  yang:date-and-time
  +--rw subservices
    +--rw subservice* [type id]
      +--rw type                                                                ide
  entityref
    +--rw id                                                                    str
  ing
    +--ro last-change?                                                         yan
  g:date-and-time
    +--ro label?                                                                str
  ing
    +--rw under-maintenance?                                                  boo
  lean
    +--rw maintenance-contact                                                 str
  ing
    +--rw (parameter)?
      +--:(service-instance-parameter)
        +--rw service-instance-parameter
          +--rw service?               string
          +--rw instance-name?         string
      +--:(service-assurance-device:device-idty)
        +--rw service-assurance-device:device-idty
          +--rw service-assurance-device:device?   string
      +--:(service-assurance-interface:device)
        +--rw service-assurance-interface:device? str
  ing
      +--:(service-assurance-interface:interface)
        +--rw service-assurance-interface:interface? str
  ing
      +--:(example-service-assurance-device-acme:acme-device-idty)
        +--rw example-service-assurance-device-acme:acme-device-idty
          +--rw example-service-assurance-device-acme:device?
string
      +--rw example-service-assurance-device-acme:acme-specific-parame
ter?  string
      +--ro health-score?                                                       uin
t8
    +--ro symptoms-history-start?                                             yan
  g:date-and-time
    +--rw symptoms
      +--ro symptom* [start-date-time id]
        +--ro id                      string
        +--ro health-score-weight?    uint8
        +--ro description?             string
        +--ro start-date-time          yang:date-and-time
        +--ro stop-date-time?          yang:date-and-time
    +--rw dependencies
      +--rw dependency* [type id]
        +--rw type                    -> /subservices/subservice/type
        +--rw id                      -> /subservices/subservice[type=current()]/
../type]/id
      +--rw dependency-type?          identityref

```

7.3. Concepts

Under some circumstances, vendor-specific subservice types might be

required. As an example of this vendor-specific implementation, this section shows how to augment the ietf-service-assurance-device module

to add support for the subservice DeviceHealthy, specific to the ACME Corporation. The new parameter is acme-specific-parameter.

7.4. YANG Module

```
module example-service-assurance-device-acme {
  yang-version 1.1;
  namespace "urn:example:example-service-assurance-device-acme";
  prefix example-service-assurance-device-acme;

  import ietf-service-assurance {
    prefix "service-assurance";
  }

  import ietf-service-assurance-device {
    prefix "service-assurance-device";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/netconf/>
    WG List:  <mailto:netconf@ietf.org>
    Author:   Benoit Claise  <mailto:bclaise@cisco.com>
    Author:   Jean Quilbeuf  <mailto:jquilbeu@cisco.com>";
  description
    "This module extends the ietf-service-assurance-device module to add
    support for the subservice DeviceHealthy, specific to the ACME Corporation.

    ACME Network Device is healthy.

    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 (RFC 2119)
    (RFC 8174) when, and only when, they appear in all
    capitals, as shown here.

    Copyright (c) 2019 IETF Trust and the persons identified as
    authors of the code.  All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD License
    set forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (https://trustee.ietf.org/license-info).
```

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices. ";

```

revision 2020-01-13 {
  description
    "Added the maintenance window concept.";
  reference
    "RFC xxxx: Title to be completed";
}

revision 2019-11-16 {
  description
    "Initial revision.";
  reference
    "RFC xxxx: Title to be completed";
}

identity device-acme-idty {
  base service-assurance-device:device-idty;
  description "Network Device is healthy.";
}

augment /service-assurance:subservices/service-assurance:subservice/service-assurance:parameter {
  description
    "Specify the required parameters for a new subservice type";
  container acme-device-idty{
    when "derived-from-or-self(..../service-assurance:type, 'device-acme-idty')";
    description
      "Specify the required parameters for the device-acme-idty subservice type";

    leaf device {
      type string;
      description "The device to monitor.";
    }

    leaf acme-specific-parameter {
      type string;
      description "The ACME Corporation sepcific parameter.";
    }
  }
}

```

8. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer

is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The 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.

There are a number of data nodes defined in this YANG module that are writable/ creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

- o /subservices/subservice/type
- o /subservices/subservice/id
- o /subservices/subservice/under-maintenance
- o /subservices/subservice/maintenance-contact

9. IANA Considerations

9.1. The IETF XML Registry

This document registers two URIs in the IETF XML registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-service-assurance
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-service-assurance-device
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-service-assurance-interface
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

9.2. The YANG Module Names Registry

This document registers three YANG modules in the YANG Module Names registry [RFC7950]. Following the format in [RFC7950], the the following registrations are requested:

```
name:      ietf-service-assurance
namespace: urn:ietf:params:xml:ns:yang:ietf-service-assurance
prefix:    inc
reference:  RFC XXXX

name:      ietf-service-assurance-device
namespace: urn:ietf:params:xml:ns:yang:ietf-service-assurance-device
prefix:    inc
reference:  RFC XXXX

name:      ietf-service-assurance-interface
namespace: urn:ietf:params:xml:ns:yang:ietf-service-assurance-interface
prefix:    inc
reference:  RFC XXXX
```

10. Open Issues

-None

11. References

11.1. Normative References

- [draft-claise-opsawg-service-assurance-architecture]
Claise, B. and J. Quilbeuf, "draft-claise-opsawg-service-assurance-architecture", 2020.
- [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>>.
- [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>>.

- [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>>.
- [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>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

11.2. Informative References

- [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>>.
- [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>>.
- [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>>.

Appendix A. Changes between revisions

v04 - v05

- o Added the concept of symptoms-history-start
- o Changed label to description, under symptoms. This was confusing as there was two labels in the models

v03 - v04

- o Add the interface subservice, with two parameters

v02 - v03

- o Added the maintenace window concepts

v01 - v02

- o Improved leaf naming
- o Clarified some concepts: symptoms, dependency

v00 - v01

- o Terminology clarifications
- o Provide example of impacting versus impacted dependencies

Acknowledgements

The authors would like to thank Jan Lindblad for his help during the design of these YANG modules. The authors would like to thank Stephane Litkowski and Charles Eckel for their reviews.

Authors' Addresses

Benoit Claise
Huawei

Email: benoit.claise@huawei.com

Jean Quilbeuf
Independent

Email: jean@quilbeuf.net

Paolo Lucente
NTT
Siriusdreef 70-72
Hoofddorp, WT 2132
Netherlands

Email: paolo@ntt.net

Paolo Fasano
TIM S.p.A
via G. Reiss Romoli, 274
10148 Torino
Italy

Email: paolo2.fasano@telecomitalia.it

Thangam Arumugam
Cisco Systems, Inc.
Milpitas (California)
United States

Email: tarumuga@cisco.com

OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: 14 November 2022

M. Boucadair, Ed.
Orange
O. Gonzalez de Dios, Ed.
S. Barguil
Telefonica
L. Munoz
Vodafone
13 May 2022

A YANG Network Data Model for Layer 2 VPNs
draft-ietf-opsawg-l2nm-16

Abstract

This document defines an L2VPN Network YANG Model (L2NM) which can be used to manage the provisioning of Layer 2 Virtual Private Network services within a network (e.g., service provider network). The L2NM complements the Layer 2 Service Model (L2SM) by providing a network-centric view of the service that is internal to a service provider. The L2NM is particularly meant to be used by a network controller to derive the configuration information that will be sent to relevant network devices.

Also, this document defines a YANG module to manage Ethernet segments and the initial versions of two IANA-maintained modules that defines a set of identities of BGP Layer 2 encapsulation types and pseudowire types.

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: A YANG Network Data Model for Layer 2 VPNs";
- * reference: RFC XXXX

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

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 14 November 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Terminology	4
3. Acronyms and Abbreviations	6
4. Reference Architecture	6
5. Relationship to Other YANG Data Models	10
6. Description of the Ethernet Segment YANG Module	11
7. Description of the L2NM YANG Module	14
7.1. Overall Structure of the Module	14
7.2. VPN Profiles	14
7.3. VPN Services	16
7.4. Global Parameters Profiles	20
7.5. VPN Nodes	23
7.5.1. BGP Auto-Discovery	26
7.5.2. Signaling Options	27
7.5.2.1. BGP	29
7.5.2.2. LDP	30
7.5.2.3. L2TP	31
7.6. VPN Network Accesses	32
7.6.1. Connection	34
7.6.2. EVPN-VPWS Service Instance	37

7.6.3. Ethernet OAM	38
7.6.4. Services	40
8. YANG Modules	45
8.1. IANA-Maintained Module for BGP Layer 2 Encapsulation Types	45
8.2. IANA-Maintained Module for Pseudowire Types	51
8.3. Ethernet Segments	58
8.4. L2NM	66
9. Security Considerations	119
10. IANA Considerations	120
10.1. Registering YANG Modules	121
10.2. BGP Layer 2 Encapsulation Types	122
10.3. Pseudowire Types	122
11. References	123
11.1. Normative References	123
11.2. Informative References	126
Appendix A. Examples	132
A.1. BGP-based VPLS	132
A.2. BGP-based VPWS with LDP Signaling	137
A.3. LDP-based VPLS	141
A.4. VPWS-EVPN Service Instance	145
A.5. Automatic ESI Assignment	151
A.6. VPN Network Access Precedence	154
Acknowledgements	157
Contributors	157
Authors' Addresses	157

1. Introduction

[RFC8466] defines an L2VPN Service Model (L2SM) YANG data model that can be used between customers and service providers for ordering Layer 2 Virtual Private Network (L2VPN) services. This document complements the L2SM by creating a network-centric view of the service: the L2VPN Network Model (L2NM).

The L2NM (Section 8.4) can be exposed, for example, by a network controller to a service controller within the service provider's network. In particular, the model can be used in the communication interface between the entity that interacts directly with the customer (i.e., the service orchestrator) and the entity in charge of network orchestration and control (a.k.a., network controller/orchestrator) by allowing for more network-centric information to be included.

The L2NM supports capabilities, such as exposing operational parameters, transport protocols selection, and precedence. It can also serve as a multi-domain orchestration interface.

The L2NM is scoped for a variety of Layer 2 Virtual Private Networks, such as:

- * Virtual Private LAN Service (VPLS) [RFC4761][RFC4762]
- * Virtual Private Wire Service (VPWS) (Section 3.1.1 of [RFC4664])
- * Various flavors of Ethernet VPNs (EVPNs):
 - VPWS EVPN [RFC8214],
 - Provider Backbone Bridging Ethernet VPNs (PBB EVPNs) [RFC7623],
 - EVPN over MPLS [RFC7432], and
 - EVPN over Virtual eXtensible Local Area Network (VXLAN) [RFC8365].

The L2NM is designed to easily support future Layer 2 VPN flavors and procedures (e.g., advanced configuration such as pseudowires resilience or Multi-Segment pseudowires [RFC7267]). A set of examples to illustrate the use of the L2NM are provided in Appendix A.

Also, this document defines the initial versions of two IANA-maintained modules that define a set of identities of BGP Layer 2 encapsulation types (Section 8.1) and pseudowire types (Section 8.2). Relying upon these IANA-maintained modules is meant to provide more flexibility in handling new types rather than being limited by a set of identities defined in the L2NM itself. Section 8.3 defines a YANG module to manage Ethernet Segments (ESes) that are required for instantiating EVPNs.

This document uses the common Virtual Private Network (VPN) YANG module defined in [RFC9181].

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

2. Terminology

This document assumes that the reader is familiar with [RFC6241], [RFC7950], [RFC8466], [RFC4026], and [RFC8309]. This document uses terminology from those documents.

This document uses the term "network model" as defined in Section 2.1 of [RFC8969].

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

This document makes use of the following terms:

Ethernet segment (ES): Refers to the set of the Ethernet links that

are used by a customer site (device or network) to connect to one or more Provider Edges (PEs).

Layer 2 VPN Service Model (L2SM): Describes the service characterization of an L2VPN that interconnects a set of sites from the customer's perspective. The customer service model does not provide details on the service provider network. An L2VPN customer service model is defined in [RFC8466].

Layer 2 VPN Network Model (L2NM): Refers to the YANG data model that describes an L2VPN service with a network-centric view. It contains information on the service provider network and might include allocated resources. Network controllers can use it to manage the Layer 2 VPN service configuration in the service provider's network. The corresponding YANG module can be used by a service orchestrator to request a VPN service to a network controller or to expose the list of active L2VPN services.

MAC-VRF: Refers to a Virtual Routing and Forwarding (VRF) table for Media Access Control (MAC) addresses on a PE.

Network controller: Denotes a functional entity responsible for the management of the service provider network.

Service orchestrator: Refers to a functional entity that interacts with the customer of an L2VPN relying upon, e.g., the L2SM. The service orchestrator is responsible for the Customer Edge - to Provider Edge (CE-PE) attachment circuits, the PE selection, and requesting the activation of the L2VPN service to a network controller.

Service provider network: Is a network able to provide L2VPN-related services.

VPN node: Is an abstraction that represents a set of policies applied on a PE and belonging to a single VPN service. A VPN service involves one or more VPN nodes. The VPN node will identify the service providers' node on which the VPN is deployed.

VPN network access: Is an abstraction that represents the network interfaces that are associated with a given VPN node. Traffic coming from the VPN network access belongs to the VPN. The attachment circuits (bearers) between Customer Edges (CEs) and Provider Edges (PEs) are terminated in the VPN network access.

VPN service provider: Is a service provider that offers L2VPN-related services.

3. Acronyms and Abbreviations

The following acronyms and abbreviations are used in this document:

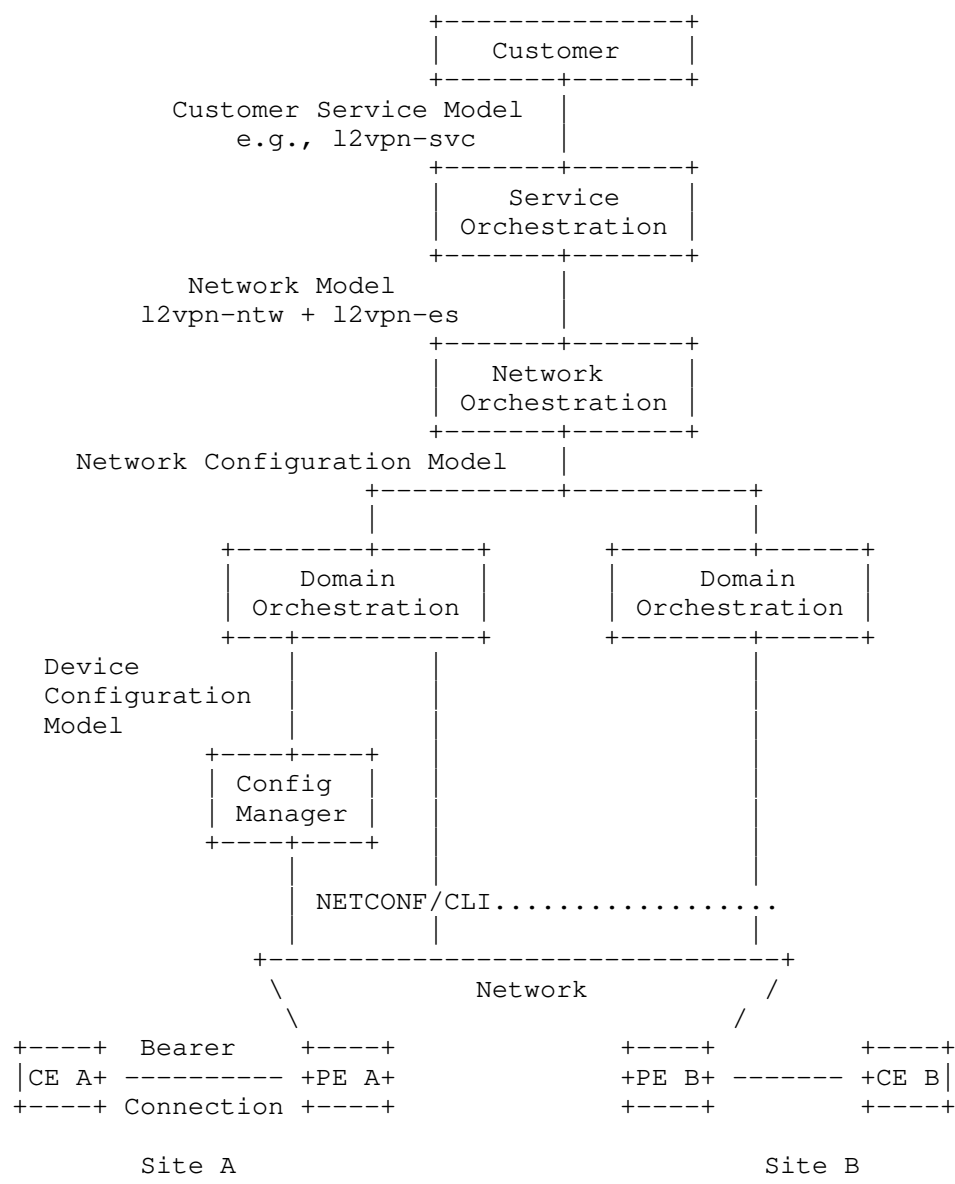
ACL	Access Control List
BGP	Border Gateway Protocol
BUM	Broadcast, unknown unicast, or multicast
CE	Customer Edge
ES	Ethernet Segment
ESI	Ethernet Segment Identifier
EVPN	Ethernet VPN
L2VPN	Layer 2 Virtual Private Network
L2SM	L2VPN Service Model
L2NM	L2VPN Network Model
MAC	Media Access Control
PBB	Provider Backbone Bridging
PCP	Priority Code Point
PE	Provider Edge
QoS	Quality of Service
RD	Route Distinguisher
RT	Route Target
VPLS	Virtual Private LAN Service
VPN	Virtual Private Network
VPWS	Virtual Private Wire Service
VRF	Virtual Routing and Forwarding

4. Reference Architecture

Figure 1 illustrates how the L2NM is used. As a reminder, this figure is an expansion of the architecture presented in Section 3 of [RFC8466] and decomposes the box marked "orchestration" in that figure into three separate functional components called "Service Orchestration", "Network Orchestration", and "Domain Orchestration".

Similar to Section 3 of [RFC8466], CE to PE attachment is achieved through a bearer with a Layer 2 connection on top. The bearer refers to properties of the attachment that are below Layer 2, while the connection refers to Layer 2 protocol-oriented properties.

The reader may refer to [RFC8309] for the distinction between the "Customer Service Model", the "Service Delivery Model", the "Network Configuration Model", and the "Device Configuration Model". The "Domain Orchestration" and "Config Manager" roles may be performed by "SDN Controllers".



NETCONF: Network Configuration Protocol
CLI: Command-Line Interface

Figure 1: L2SM and L2NM Interaction

The customer may use various means to request a service that may trigger the instantiation of an L2NM. The customer may use the L2SM or may rely upon more abstract models to request a service that relies upon an L2VPN service. For example, the customer may supply an IP Connectivity Provisioning Profile (CPP) that characterizes the requested service [RFC7297], an enhanced VPN (VPN+) service [I-D.ietf-teas-enhanced-vpn], or an IETF network slice service [I-D.ietf-teas-ietf-network-slices].

Note also that both the L2SM and the L2NM may be used in the context of the Abstraction and Control of TE Networks (ACTN) framework [RFC8453]. Figure 2 shows the Customer Network Controller (CNC), the Multi-Domain Service Coordinator (MDSC), and the Provisioning Network Controller (PNC).

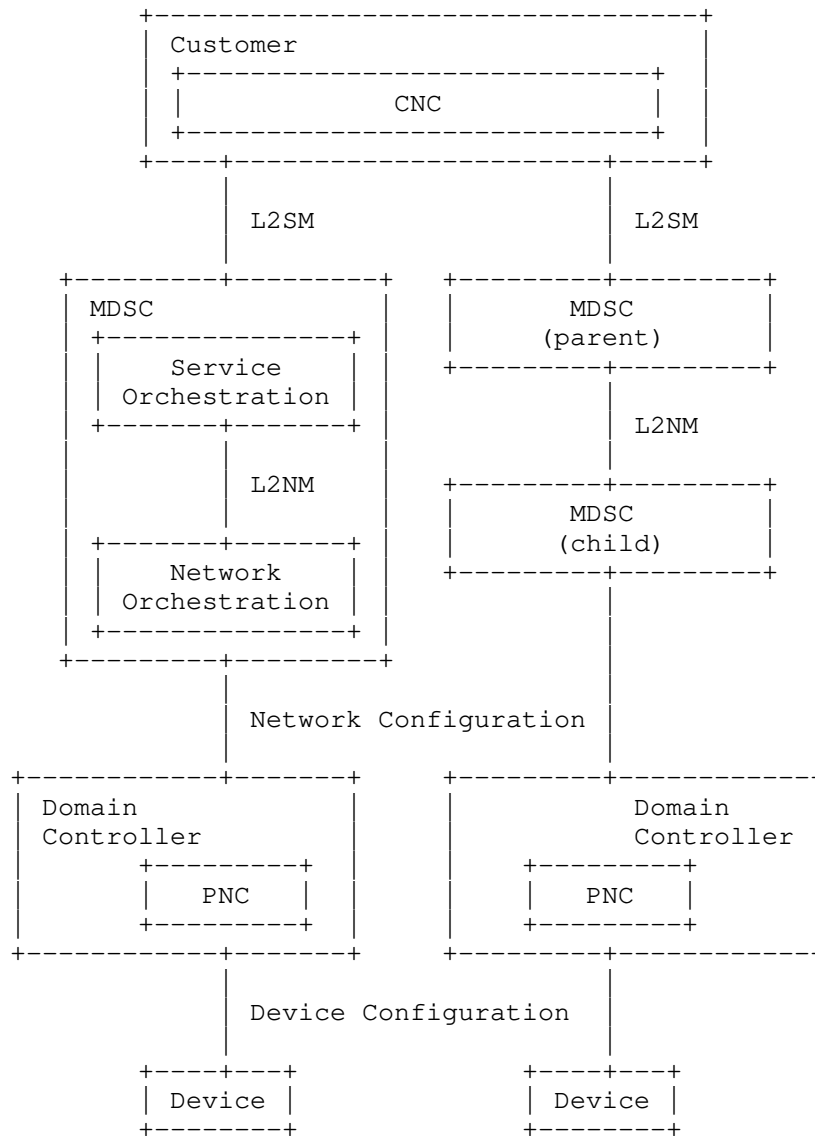


Figure 2: L2SM and L2NM in the Context of ACTN

5. Relationship to Other YANG Data Models

The "ietf-vpn-common" module [RFC9181] 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., [RFC8466]). The L2NM reuses these common types and groupings.

Also, the L2NM uses the IANA-maintained modules "iana-bgp-l2-encaps" (Section 8.1) and "iana-pseudowire-types" (Section 8.2) to identify a Layer 2 encapsulation type. More details are provided in Sections 7.5.2.1 and 7.5.2.3.

For the particular case of EVPN, the L2NM includes a name that refers to an Ethernet segment that is created using the "ietf-ethernet-segment" module (Section 8.3). Some ES-related examples are provided in Appendices A.4 and A.5.

As discussed in Section 4, the L2NM is used to manage L2VPN services within a service provider network. The module provides a network view of the L2VPN service. Such a view is only visible within the service provider and is not exposed outside (to customers, for example). The following discusses how the L2NM interfaces with other YANG modules:

L2SM: The L2NM is not a customer service model.

The internal view of the service (i.e., the L2NM) may be mapped to an external view which is visible to customers: L2VPN Service Model (L2SM) [RFC8466].

The L2NM can be fed with inputs that are requested by customers, typically, relying upon an L2SM template. Concretely, some parts of the L2SM module can be directly mapped into the L2NM while other parts are generated as a function of the requested service and local guidelines. Finally, there are parts local to the service provider and do not map directly to the L2SM.

Note that using the L2NM within a service provider does not assume, nor does it precludes exposing the VPN service via the L2SM. This is deployment specific. Nevertheless, the design of L2NM tries to align as much as possible with the features supported by the L2SM to ease the grafting of both the L2NM and the L2SM for the sake of highly automated VPN service provisioning and delivery.

Network Topology Modules: An L2VPN involves nodes that are part of a

topology managed by the service provider network. Such a topology can be represented using the network topology module in [RFC8345] or its extension, such as a network YANG module for Service Attachment Points (SAPs) [I-D.ietf-opsawg-sap].

Device Modules: The L2NM is not a device model.

Once a global VPN service is captured by means of the L2NM, 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:

- * Interfaces [RFC8343].
- * BGP [I-D.ietf-idr-bgp-model].
- * MPLS [RFC8960].
- * Access Control Lists (ACLs) [RFC8519].

How the L2NM is used to derive device-specific actions is implementation specific.

6. Description of the Ethernet Segment YANG Module

The 'ietf-ethernet-segment' module (Figure 3) is used to manage a set of Ethernet segments in the context of an EVPN service.

```

module: ietf-ethernet-segment
  +--rw ethernet-segments
    +--rw ethernet-segment* [name]
      +--rw name string
      +--rw esi-type? identityref
      +--rw (esi-choice)?
        +--:(directly-assigned)
        | +--rw ethernet-segment-identifier? yang:hex-string
        +--:(auto-assigned)
        | +--rw esi-auto
        |   +--rw (auto-mode)?
        |     +--:(from-pool)
        |     | +--rw esi-pool-name? string
        |     +--:(full-auto)
        |     | +--rw auto? empty
        |     +--ro auto-ethernet-segment-identifier?
        |         yang:hex-string
        +--rw esi-redundancy-mode? identityref
      +--rw df-election
        +--rw df-election-method? identityref
        +--rw revertive? boolean
        +--rw election-wait-time? uint32
      +--rw split-horizon-filtering? boolean
      +--rw pbb
        +--rw backbone-src-mac? yang:mac-address
      +--rw member* [ne-id interface-id]
        +--rw ne-id string
        +--rw interface-id string

```

Figure 3: Ethernet Segments Tree Structure

The descriptions of the data nodes depicted in Figure 3 are as follows:

'name': Sets a name to uniquely identify an ES within a service provider network. This name is referenced in the VPN network access level of the L2NM (Section 7.6).

'esi-type': Indicates the Ethernet Segment Identifier (ESI) type as discussed in Section 5 of [RFC7432]. ESIs can be automatically assigned either with or without indicating a pool from which an ESI should be taken ('esi-pool-name'). The following types are supported:

'esi-type-0-operator': The ESI is directly configured by the VPN service provider. The configured value is provided in 'ethernet-segment-identifier'.

'esi-type-1-lacp': The ESI is auto-generated from the IEEE 802.1AX Link Aggregation Control Protocol (LACP) [IEEE802.1AX].

'esi-type-2-bridge': The ESI is auto-generated and determined based on the Layer 2 bridge protocol.

'esi-type-3-mac': The ESI is a MAC-based ESI value that can be auto-generated or configured by the VPN service provider.

'esi-type-4-router-id': The ESI is auto-generated or configured by the VPN service provider based on the Router ID. The 'router-id' supplied in Section 7.5 can be used to auto-derive an ESI when this type is used.

'esi-type-5-asn': The ESI is auto-generated or configured by the VPN service provider based on the Autonomous System (AS) number. The 'local-autonomous-system' supplied in Section 7.4 can be used to auto-derive an ESI when this type is used.

Auto-generated values can be retrieved using 'auto-ethernet-segment-identifier'.

'esi-redundancy-mode': Specifies the EVPN redundancy mode for a given ES. The following modes are supported: Single-Active (Section 14.1.1 of [RFC7432]) or All-Active (Section 14.1.2 of [RFC7432]).

'df-election': Specifies a set of parameters related to the Designated Forwarder (DF) election (Section 8.5 of [RFC7432]). For example, this data node can be used to indicate an election method (e.g., [RFC8584] or [I-D.ietf-bess-evpn-pref-df]). If no election method is indicated, the default one defined in Section 8.5 of [RFC7432] is used.

'split-horizon-filtering': Controls the activation of the split-horizon filtering for an ES (Section 8.3 of [RFC7432]).

'pbb': Indicates data nodes that are specific to PBB [IEEE-802-1ah]:

'backbone-src-mac': Associates a Provider Backbone MAC (B-MAC) address with an ES. This is particularly useful for All-Active multihomed ESes (Section 9.1 of [RFC7623]).

'member': Lists the members of an ES in a service provider network.

7. Description of the L2NM YANG Module

The L2NM ('ietf-l2vpn-ntw', Section 8.4) is used to manage L2VPNs within a service provider network. In particular, the 'ietf-l2vpn-ntw' module can be used to create, modify, delete and retrieve L2VPN services in a network controller. The module is designed to minimize the amount of customer-related information.

The full tree diagram of the module can be generated using the "pyang" tool [PYANG]. That tree is not included here because it is too long (Section 3.3 of [RFC8340]). Instead, subtrees are provided for the reader's convenience.

7.1. Overall Structure of the Module

The 'ietf-l2vpn-ntw' module uses two main containers: 'vpn-profiles' and 'vpn-services' (see Figure 4).

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

The 'vpn-services' container maintains the set of L2VPN services managed in the service provider network. The module allows creating a new L2VPN service by adding a new instance of 'vpn-service'. The 'vpn-service' is the data structure that abstracts the VPN service (Section 7.3).

```

module: ietf-l2vpn-ntw
  +--rw l2vpn-ntw
    +--rw vpn-profiles
    |   ...
    +--rw vpn-services
      +--rw vpn-service* [vpn-id]
      ...
      +--rw vpn-nodes
        +--rw vpn-node* [vpn-node-id]
        ...
        +--rw vpn-network-accesses
          +--rw vpn-network-access* [id]
          ...

```

Figure 4: Overall L2NM Tree Structure

7.2. VPN Profiles

The 'vpn-profiles' container (Figure 5) is used by a VPN service provider to define and maintain a set of VPN profiles [RFC9181] that apply to one or several VPN services.

```

+--rw l2vpn-ntw
  +--rw vpn-profiles
    +--rw valid-provider-identifiers
      +--rw external-connectivity-identifier* [id]
        {external-connectivity}?
        +--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 5: VPN Profiles Subtree Structure

This document does not make any assumption about the exact definition of these profiles. The exact definition of the profiles is local to each VPN service provider. The model only includes an identifier for these profiles in order to ease identifying and binding local policies when building a VPN service. As shown in Figure 5, the following identifiers can be included:

'external-connectivity-identifier': This identifier refers to a profile that defines the external connectivity provided to a VPN service (or a subset of VPN sites). External connectivity may be access to the Internet or restricted connectivity, such as access to a public/private cloud.

'encryption-profile-identifier': An encryption profile refers to a set of policies related to the encryption schemes and setup that can be applied when building and offering a VPN service.

'qos-profile-identifier': A Quality of Service (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 policies [RFC5880] 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, for example, of applying ACLs.

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

7.3. VPN Services

The 'vpn-service' is the data structure that abstracts an L2VPN service in the service provider network. Each 'vpn-service' is uniquely identified by an identifier: 'vpn-id'. Such a 'vpn-id' is only meaningful locally within the network controller. The subtree of the 'vpn-services' is shown in Figure 6.

```

+--rw vpn-services
  +--rw vpn-service* [vpn-id]
    +--rw vpn-id                vpn-common:vpn-id
    +--rw vpn-name?             string
    +--rw vpn-description?      string
    +--rw customer-name?        string
    +--rw parent-service-id?    vpn-common:vpn-id
    +--rw vpn-type?             identityref
    +--rw vpn-service-topology? identityref
    +--rw bgp-ad-enabled?        boolean
    +--rw signaling-type?        identityref
    +--rw global-parameters-profiles
    |   ...
  +--rw underlay-transport
    +--rw (type)?
    |   +--:(abstract)
    |   |   +--rw transport-instance-id? string
    |   |   +--rw instance-type?        identityref
    |   +--:(protocol)
    |   |   +--rw protocol*              identityref
  +--rw status
    +--rw admin-status
    |   +--rw status?                identityref
    |   +--rw last-change?           yang:date-and-time
    +--ro oper-status
    |   +--ro status?                identityref
    |   +--ro last-change?           yang:date-and-time
  +--rw vpn-nodes
    ...

```

Figure 6: VPN Services Subtree

The descriptions of the VPN service data nodes that are depicted in Figure 6 are as follows:

'vpn-id': An identifier that is used to uniquely identify the L2VPN service within the L2NM scope.

'vpn-name': Associates a name with the service in order to facilitate the identification of the service.

'vpn-description': Includes a textual description of the service.

The internal structure of a VPN description is local to each VPN service provider.

'customer-name': Indicates the name of the customer who ordered the service.

'parent-service-id': Refers to an identifier of the parent service (e.g., the L2SM, IETF network slice, VPN+) that triggered the creation of the L2VPN service. This identifier is used to easily correlate the (network) service as built in the network with a service order. A controller can use that correlation to enrich or populate some fields (e.g., description fields) as a function of local deployments.

'vpn-type': Indicates the L2VPN type. The following types, defined in [RFC9181], can be used for the L2NM:

'vpls': Virtual Private LAN Service (VPLS) as defined in [RFC4761] or [RFC4762]. This type is also used for hierarchical VPLS (H-VPLS) (Section 10 of [RFC4762]).

'vpws': Virtual Private Wire Service (VPWS) as defined in Section 3.1.1 of [RFC4664].

'vpws-evpn': VPWS as defined in [RFC8214].

'pbb-evpn': Provider Backbone Bridging (PBB) EVPNs as defined in [RFC7623].

'mpls-evpn': MPLS-based EVPNs [RFC7432].

'vxlan-evpn': VXLAN based EVPNs [RFC8365].

The type is used as a condition for the presence of some data nodes in the L2NM.

'vpn-service-topology': Indicates the network topology for the

service: hub-spoke, any-to-any, or custom. These types are defined in [RFC9181].

'bgp-ad-enabled': Controls whether BGP auto-discovery is enabled. If so, additional data nodes are included (Section 7.5.1).

'signaling-type': Indicates the signaling that is used for setting up pseudowires. Signaling type values are taken from [RFC9181]. The following signaling options are supported:

'bgp-signaling': The L2NM supports two flavors of BGP-signaled L2VPNs:

'l2vpn-bgp': The service is a Multipoint VPLS that uses a BGP control plane as described in [RFC4761] and [RFC6624].

'evpn-bgp': The service is a Multipoint VPLS that uses also a BGP control plane, but also includes the additional EVPN features and related parameters [RFC7432] and [RFC7209].

'ldp-signaling': A Multipoint VPLS that uses a mesh of LDP-signaled Pseudowires [RFC6074].

'l2tp-signaling': The L2NM uses L2TP-signaled Pseudowires as described in [RFC6074].

Table 1 summarizes the allowed signaling types for each VPN service type ('vpn-type'). See Section 7.5.2 for more details.

VPN Type	Signaling Options
vppls	l2tp-signaling, ldp-signaling, bgp-signaling (l2vpn-bgp)
vpws	l2tp-signaling, ldp-signaling, bgp-signaling (l2vpn-bgp)
vpws-evpn	bgp-signaling (evpn-bgp)
pbb-evpn	bgp-signaling (evpn-bgp)
mpls-evpn	bgp-signaling (evpn-bgp)
vxlan-evpn	bgp-signaling (evpn-bgp)

Table 1: Signaling Options per VPN Service Type

'global-parameters-profiles': Defines reusable parameters for the same L2VPN service.

More details are provided in Section 7.4.

'underlay-transport': Describes the preference for the transport technology to carry the traffic of the VPN service. This preference is especially useful in networks with multiple domains and Network-to-Network Interface (NNI) types. The underlay transport can be expressed as an abstract transport instance (e.g., an identifier of a VPN+ instance, a virtual network identifier, or a network slice name) or as an ordered list of the actual protocols to be enabled in the network.

A rich set of protocol identifiers that can be used to refer to an underlay transport (or how such an underlay is set up) are defined in [RFC9181].

The model defined in Section 6.3.2 of [I-D.ietf-teas-te-service-mapping-yang] may be used if specific protection and availability requirements are needed between PEs.

'status': Used to track the overall status of a given VPN service. Both operational and administrative status are maintained together with a timestamp. For example, a service can be created, but not put into effect.

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

'vpn-node': An abstraction that represents a set of policies applied to a network node and belonging to a single 'vpn-service'. An L2VPN 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 L2SM. Whether that information is included in the L2NM, e.g., to populate the various 'description' data nodes is implementation specific.

More details are provided in Section 7.5.

7.4. Global Parameters Profiles

The 'global-parameters-profile' defines reusable parameters for the same L2VPN service instance ('vpn-service'). Global parameters profiles are defined at the VPN service level, activated at the VPN node level, and then an activated VPN profile may be used at the VPN network access level. Each VPN instance profile is identified by 'profile-id'. Some of the data nodes can be adjusted at the VPN node or VPN network access levels. These adjusted values take precedence over the global values. The subtree of 'global-parameters-profile' is depicted in Figure 7.

```

...
+--rw vpn-services
  +--rw vpn-service* [vpn-id]
    ...
    +--rw global-parameters-profiles
      +--rw global-parameters-profile* [profile-id]
        +--rw profile-id string
        +--rw (rd-choice)?
          +--:(directly-assigned)
          | +--rw rd?

```

```

|         rt-types:route-distinguisher
+---:(directly-assigned-suffix)
|   +---rw rd-suffix?          uint16
+---:(auto-assigned)
|   +---rw rd-auto
|       +---rw (auto-mode)?
|           +---:(from-pool)
|               |   +---rw rd-pool-name?    string
|           +---:(full-auto)
|               |   +---rw auto?            empty
|           +---ro auto-assigned-rd?
|               rt-types:route-distinguisher
+---:(auto-assigned-suffix)
|   +---rw rd-auto-suffix
|       +---rw (auto-mode)?
|           +---:(from-pool)
|               |   +---rw rd-pool-name?    string
|           +---:(full-auto)
|               |   +---rw auto?            empty
|           +---ro auto-assigned-rd-suffix? uint16
+---:(no-rd)
|   +---rw no-rd?              empty
+---rw vpn-target* [id]
|   +---rw id                  uint8
|   +---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 local-autonomous-system? inet:as-number
+---rw svc-mtu?               uint32
+---rw ce-vlan-preservation?   boolean
+---rw ce-vlan-cos-preservation? boolean
+---rw control-word-negotiation? boolean
+---rw mac-policies
|   +---rw mac-addr-limit
|       |   +---rw limit-number?    uint16
|       |   +---rw time-interval?   uint32
|       |   +---rw action?          identityref
+---rw mac-loop-prevention
|   +---rw window?            uint32
|   +---rw frequency?         uint32
|   +---rw retry-timer?       uint32
|   +---rw protection-type?    identityref
+---rw multicast {vpn-common:multicast}?
|   +---rw enabled?            boolean

```

```

|      +---rw customer-tree-flavors
|      +---rw tree-flavor*   identityref
|      ...

```

Figure 7: Global Parameters Profiles Subtree

The description of the global parameters profile is as follows:

'profile-id': Uniquely identifies a global parameter profile in the context of an L2VPN service.

'rd': As defined in [RFC9181], these RD assignment modes are supported: direct assignment, automatic assignment from a given pool, full automatic assignment, and no assignment.

Also, the module accommodates deployments where only the Assigned Number subfield of RDs is assigned from a pool while the Administrator subfield is set to, e.g., the Router ID that is assigned to a VPN node. The module supports these modes for managing the Assigned Number subfield: explicit assignment, auto-assignment from a pool, and full auto-assignment.

'vpn-targets': Specifies RT import/export rules for the VPN service.

'local-autonomous-system': Indicates the Autonomous System Number (ASN) that is configured for the VPN node. The ASN can be used to auto-derive some other attributes such as RDs or Ethernet Segment Identifiers (ESIs).

'svc-mtu': Is the service MTU for an L2VPN service (i.e., Layer 2 MTU including L2 frame header/tail). It is also known as the maximum transmission unit or maximum frame size.

'ce-vlan-preservation': Is set to preserve the Customer Edge VLAN IDs (CE-VLAN IDs) from ingress to egress, i.e., CE-VLAN tag of the egress frame are identical to those of the ingress frame that yielded this egress service frame. If all-to-one bundling within a site is enabled, then preservation applies to all ingress service frames. If all-to-one bundling is disabled, then preservation applies to tagged Ingress service frames having CE-VLAN ID 1 through 4094.

'ce-vlan-cos-preservation': Controls the CE VLAN CoS preservation. When set, Priority Code Point (PCP) bits in the CE-VLAN tag of the egress frame are identical to those of the ingress frame that yielded this egress service frame.

'control-word-negotiation': Controls whether control-word

negotiation is enabled (if set to true) or not (if set to false). Refer to Section 7 of [RFC8077] for more details.

'mac-policies': Includes a set of MAC policies that apply to the service:

'mac-addr-limit': Is a container of MAC address limit configuration. It includes the following data nodes:

'limit-number': Maximum number of MAC addresses learned from the customer for a single service instance.

'time-interval': The aging time of the mac address.

'action': Specifies the action when the upper limit is exceeded: drop the packet, flood the packet, or simply send a warning message.

'mac-loop-prevention': Container for MAC loop prevention.

'window': The time interval over which a MAC mobility event is detected and checked.

'frequency': The number of times to detect MAC duplication, where a 'duplicate MAC address' situation has occurred within the 'window' time interval and the duplicate MAC address has been added to a list of duplicate MAC addresses.

'retry-timer': The retry timer. When the retry timer expires, the duplicate MAC address will be flushed from the MAC-VRF.

'protection-type': It defines the loop prevention type (e.g., shut).

'multicast': Controls whether multicast is allowed in the service.

7.5. VPN Nodes

The 'vpn-node' (Figure 8) is an abstraction that represents a set of policies/configurations applied to a network node and that belong to a single 'vpn-service'. A 'vpn-node' contains 'vpn-network-accesses', which are the interfaces involved in the creation of the VPN. The customer sites are connected to the 'vpn-network-accesses'.

```

+--rw l2vpn-ntw
+--rw vpn-profiles
|   ...
+--rw vpn-services
+--rw vpn-service* [vpn-id]
    ...
+--rw vpn-nodes
+--rw vpn-node* [vpn-node-id]
+--rw vpn-node-id          vpn-common:vpn-id
+--rw description?         string
+--rw ne-id?               string
+--rw role?                identityref
+--rw router-id?           rt-types:router-id
+--rw active-global-parameters-profiles
|   +--rw global-parameters-profile* [profile-id]
|   |   +--rw profile-id          leafref
|   |   +--rw local-autonomous-system?
|   |   |   inet:as-number
|   |   +--rw svc-mtu?            uint32
|   |   +--rw ce-vlan-preservation? boolean
|   |   +--rw ce-vlan-cos-preservation? boolean
|   |   +--rw control-word-negotiation? boolean
|   |   +--rw mac-policies
|   |   |   +--rw mac-addr-limit
|   |   |   |   +--rw limit-number?    uint16
|   |   |   |   +--rw time-interval?   uint32
|   |   |   |   +--rw action?          identityref
|   |   |   +--rw mac-loop-prevention
|   |   |   |   +--rw window?          uint32
|   |   |   |   +--rw frequency?       uint32
|   |   |   |   +--rw retry-timer?     uint32
|   |   |   |   +--rw protection-type? identityref
|   |   +--rw multicast {vpn-common:multicast}?
|   |   |   +--rw enabled?            boolean
|   |   +--rw customer-tree-flavors
|   |   |   +--rw tree-flavor*        identityref
+--rw status
|   ...
+--rw bgp-auto-discovery
|   ...
+--rw signaling-option
|   ...
+--rw vpn-network-accesses
    ...

```

Figure 8: VPN Nodes Subtree

The descriptions of VPN node data nodes are as follows:

- 'vpn-node-id': Used to uniquely identify a node that enables a VPN network access.
- 'description': Provides a textual description of the VPN node.
- 'ne-id': Includes an identifier of the network element where the VPN node is deployed.
- 'role': Indicates the role of the VPN instance profile in the VPN. Role values are defined in [RFC9181] (e.g., 'any-to-any-role', 'spoke-role', 'hub-role').
- 'router-id': Indicates a 32-bit number that is used to uniquely identify a router within an Autonomous System (AS).
- 'active-global-parameters-profiles': Lists the set of active global VPN parameters profiles for this VPN node. Concretely, one or more global profiles that are defined at the VPN service level (i.e., under 'l2vpn-ntw/vpn-services/vpn-service' level) can be activated at the VPN node level; each of these profiles is uniquely identified by means of 'profile-id'. The structure of 'active-global-parameters-profiles' uses the same data nodes as Section 7.4 except RD and RT related data nodes.
- Values defined in 'active-global-parameters-profiles' overrides the values defined in the VPN service level.
- 'status': Tracks the status of a node involved in a VPN service. Both operational and administrative status are maintained. A mismatch between the administrative status vs. the operational status can be used as a trigger to detect anomalies.
- 'bgp-auto-discovery': See Section 7.5.1.
- 'signaling-option': See Section 7.5.2.
- 'vpn-network-accesses': Represents the point to which sites are connected.

Note that, unlike the L2SM, the L2NM does not need to model the customer site -- only the points that receive traffic from the site are covered (i.e., the PE side of Provider Edge to Customer Edge (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 that can be applied during the service creation.

See Section 7.6 for more details.

7.5.1. BGP Auto-Discovery

The 'bgp-auto-discovery' container (Figure 9) includes the required information for the activation of BGP auto-discovery [RFC4761][RFC6624].

```

+--rw l2vpn-ntw
  +--rw vpn-profiles
  |   ...
  +--rw vpn-services
    +--rw vpn-service* [vpn-id]
    |   ...
    +--rw vpn-nodes
      +--rw vpn-node* [vpn-node-id]
      |   ...
      +--rw bgp-auto-discovery
        +--rw (bgp-type)?
        |   +--:(l2vpn-bgp)
        |   |   +--rw vpn-id?
        |   |   |   vpn-common:vpn-id
        |   +--:(evpn-bgp)
        |   |   +--rw evpn-type?                leafref
        |   |   +--rw auto-rt-enable?            boolean
        |   |   +--ro auto-route-target?
        |   |   |   rt-types:route-target
        |   +--rw (rd-choice)?
        |   |   +--:(directly-assigned)
        |   |   |   +--rw rd?
        |   |   |   |   rt-types:route-distinguisher
        |   |   +--:(directly-assigned-suffix)
        |   |   |   +--rw rd-suffix?                uint16
        |   |   +--:(auto-assigned)
        |   |   |   +--rw rd-auto
        |   |   |   |   +--rw (auto-mode)?
        |   |   |   |   |   +--:(from-pool)
        |   |   |   |   |   |   +--rw rd-pool-name?    string
        |   |   |   |   |   +--:(full-auto)
        |   |   |   |   |   +--rw auto?                empty
        |   |   |   |   +--ro auto-assigned-rd?
        |   |   |   |   |   rt-types:route-distinguisher
        |   |   +--:(auto-assigned-suffix)
        |   |   |   +--rw rd-auto-suffix
        |   |   |   |   +--rw (auto-mode)?
        |   |   |   |   |   +--:(from-pool)
        |   |   |   |   |   |   +--rw rd-pool-name?    string
        |   |   |   |   |   +--:(full-auto)

```



```

...
+--rw vpn-nodes
  +--rw vpn-node* [vpn-node-id]
    ...
  +--rw signaling-option
    +--rw advertise-mtu?          boolean
    +--rw mtu-allow-mismatch?    boolean
    +--rw signaling-type?        leafref
    +--rw (signaling-option)?
      +--:(bgp)
      |   ...
      +--:(ldp-or-l2tp)
        +--rw (ldp-or-l2tp)?
          +--:(ldp)
          |   ...
          +--:(l2tp)
          |   ...

```

Figure 10: Signaling Option Overall Subtree

The following signaling data nodes are supported:

- 'advertise-mtu': Controls whether MTU is advertised when setting a pseudowire (e.g., Section 4.3 of [RFC4667], Section 5.1 of [RFC6624], or Section 6.1 of [RFC4762]).
- 'mtu-allow-mismatch': When set to true, it allows MTU mismatch for a pseudowire (see, e.g., Section 4.3 of [RFC4667]).
- 'signaling-type': Indicates the signaling type. This type inherits the value of 'signaling-type' defined at the service level (Section 7.3).
- 'bgp': Is provided when BGP is used for L2VPN signaling. Refer to Section 7.5.2.1 for more details.
- 'ldp': The model supports the configuration of the parameters that are discussed in Section 6 of [RFC4762]. Refer to Section 7.5.2.2 for more details.
- 'l2tp': The model supports the configuration of the parameters that are discussed in Section 4 of [RFC4667]. Refer to Section 7.5.2.3 for more details.

7.5.2.1. BGP

The structure of the BGP-related data nodes is provided in Figure 11.

```

...
+--rw (signaling-option)?
  ...
  +--:(bgp)
    +--rw (bgp-type)?
      +--:(l2vpn-bgp)
        +--rw ce-range?          uint16
        +--rw pw-encapsulation-type?
          |
          | identityref
        +--rw vpls-instance
          |
          | +--rw vpls-edge-id?          uint16
          | +--rw vpls-edge-id-range?    uint16
      +--:(evpn-bgp)
        +--rw evpn-type?          leafref
        +--rw service-interface-type?
          |
          | identityref
        +--rw evpn-policies
          +--rw mac-learning-mode?
            |
            | identityref
          +--rw ingress-replication?
            |
            | boolean
          +--rw p2mp-replication?
            |
            | boolean
          +--rw arp-proxy {vpn-common:ipv4}?
            +--rw enable?          boolean
            +--rw arp-suppression?
              |
              | boolean
            +--rw ip-mobility-threshold?
              |
              | uint16
            +--rw duplicate-ip-detection-interval?
              |
              | uint16
          +--rw nd-proxy {vpn-common:ipv6}?
            +--rw enable?          boolean
            +--rw nd-suppression?
              |
              | boolean
            +--rw ip-mobility-threshold?
              |
              | uint16
            +--rw duplicate-ip-detection-interval?
              |
              | uint16
          +--rw underlay-multicast?
            |
            | boolean
          +--rw flood-unknown-unicast-supression?
            |
            | boolean
          +--rw vpws-vlan-aware?    boolean

```

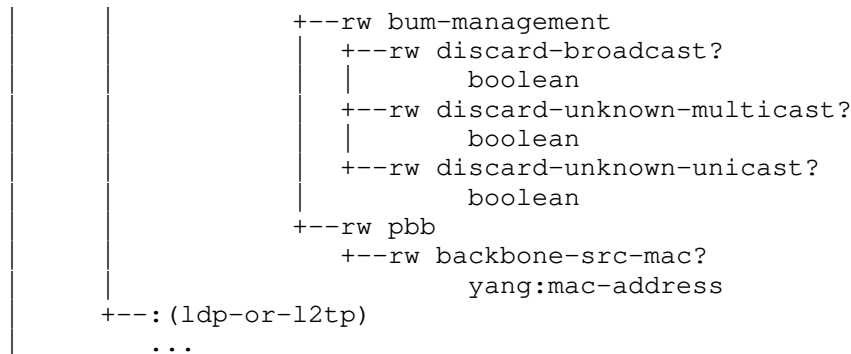


Figure 11: Signaling Option Subtree (BGP)

Remote CEs that are entitled to connect to the same VPN should fit with the CE range ('ce-range') as discussed in Section 2.2.3 of [RFC6624]. 'pw-encapsulation-type' is used to control the pseudowire encapsulation type (Section 3 of [RFC6624]). The value of the 'pw-encapsulation-type' are taken from the IANA-maintained "iana-bgp-l2-encaps" module (Section 8.1).

For the specific case of VPLS, the VPLS Edge ID (VE ID, 'vpls-edge-id') and a VE ID range ('vpls-edge-id-range') are provided as per Section 3.2 of [RFC4761]. If different VE IDs are required (e.g., multihoming as per Section 3.5 of [RFC4761]), these IDs are configured at the VPN network access level (under 'signaling-option' in Section 7.6).

For EVPN-related L2VPNs, 'service-interface-type' indicates whether this is a VLAN-based, VLAN bundle, or VLAN-aware bundle service interface (Section 6 of [RFC7432]). Moreover, a set of policies can be provided such as MAC address learning mode (Section 9 of [RFC7432]), ingress replication (Section 12.1 of [RFC7432]), Address Resolution Protocol (ARP) and Neighbor Discovery (ND) proxy (Section 10 of [RFC7432]), processing of Broadcast, unknown unicast, or multicast (BUM) (Section 12 of [RFC7432]), etc.

7.5.2.2. LDP

The model supports the configuration of the parameters that are discussed in Section 6 of [RFC4762]. Such parameters include an Attachment Group Identifier (AGI) (a.k.a., VPLS-id), a Source Attachment Individual Identifier (SAII), a list of peers that are associated with a Target Attachment Individual Identifier (TAII), a pseudowire type, and a pseudowire description (Figure 12). Unlike BGP, only Ethernet and Ethernet tagged mode are supported. The AGI, SAII, and TAI are encoded following the types defined in Section 3.4

of [RFC4446].

```

...
+--rw (signaling-option)?
...
+--:(bgp)
|
...
+--:(ldp-or-l2tp)
+--rw ldp-or-l2tp
+--rw agi?
|
rt-types:route-distinguisher
+--rw saii?                               uint32
+--rw remote-targets* [taii]
|   +--rw tائي                               uint32
|   +--rw peer-addr          inet:ip-address
+--rw (ldp-or-l2tp)?
+--:(ldp)
|
+--rw t-ldp-pw-type?
|   identityref
+--rw pw-type?          identityref
+--rw pw-description?    string
+--rw mac-addr-withdraw? boolean
+--rw pw-peer-list*
|   [peer-addr vc-id]
|   +--rw peer-addr
|   |   inet:ip-address
|   +--rw vc-id    string
|   +--rw pw-priority?  uint32
+--rw qinq
|   +--rw s-tag    dot1q-types:vlanid
|   +--rw c-tag    dot1q-types:vlanid
+--:(l2tp)
...
...

```

Figure 12: Signaling Option Subtree (LDP)

7.5.2.3. L2TP

The model supports the configuration of the parameters that are discussed in Section 4 of [RFC4667]. Such parameters include a Router ID that is used to uniquely identify a PE, a pseudowire type, an AGI, an SAI, and a list of peers that are associated with a TAI (Figure 13). The pseudowire type ('pseudowire-type') value is taken from the IANA-maintained "iana-pseudowire-types" module (Section 8.2).

```

...
+--rw (signaling-option)?
  ...
  +--:(bgp)
  |   ...
  +--:(ldp-or-l2tp)
  |   +--rw ldp-or-l2tp
  |   |   +--rw agi?
  |   |   |   rt-types:route-distinguisher
  |   |   +--rw saii?                               uint32
  |   |   +--rw remote-targets* [taii]
  |   |   |   +--rw taii                               uint32
  |   |   |   +--rw peer-addr       inet:ip-address
  |   +--rw (ldp-or-l2tp)?
  |   |   +--:(ldp)
  |   |   |   ...
  |   |   +--:(l2tp)
  |   |   |   +--rw router-id?
  |   |   |   |   rt-types:router-id
  |   |   |   +--rw pseudowire-type?
  |   |   |   |   identityref
...

```

Figure 13: Signaling Option Subtree (L2TP)

7.6. VPN Network Accesses

A 'vpn-network-access' (Figure 14) represents an entry point to a VPN service. In other words, this container encloses the parameters that describe the access information for the traffic that belongs to a particular L2VPN.

A 'vpn-network-access' includes information such as the connection on which the access is defined, the specific Layer 2 service requirements, etc.

```

...
+--rw vpn-nodes
  +--rw vpn-node* [vpn-node-id]
    ...
    +--rw vpn-network-accesses
      +--rw vpn-network-access* [id]
        +--rw id                               vpn-common:vpn-id
        +--rw description?                     string
        +--rw interface-id?                   string
        +--rw active-vpn-node-profile?        leafref
        +--rw status
        |   ...
        +--rw connection
        |   ...
        +--rw (signaling-option)?
          +--:(bgp)
            +--rw (bgp-type)?
              +--:(l2vpn-bgp)
                +--rw ce-id?                   uint16
                +--rw remote-ce-id?           uint16
                +--rw vpls-instance
                |   +--rw vpls-edge-id?       uint16
              +--:(evpn-bgp)
                +--rw df-preference?           uint16
                +--rw vpws-service-instance
                |   ...
            +--rw group* [group-id]
              +--rw group-id                   string
              +--rw precedence?
              |   identityref
              +--rw ethernet-segment-identifier? string
            +--rw ethernet-service-oam
            |   ...
            +--rw service
            |   ...

```

Figure 14: VPN Network Access Subtree

The VPN network access comprises:

'id': Includes an identifier of the VPN network access.

'description': Includes a textual description of the VPN network access.

'interface-id': Indicates the interface on which the VPN network access is bound.

'active-vpn-node-profile': Provides a pointer to an active 'global-parameters-profile' at the VPN node level. Referencing an active 'global-parameters-profile' implies that all associated data nodes will be inherited by the VPN network access. However, some of the inherited data nodes (e.g., ACL policies) can be overridden at the VPN network access level. In such case, adjusted values take precedence over inherited values.

'status': Indicates the administrative and operational status of the VPN network access.

'connection': Represents and groups the set of Layer 2 connectivity from where the traffic of the L2VPN in a particular VPN Network access is coming. See Section 7.6.1.

'signaling-option': Indicates a set of signaling options that are specific to a given VPN network access, e.g., a CE ID ('ce-id' identifying the CE within the VPN) and a remote CE ID as discussed in Section 2.2.2 of [RFC6624].

It can also include a set of data nodes that are required for the configuration of a VPWS-EVPN [RFC8214]. See Section 7.6.2.

'group': Is used for grouping VPN network accesses by assigning the same identifier to these accesses. The precedence attribute is used to differentiate the primary and secondary accesses for a service with multiple accesses. An example to illustrate the use of this container for redundancy purposes is provided in Appendix A.6. This container is also used to identify the link of an ES by allocating the same ESI. An example to illustrate this functionality is provided in Appendices A.4 and A.5.

'ethernet-service-oam': Carries information about the service OAM. See Section 7.6.3.

'service': Specifies the service parameters (e.g., QoS, multicast) to apply for a given VPN network access. See Section 7.6.4.

7.6.1. Connection

The 'connection' container (Figure 15) is used to configure the relevant properties of the interface to which the L2VPN instance is attached to (e.g., encapsulation type, Link Aggregation Group (LAG) interfaces, split-horizon). The L2NM supports tag manipulation operations (e.g., tag rewrite).

Note that the 'connection' container does not include the physical-specific configuration as this is assumed to be directly handled using device modules (e.g., interfaces module). Moreover, this design is also meant to avoid manipulated global parameters at the service level and lower the risk of impacting other services sharing the same physical interface.

A reference to the bearer is maintained to allow keeping the link between the L2SM and the L2NM when both data models are used in a given deployment.

Some consistency checks should be ensured by implementations (typically, network controllers) for LAG interface as the same information (e.g., LACP system-id) should be provided to the involved nodes.

The L2NM inherits the 'member-link-list' structure from the L2SM (including indication of OAM 802.3ah support [IEEE-802-3ah]).

```

...
+---rw vpn-nodes
  +---rw vpn-node* [vpn-node-id]
    ...
    +---rw vpn-network-accesses
      +---rw vpn-network-access* [id]
        ...
        +---rw connection
          +---rw l2-termination-point?
          |   string
          +---rw local-bridge-reference?
          |   string
          +---rw bearer-reference?          string
          |   {vpn-common:bearer-reference}?
          +---rw encapsulation
            +---rw encap-type?              identityref
            +---rw dot1q
              +---rw tag-type?              identityref
              +---rw cvlan-id?
              |   dot1q-types:vlanid
              +---rw tag-operations
                +---rw (op-choice)?
                |   +---:(pop)
                |   |   +---rw pop?          empty
                |   +---:(push)
                |   |   +---rw push?         empty
                |   +---:(translate)
                |   |   +---rw translate?    empty
                +---rw tag-1?

```

```

|         dot1q-types:vlanid
+---rw tag-1-type?
|         dot1q-types:dot1q-tag-type
+---rw tag-2?
|         dot1q-types:vlanid
+---rw tag-2-type?
|         dot1q-types:dot1q-tag-type
+---rw priority-tagged
| +---rw tag-type? identityref
+---rw qinq
+---rw tag-type? identityref
+---rw svlan-id
|         dot1q-types:vlanid
+---rw cvlan-id
|         dot1q-types:vlanid
+---rw tag-operations
+---rw (op-choice)?
| +---:(pop)
| | +---rw pop? uint8
| +---:(push)
| | +---rw push? empty
| +---:(translate)
| | +---rw translate? empty
+---rw tag-1?
|         dot1q-types:vlanid
+---rw tag-1-type?
|         dot1q-types:dot1q-tag-type
+---rw tag-2?
|         dot1q-types:vlanid
+---rw tag-2-type?
|         dot1q-types:dot1q-tag-type
+---rw lag-interface
| {vpn-common:lag-interface}?
+---rw lag-interface-id? string
+---rw lacp
+---rw lacp-state? boolean
+---rw mode? identityref
+---rw speed? uint32
+---rw mini-link-num? uint32
+---rw system-id?
|         yang:mac-address
+---rw admin-key? uint16
+---rw system-priority? uint16
+---rw member-link-list
| +---rw member-link* [name]
| | +---rw name string
| | +---rw speed? uint32
| | +---rw mode? identityref

```

```

|
|
|      +--rw link-mtu?      uint32
|      +--rw oam-802.3ah-link
|          |
|          +--rw enable?    boolean
|      +--rw flow-control?  boolean
|      +--rw lldp?          boolean
+--rw split-horizon
+--rw group-name?    string
...

```

Figure 15: Connection Subtree

7.6.2. EVPN-VPWS Service Instance

The 'vpws-service-instance' provides the local and remote VPWS Service Instance (VSI) [RFC8214]. This container is only present when the 'vpn-type' is VPWS-EVPN. As shown in Figure 16, the VSIs can be configured by a VPN service provider or auto-generated.

An example to illustrate the use of the L2NM to configure VPWS-EVPN instances is provided in Appendix A.4.

```

...
+--rw vpn-nodes
  +--rw vpn-node* [vpn-node-id]
    ...
    +--rw vpn-network-accesses
      +--rw vpn-network-access* [id]
        ...
        +--rw (signaling-option)?
          +--:(bgp)
            +--rw (bgp-type)?
              +--:(l2vpn-bgp)
                |
                ...
                +--:(evpn-bgp)
                  +--rw vpws-service-instance
                    +--rw (local-vsi-choice)?
                      +--:(directly-assigned)
                        |
                        +--rw local-vpws-service-instance?
                          uint32
                      +--:(auto-assigned)
                        +--rw local-vsi-auto
                          +--rw (auto-mode)?
                            +--:(from-pool)
                              |
                              +--rw vsi-pool-name?
                                string
                            +--:(full-auto)
                              |
                              +--rw auto?          empty
                              +--ro auto-local-vsi? uint32
                        +--rw (remote-vsi-choice)?
                          +--:(directly-assigned)
                            |
                            +--rw remote-vpws-service-instance?
                              uint32
                          +--:(auto-assigned)
                            +--rw remote-vsi-auto
                              +--rw (auto-mode)?
                                +--:(from-pool)
                                  |
                                  +--rw vsi-pool-name?
                                    string
                                +--:(full-auto)
                                  |
                                  +--rw auto?          empty
                                  +--ro auto-remote-vsi? uint32
          ...

```

Figure 16: EVPN-VPWS Service Instance Subtree

7.6.3. Ethernet OAM

Ethernet OAM refers to both [IEEE-802-lag] and [ITU-T-Y-1731].

As shown in Figure 17, the L2NM inherits the same structure as in Section 5.3.2.2.6 of [RFC8466] for OAM matters.

```

+--rw l2vpn-ntw
+--rw vpn-profiles
|   ...
+--rw vpn-services
+--rw vpn-service* [vpn-id]
    ...
+--rw vpn-nodes
+--rw vpn-node* [vpn-node-id]
    ...
+--rw vpn-network-accesses
+--rw vpn-network-access* [id]
    ...
+--rw ethernet-service-oam
+--rw md-name?      string
+--rw md-level?     uint8
+--rw cfm-802.1-ag
+--rw n2-uni-c* [maid]
+--rw maid          string
+--rw mep-id?       uint32
+--rw mep-level?    uint32
+--rw mep-up-down?  enumeration
+--rw remote-mep-id? uint32
+--rw cos-for-cfm-pdus? uint32
+--rw ccm-interval? uint32
+--rw ccm-holdtime? uint32
+--rw ccm-p-bits-pri? ccm-priority-type
+--rw n2-uni-n* [maid]
+--rw maid          string
+--rw mep-id?       uint32
+--rw mep-level?    uint32
+--rw mep-up-down?  enumeration
+--rw remote-mep-id? uint32
+--rw cos-for-cfm-pdus? uint32
+--rw ccm-interval? uint32
+--rw ccm-holdtime? uint32
+--rw ccm-p-bits-pri? ccm-priority-type
+--rw y-1731* [maid]
+--rw maid          string
+--rw mep-id?       uint32
+--rw pm-type?      identityref
+--rw remote-mep-id? uint32

```

```

|      +---rw message-period?      uint32
|      +---rw measurement-interval? uint32
|      +---rw cos?                  uint32
|      +---rw loss-measurement?     boolean
|      +---rw synthethic-loss-measurement?
|      |      boolean
|      +---rw delay-measurement
|      |      +---rw enable-dm?    boolean
|      |      +---rw two-way?      boolean
|      +---rw frame-size?          uint32
|      +---rw session-type?        enumeration
...

```

Figure 17: OAM Subtree

7.6.4. Services

The 'service' container (Figure 18) provides a set of service-specific configuration such as Quality of Service (QoS).

```

+---rw l2vpn-ntw
|   +---rw vpn-profiles
|   |   ...
|   +---rw vpn-services
|   |   +---rw vpn-service* [vpn-id]
|   |   ...
|   |   +---rw vpn-nodes
|   |   |   +---rw vpn-node* [vpn-node-id]
|   |   |   ...
|   |   +---rw vpn-network-accesses
|   |   |   +---rw vpn-network-access* [id]
|   |   |   ...
|   |   +---rw service
|   |   |   +---rw mtu?          uint32
|   |   |   +---rw svc-pe-to-ce-bandwidth
|   |   |   |   {vpn-common:inbound-bw}?
|   |   |   |   ...
|   |   |   +---rw svc-ce-to-pe-bandwidth
|   |   |   |   {vpn-common:outbound-bw}?
|   |   |   |   ...
|   |   |   +---rw qos {vpn-common:qos}?
|   |   |   |   ...
|   |   |   +---rw mac-policies
|   |   |   |   ...
|   |   |   +---rw broadcast-unknown-unicast-multicast
|   |   |   ...

```

Figure 18: Service Overall Subtree

The description of the service data nodes is as follows:

'mtu': Specifies the Layer 2 MTU for the VPN network access.

'svc-pe-to-ce-bandwidth' and 'svc-ce-to-pe-bandwidth': Specify the service bandwidth for the L2VPN service.

'svc-pe-to-ce-bandwidth' indicates the inbound bandwidth of the connection (i.e., download bandwidth from the service provider to the site).

'svc-ce-to-pe-bandwidth' indicates the outbound bandwidth of the connection (i.e., upload bandwidth from the site to the service provider).

'svc-pe-to-ce-bandwidth' and 'svc-ce-to-pe-bandwidth' can be represented using the Committed Information Rate (CIR), the Excess Information Rate (EIR), or the Peak Information Rate (PIR).

As shown in Figure 19, the structure of service bandwidth data nodes is inherited from the L2SM [RFC8466]. The following types, defined in [RFC9181], can be used to indicate the bandwidth type:

'bw-per-cos': The bandwidth is per Class of Service (CoS).

'bw-per-port': The bandwidth is per VPN network access.

'bw-per-site': The bandwidth is to all VPN network accesses that belong to the same site.

'bw-per-service': The bandwidth is per L2VPN service.

```

+---rw service
...
+---rw svc-pe-to-ce-bandwidth
    {vpn-common:inbound-bw}?
    +---rw pe-to-ce-bandwidth* [bw-type]
    +---rw bw-type          identityref
    +---rw (type)?
    +---:(per-cos)
    |   +---rw cos* [cos-id]
    |   |   +---rw cos-id      uint8
    |   |   +---rw cir?        uint64
    |   |   +---rw cbs?        uint64
    |   |   +---rw eir?        uint64
    |   |   +---rw ebs?        uint64
    |   |   +---rw pir?        uint64
    |   |   +---rw pbs?        uint64
    |   +---:(other)
    |   |   +---rw cir?        uint64
    |   |   +---rw cbs?        uint64
    |   |   +---rw eir?        uint64
    |   |   +---rw ebs?        uint64
    |   |   +---rw pir?        uint64
    |   |   +---rw pbs?        uint64
+---rw svc-ce-to-pe-bandwidth
    {vpn-common:outbound-bw}?
    +---rw ce-to-pe-bandwidth* [bw-type]
    +---rw bw-type          identityref
    +---rw (type)?
    +---:(per-cos)
    |   +---rw cos* [cos-id]
    |   |   +---rw cos-id      uint8
    |   |   +---rw cir?        uint64
    |   |   +---rw cbs?        uint64
    |   |   +---rw eir?        uint64
    |   |   +---rw ebs?        uint64
    |   |   +---rw pir?        uint64
    |   |   +---rw pbs?        uint64
    |   +---:(other)
    |   |   +---rw cir?        uint64
    |   |   +---rw cbs?        uint64
    |   |   +---rw eir?        uint64
    |   |   +---rw ebs?        uint64
    |   |   +---rw pir?        uint64
    |   |   +---rw pbs?        uint64
...

```

Figure 19: Service Bandwidth Subtree

'qos': Is used to define a set of QoS policies to apply on a given VPN network access (Figure 20). The QoS classification can be based on many criteria such as source MAC address, destination MAC address, etc. See also Section 5.10.2.1 of [RFC8466] for more discussion of QoS classification including the use of color types.

```

+--rw service
...
+--rw qos {vpn-common:qos}?
|
|   +--rw qos-classification-policy
|   |
|   |   +--rw rule* [id]
|   |   |
|   |   |   +--rw id string
|   |   |   +--rw (match-type)?
|   |   |   |
|   |   |   |   +--:(match-flow)
|   |   |   |   |
|   |   |   |   |   +--rw match-flow
|   |   |   |   |   |
|   |   |   |   |   |   +--rw dscp? inet:dscp
|   |   |   |   |   |   +--rw dot1q? uint16
|   |   |   |   |   |   +--rw pcp? uint8
|   |   |   |   |   |   +--rw src-mac-address?
|   |   |   |   |   |   |   yang:mac-address
|   |   |   |   |   |   +--rw dst-mac-address?
|   |   |   |   |   |   |   yang:mac-address
|   |   |   |   |   |   +--rw color-type?
|   |   |   |   |   |   |   identityref
|   |   |   |   |   |   +--rw any? empty
|   |   |   |   |   +--:(match-application)
|   |   |   |   |   |   +--rw match-application?
|   |   |   |   |   |   |   identityref
|   |   |   |   +--rw target-class-id? string
|   |   +--rw qos-profile
|   |   |   +--rw qos-profile* [profile]
|   |   |   |   +--rw profile leafref
|   |   |   |   +--rw direction? identityref
|   |
|   ...

```

Figure 20: QoS Subtree

'mac-policies': Lists a set of MAC-related policies such as MAC ACLs. Similar to [RFC8519], an ACL match can be based upon source MAC address, source MAC address mask, destination MAC address, destination MAC address mask, or a combination thereof.

A data frame that matches an ACL can be dropped, flooded, or trigger an alarm. A rate-limit policy can be defined for handling frames that match an ACL entry with 'flood' action.

When 'mac-loop-prevention' or 'mac-addr-limit' data nodes are provided, they take precedence over the ones included in the 'global-parameters-profile' at the VPN service or VPN node levels.

```

+--rw service
...
+--rw mac-policies
|
| +--rw access-control-list* [name]
| |
| | +--rw name string
| | +--rw src-mac-address*
| | | yang:mac-address
| | +--rw src-mac-address-mask*
| | | yang:mac-address
| | +--rw dst-mac-address*
| | | yang:mac-address
| | +--rw dst-mac-address-mask*
| | | yang:mac-address
| | +--rw action? identityref
| | +--rw rate-limit? decimal64
| +--rw mac-loop-prevention
| |
| | +--rw window? uint32
| | +--rw frequency? uint32
| | +--rw retry-timer? uint32
| | +--rw protection-type? identityref
| +--rw mac-addr-limit
| |
| | +--rw limit-number? uint16
| | +--rw time-interval? uint32
| | +--rw action? identityref
...

```

Figure 21: MAC Policies Subtree

'broadcast-unknown-unicast-multicast': Defines the type of site in the customer multicast service topology: source, receiver, or both. It is also used to define multicast group-to-port mappings.

```

+--rw service
...
+--rw broadcast-unknown-unicast-multicast
|
| +--rw multicast-site-type?
| | enumeration
| +--rw multicast-gp-address-mapping* [id]
| |
| | +--rw id uint16
| | +--rw vlan-id uint32
| | +--rw mac-gp-address
| | | yang:mac-address
| | +--rw port-lag-number? uint32
+--rw bum-overall-rate? uint64

```

Figure 22: BUM Subtree

8. YANG Modules

8.1. IANA-Maintained Module for BGP Layer 2 Encapsulation Types

The "iana-bgp-l2-encaps" YANG module echoes the registry available at [IANA-BGP-L2].

This module references [RFC3032], [RFC4446], [RFC4448], [RFC4553], [RFC4618], [RFC4619], [RFC4717], [RFC4761], [RFC4816], [RFC4842], and [RFC5086].

<CODE BEGINS>

```
file "iana-bgp-l2-encaps@2021-07-05.yang"
```

```
module iana-bgp-l2-encaps {  
  yang-version 1.1;  
  namespace "urn:ietf:params:xml:ns:yang:iana-bgp-l2-encaps";  
  prefix iana-bgp-l2-encaps;
```

```
  organization
```

```
    "IANA";
```

```
  contact
```

```
    "Internet Assigned Numbers Authority
```

```
    Postal: ICANN
```

```
      12025 Waterfront Drive, Suite 300
```

```
      Los Angeles, CA 90094-2536
```

```
      United States of America
```

```
    Tel: +1 310 301 5800
```

```
    <mailto:iana@iana.org>;
```

```
  description
```

```
    "This module contains a collection of IANA-maintained YANG  
    data types that are used for referring to BGP Layer 2  
    encapsulation types.
```

```
    Copyright (c) 2022 IETF Trust and the persons identified as  
    authors of the code. All rights reserved.
```

```
    Redistribution and use in source and binary forms, with or  
    without modification, is permitted pursuant to, and subject  
    to the license terms contained in, the Revised BSD License  
    set forth in Section 4.c of the IETF Trust's Legal Provisions  
    Relating to IETF Documents  
    (https://trustee.ietf.org/license-info).
```

```
    This version of this YANG module is part of RFC XXXX; see
```

```
    the RFC itself for full legal notices.";

revision 2021-07-05 {
  description
    "First revision.";
  reference
    "RFC XXXX: A YANG Network Data Model for Layer 2 VPNs.";
}

identity bgp-l2-encaps-type {
  description
    "Base BGP Layer 2 encapsulation type.";
  reference
    "RFC 6624: Layer 2 Virtual Private Networks Using BGP for
      Auto-Discovery and Signaling";
}

identity frame-relay {
  base bgp-l2-encaps-type;
  description
    "Frame Relay.";
  reference
    "RFC 4446: IANA Allocations for Pseudowire Edge
      to Edge Emulation (PWE3)";
}

identity atm-aal5 {
  base bgp-l2-encaps-type;
  description
    "ATM AAL5 SDU VCC transport.";
  reference
    "RFC 4446: IANA Allocations for Pseudowire Edge
      to Edge Emulation (PWE3)";
}

identity atm-cell {
  base bgp-l2-encaps-type;
  description
    "ATM transparent cell transport.";
  reference
    "RFC 4816: Pseudowire Emulation Edge-to-Edge (PWE3)
      Asynchronous Transfer Mode (ATM) Transparent
      Cell Transport Service";
}

identity ethernet-tagged-mode {
  base bgp-l2-encaps-type;
  description
```

```
        "Ethernet (VLAN) Tagged Mode.";
    reference
        "RFC 4448: Encapsulation Methods for Transport of Ethernet
        over MPLS Networks";
}

identity ethernet-raw-mode {
    base bgp-l2-encaps-type;
    description
        "Ethernet Raw Mode.";
    reference
        "RFC 4448: Encapsulation Methods for Transport of Ethernet
        over MPLS Networks";
}

identity hdlc {
    base bgp-l2-encaps-type;
    description
        "Cisco HDLC.";
    reference
        "RFC 4618: Encapsulation Methods for Transport of
        PPP/High-Level Data Link Control (HDLC)
        over MPLS Networks";
}

identity ppp {
    base bgp-l2-encaps-type;
    description
        "PPP.";
    reference
        "RFC 4618: Encapsulation Methods for Transport of
        PPP/High-Level Data Link Control (HDLC)
        over MPLS Networks";
}

identity circuit-emulation {
    base bgp-l2-encaps-type;
    description
        "SONET/SDH Circuit Emulation Service.";
    reference
        "RFC 4842: Synchronous Optical Network/Synchronous Digital
        Hierarchy (SONET/SDH) Circuit Emulation over Packet
        (CEP)";
}

identity atm-to-vcc {
    base bgp-l2-encaps-type;
    description
```

```
    "ATM n-to-one VCC cell transport.";
  reference
    "RFC 4717: Encapsulation Methods for Transport of
      Asynchronous Transfer Mode (ATM) over MPLS
      Networks";
}

identity atm-to-vpc {
  base bgp-l2-encaps-type;
  description
    "ATM n-to-one VPC cell transport.";
  reference
    "RFC 4717: Encapsulation Methods for Transport of
      Asynchronous Transfer Mode (ATM) over MPLS
      Networks";
}

identity layer-2-transport {
  base bgp-l2-encaps-type;
  description
    "IP Layer 2 Transport.";
  reference
    "RFC 3032: MPLS Label Stack Encoding";
}

identity fr-port-mode {
  base bgp-l2-encaps-type;
  description
    "Frame Relay Port mode.";
  reference
    "RFC 4619: Encapsulation Methods for Transport of Frame Relay
      over Multiprotocol Label Switching (MPLS)
      Networks";
}

identity e1 {
  base bgp-l2-encaps-type;
  description
    "Structure-agnostic E1 over packet.";
  reference
    "RFC 4553: Structure-Agnostic Time Division Multiplexing (TDM)
      over Packet (SAToP)";
}

identity t1 {
  base bgp-l2-encaps-type;
  description
    "Structure-agnostic T1 (DS1) over packet.";
```

```
    reference
      "RFC 4553: Structure-Agnostic Time Division Multiplexing (TDM)
        over Packet (SAToP)";
  }

  identity vpls {
    base bgp-l2-encaps-type;
    description
      "VPLS.";
    reference
      "RFC 4761: Virtual Private LAN Service (VPLS)
        Using BGP for Auto-Discovery and Signaling";
  }

  identity t3 {
    base bgp-l2-encaps-type;
    description
      "Structure-agnostic T3 (DS3) over packet.";
    reference
      "RFC 4553: Structure-Agnostic Time Division Multiplexing (TDM)
        over Packet (SAToP)";
  }

  identity structure-aware {
    base bgp-l2-encaps-type;
    description
      "Nx64kbit/s Basic Service using Structure-aware.";
    reference
      "RFC 5086: Structure-Aware Time Division Multiplexed (TDM)
        Circuit Emulation Service over Packet Switched
        Network (CESoPSN)";
  }

  identity dlci {
    base bgp-l2-encaps-type;
    description
      "Frame Relay DLCI.";
    reference
      "RFC 4619: Encapsulation Methods for Transport of Frame Relay
        over Multiprotocol Label Switching (MPLS)
        Networks";
  }

  identity e3 {
    base bgp-l2-encaps-type;
    description
      "Structure-agnostic E3 over packet.";
    reference
```

```
        "RFC 4553: Structure-Agnostic Time Division Multiplexing (TDM)
          over Packet (SAToP)";
    }

    identity ds1 {
        base bgp-l2-encaps-type;
        description
            "Octet-aligned payload for Structure-agnostic DS1 circuits.";
        reference
            "RFC 4553: Structure-Agnostic Time Division Multiplexing (TDM)
              over Packet (SAToP)";
    }

    identity cas {
        base bgp-l2-encaps-type;
        description
            "E1 Nx64kbit/s with CAS using Structure-aware.";
        reference
            "RFC 5086: Structure-Aware Time Division Multiplexed (TDM)
              Circuit Emulation Service over Packet Switched
              Network (CESoPSN)";
    }

    identity esf {
        base bgp-l2-encaps-type;
        description
            "DS1 (ESF) Nx64kbit/s with CAS using Structure-aware.";
        reference
            "RFC 5086: Structure-Aware Time Division Multiplexed (TDM)
              Circuit Emulation Service over Packet Switched
              Network (CESoPSN)";
    }

    identity sf {
        base bgp-l2-encaps-type;
        description
            "DS1 (SF) Nx64kbit/s with CAS using Structure-aware.";
        reference
            "RFC 5086: Structure-Aware Time Division Multiplexed (TDM)
              Circuit Emulation Service over Packet Switched
              Network (CESoPSN)";
    }
}
<CODE ENDS>
```

8.2. IANA-Maintained Module for Pseudowire Types

The initial version of the "iana-pseudowire-types" YANG module echoes the registry available at [IANA-PW-Types].

This module references [MFA], [RFC2507], [RFC2508], [RFC3032], [RFC3545], [RFC4448], [RFC4618], [RFC4619], [RFC4717], [RFC4842], [RFC4863], [RFC4901], [RFC5086], [RFC5087], [RFC5143], [RFC5795], and [RFC6307].

<CODE BEGINS>

```
file "iana-pseudowire-types@2021-07-05.yang"
```

```
module iana-pseudowire-types {
```

```
  yang-version 1.1;
```

```
  namespace "urn:ietf:params:xml:ns:yang:iana-pseudowire-types";
```

```
  prefix iana-pw-types;
```

```
  organization
```

```
    "IANA";
```

```
  contact
```

```
    "Internet Assigned Numbers Authority
```

```
    Postal: ICANN
```

```
      12025 Waterfront Drive, Suite 300
```

```
      Los Angeles, CA 90094-2536
```

```
      United States of America
```

```
    Tel: +1 310 301 5800
```

```
    <mailto:iana@iana.org>;
```

```
  description
```

```
    "This module contains a collection of IANA-maintained YANG  
    data types that are used for referring to Pseudowire Types.
```

```
    Copyright (c) 2022 IETF Trust and the persons identified as  
    authors of the code. All rights reserved.
```

```
    Redistribution and use in source and binary forms, with or  
    without modification, is permitted pursuant to, and subject  
    to the license terms contained in, the Revised BSD License  
    set forth in Section 4.c of the IETF Trust's Legal Provisions  
    Relating to IETF Documents  
    (https://trustee.ietf.org/license-info).
```

```
    This version of this YANG module is part of RFC XXXX; see  
    the RFC itself for full legal notices.";
```

```
  revision 2021-07-05 {
```

```
    description
```

```
      "First revision.";
```

```
    reference
      "RFC XXXX: A YANG Network Data Model for Layer 2 VPNs.";
  }

  identity iana-pw-types {
    description
      "Base Pseudowire Layer 2 encapsulation type.";
  }

  identity frame-relay {
    base iana-pw-types;
    description
      "Frame Relay DLCI (Martini Mode).";
    reference
      "RFC 4619: Encapsulation Methods for Transport of Frame Relay
        over Multiprotocol Label Switching (MPLS)
        Networks";
  }

  identity atm-aal5 {
    base iana-pw-types;
    description
      "ATM AAL5 SDU VCC transport.";
    reference
      "RFC 4717: Encapsulation Methods for Transport of
        Asynchronous Transfer Mode (ATM) over MPLS
        Networks";
  }

  identity atm-cell {
    base iana-pw-types;
    description
      "ATM transparent cell transport.";
    reference
      "RFC 4717: Encapsulation Methods for Transport of
        Asynchronous Transfer Mode (ATM) over MPLS
        Networks";
  }

  identity ethernet-tagged-mode {
    base iana-pw-types;
    description
      "Ethernet (VLAN) Tagged Mode.";
    reference
      "RFC 4448: Encapsulation Methods for Transport of Ethernet
        over MPLS Networks";
  }
```

```
identity ethernet {
  base iana-pw-types;
  description
    "Ethernet.";
  reference
    "RFC 4448: Encapsulation Methods for Transport of Ethernet
      over MPLS Networks";
}

identity hdlc {
  base iana-pw-types;
  description
    "HDLC.";
  reference
    "RFC 4618: Encapsulation Methods for Transport of
      PPP/High-Level Data Link Control (HDLC)
      over MPLS Networks";
}

identity ppp {
  base iana-pw-types;
  description
    "PPP.";
  reference
    "RFC 4618: Encapsulation Methods for Transport of
      PPP/High-Level Data Link Control (HDLC)
      over MPLS Networks";
}

identity circuit-emulation-mpls {
  base iana-pw-types;
  description
    "SONET/SDH Circuit Emulation Service Over MPLS Encapsulation.";
  reference
    "RFC 5143: Synchronous Optical Network/Synchronous Digital
      Hierarchy (SONET/SDH) Circuit Emulation Service over
      MPLS (CEM) Encapsulation";
}

identity atm-to-vcc {
  base iana-pw-types;
  description
    "ATM n-to-one VCC cell transport.";
  reference
    "RFC 4717: Encapsulation Methods for Transport of
      Asynchronous Transfer Mode (ATM) over MPLS
      Networks";
}
```

```
identity atm-to-vpc {
  base iana-pw-types;
  description
    "ATM n-to-one VPC cell transport.";
  reference
    "RFC 4717: Encapsulation Methods for Transport of
      Asynchronous Transfer Mode (ATM) over MPLS
      Networks";
}

identity layer-2-transport {
  base iana-pw-types;
  description
    "IP Layer2 Transport.";
  reference
    "RFC 3032: MPLS Label Stack Encoding";
}

identity atm-one-to-one-vcc {
  base iana-pw-types;
  description
    "ATM one-to-one VCC Cell Mode.";
  reference
    "RFC 4717: Encapsulation Methods for Transport of
      Asynchronous Transfer Mode (ATM) over MPLS
      Networks";
}

identity atm-one-to-one-vpc {
  base iana-pw-types;
  description
    "ATM one-to-one VPC Cell Mode.";
  reference
    "RFC 4717: Encapsulation Methods for Transport of
      Asynchronous Transfer Mode (ATM) over MPLS
      Networks";
}

identity atm-aal5-vcc {
  base iana-pw-types;
  description
    "ATM AAL5 PDU VCC transport.";
  reference
    "RFC 4717: Encapsulation Methods for Transport of
      Asynchronous Transfer Mode (ATM) over MPLS
      Networks";
}
```

```
identity fr-port-mode {
  base iana-pw-types;
  description
    "Frame-Relay Port mode.";
  reference
    "RFC 4619: Encapsulation Methods for Transport of Frame Relay
      over Multiprotocol Label Switching (MPLS)
      Networks";
}

identity circuit-emulation-packet {
  base iana-pw-types;
  description
    "SONET/SDH Circuit Emulation over Packet.";
  reference
    "RFC 4842: Synchronous Optical Network/Synchronous Digital
      Hierarchy (SONET/SDH) Circuit Emulation over Packet
      (CEP)";
}

identity e1 {
  base iana-pw-types;
  description
    "Structure-agnostic E1 over Packet.";
  reference
    "RFC 4553: Structure-Agnostic Time Division Multiplexing (TDM)
      over Packet (SAToP)";
}

identity t1 {
  base iana-pw-types;
  description
    "Structure-agnostic T1 (DS1) over Packet.";
  reference
    "RFC 4553: Structure-Agnostic Time Division Multiplexing (TDM)
      over Packet (SAToP)";
}

identity e3 {
  base iana-pw-types;
  description
    "Structure-agnostic E3 over Packet.";
  reference
    "RFC 4553: Structure-Agnostic Time Division Multiplexing (TDM)
      over Packet (SAToP)";
}

identity t3 {
```

```
    base iana-pw-types;
    description
        "Structure-agnostic T3 (DS3) over Packet.";
    reference
        "RFC 4553: Structure-Agnostic Time Division Multiplexing (TDM)
        over Packet (SAToP)";
}

identity ces-over-psn {
    base iana-pw-types;
    description
        "CESoPSN basic mode.";
    reference
        "RFC 5086: Structure-Aware Time Division Multiplexed (TDM)
        Circuit Emulation Service over Packet Switched
        Network (CESoPSN)";
}

identity tdm-over-ip-aal1 {
    base iana-pw-types;
    description
        "TDMoIP AAL1 Mode.";
    reference
        "RFC 5087: Time Division Multiplexing over IP (TDMoIP)";
}

identity ces-over-psn-cas {
    base iana-pw-types;
    description
        "CESoPSN TDM with CAS.";
    reference
        "RFC 5086: Structure-Aware Time Division Multiplexed (TDM)
        Circuit Emulation Service over Packet Switched
        Network (CESoPSN)";
}

identity tdm-over-ip-aal2 {
    base iana-pw-types;
    description
        "TDMoIP AAL2 Mode.";
    reference
        "RFC 5087: Time Division Multiplexing over IP (TDMoIP)";
}

identity dlci {
    base iana-pw-types;
    description
        "Frame Relay DLCI.";
```

```
reference
  "RFC 4619: Encapsulation Methods for Transport of Frame Relay
    over Multiprotocol Label Switching (MPLS)
    Networks";
}

identity rohc {
  base iana-pw-types;
  description
    "ROHC Transport Header-compressed Packets.";
  reference
    "RFC 5795: The RObust Header Compression (ROHC) Framework
      RFC 4901: Protocol Extensions for Header Compression over
      MPLS";
}

identity ecrtmp {
  base iana-pw-types;
  description
    "ECRTP Transport Header-compressed Packets.";
  reference
    "RFC 3545: Enhanced Compressed RTP (CRTP) for Links with High
      Delay, Packet Loss and Reordering
      RFC 4901: Protocol Extensions for Header Compression over
      MPLS";
}

identity iphc {
  base iana-pw-types;
  description
    "IPHC Transport Header-compressed Packets.";
  reference
    "RFC 2507: IP Header Compression
      RFC 4901: Protocol Extensions for Header Compression over
      MPLS";
}

identity crtp {
  base iana-pw-types;
  description
    "cRTP Transport Header-compressed Packets.";
  reference
    "RFC 2508: Compressing IP/UDP/RTP Headers for Low-Speed Serial
      Links
      RFC 4901: Protocol Extensions for Header Compression over
      MPLS";
}
```

```
identity atm-vp-virtual-trunk {
  base iana-pw-types;
  description
    "ATM VP Virtual Trunk.";
  reference
    "MFA Forum: The Use of Virtual Trunks for ATM/MPLS
      Control Plane Interworking Specification";
}

identity fc-port-mode {
  base iana-pw-types;
  description
    "FC Port Mode.";
  reference
    "RFC 6307: Encapsulation Methods for Transport of
      Fibre Channel Traffic over MPLS Networks";
}

identity wildcard {
  base iana-pw-types;
  description
    "Wildcard.";
  reference
    "RFC 4863: Wildcard Pseudowire Type";
}
}
<CODE ENDS>
```

8.3. Ethernet Segments

The "ietf-ethernet-segment" YANG module uses types defined in [RFC6991].

```
<CODE BEGINS>
file "ietf-ethernet-segment@2021-10-01.yang"
module ietf-ethernet-segment {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-ethernet-segment";
  prefix l2vpn-es;

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types, Section 3";
  }

  organization
    "IETF OPSA (Operations and Management Area) Working Group";
```

contact

"WG Web: <<https://datatracker.ietf.org/wg/opsawg/>>
WG List: <<mailto:opsawg@ietf.org>>

Editor: Mohamed Boucadair
<<mailto:mohamed.boucadair@orange.com>>
Editor: Samier Barguil
<<mailto:samier.barguilgiraldo.ext@telefonica.com>>
Author: Oscar Gonzalez de Dios
<<mailto:oscar.gonzalezdedios@telefonica.com>>";

description

"This YANG module defines a model for Ethernet Segments.

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

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Revised BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

revision 2021-10-01 {

description

"Initial version.";

reference

"RFC XXXX: A YANG Network Data Model for Layer 2 VPNs.";

}

/* Identities */

identity esi-type {

description

"T-(Ethernet Segment Identifier (ESI) Type) is a 1-octet field (most significant octet) that specifies the format of the remaining 9 octets (ESI Value).";

reference

"RFC 7432: BGP MPLS-Based Ethernet VPN, Section 5";

}

identity esi-type-0-operator {

base esi-type;

description

"This type indicates an arbitrary 9-octet ESI value,

```
        which is managed and configured by the operator.";
    }

    identity esi-type-1-lacp {
        base esi-type;
        description
            "When IEEE 802.1AX Link Aggregation Control Protocol (LACP)
            is used between the Provider Edge (PE) and Customer Edge (CE)
            devices, this ESI type indicates an auto-generated ESI value
            determined from LACP.";
        reference
            "IEEE Std. 802.1AX: Link Aggregation";
    }

    identity esi-type-2-bridge {
        base esi-type;
        description
            "The ESI value is auto-generated and determined based
            on the Layer 2 bridge protocol.";
    }

    identity esi-type-3-mac {
        base esi-type;
        description
            "This type indicates a MAC-based ESI value that can be
            auto-generated or configured by the operator.";
    }

    identity esi-type-4-router-id {
        base esi-type;
        description
            "This type indicates a Router ID ESI value that can be
            auto-generated or configured by the operator.";
    }

    identity esi-type-5-asn {
        base esi-type;
        description
            "This type indicates an Autonomous System (AS)-based ESI value
            that can be auto-generated or configured by the operator.";
    }

    identity df-election-methods {
        description
            "Base Identity Designated Forwarder (DF) election method.";
    }

    identity default-7432 {
```

```
base df-election-methods;
description
    "The default DF election method.

    The default procedure for DF election at the granularity of
    <ES,VLAN> for VLAN-based service or <ES, VLAN bundle> for
    VLAN-(aware) bundle service is referred to as
    'service carving'.";
reference
    "RFC 7432: BGP MPLS-Based Ethernet VPN, Section 8.5";
}

identity highest-random-weight {
    base df-election-methods;
    description
        "The highest random weight (HRW) method.";
    reference
        "RFC 8584: Framework for Ethernet VPN Designated
        Forwarder Election Extensibility, Section 3";
}

identity preference {
    base df-election-methods;
    description
        "The preference based method. PEs are assigned with
        preferences to become the DF in the Ethernet Segment (ES).
        The exact preference-based algorithm (e.g., lowest-preference
        algorithm, highest-preference algorithm) to use is
        signaled at the control plane.";
}

identity es-redundancy-mode {
    description
        "Base identity for ES redundancy modes.";
}

identity single-active {
    base es-redundancy-mode;
    description
        "Indicates Single-Active redundancy mode for a given ES.";
    reference
        "RFC 7432: BGP MPLS-Based Ethernet VPN, Section 14.1.1";
}

identity all-active {
    base es-redundancy-mode;
    description
        "Indicates All-Active redundancy mode for a given ES.";
```

```
reference
  "RFC 7432: BGP MPLS-Based Ethernet VPN, Section 14.1.2";
}

/* Main Ethernet Segment Container */

container ethernet-segments {
  description
    "Top container for the Ethernet Segment Identifier (ESI).";
  list ethernet-segment {
    key "name";
    description
      "Top list for ESIs.";
    leaf name {
      type string;
      description
        "Includes the name of the Ethernet Segment (ES).";
    }
    leaf esi-type {
      type identityref {
        base esi-type;
      }
      default "esi-type-0-operator";
      description
        "T-(ESI Type) is a 1-octet field (most significant
         octet) that specifies the format of the remaining
         9 octets (ESI Value).";
      reference
        "RFC 7432: BGP MPLS-Based Ethernet VPN, Section 5";
    }
    choice esi-choice {
      description
        "Ethernet segment choice between several types.
         For ESI Type 0: The esi is directly configured by the
         operator.
         For ESI Type 1: The auto-mode must be used.
         For ESI Type 2: The auto-mode must be used.
         For ESI Type 3: The directly-assigned or auto-mode must
         be used.
         For ESI Type 4: The directly-assigned or auto-mode must
         be used.
         For ESI Type 5: The directly-assigned or auto-mode must
         be used.";
      case directly-assigned {
        description
          "Explicitly assign an ESI value.";
        leaf ethernet-segment-identifier {
          type yang:hex-string {
```

```
        length "29";
    }
    description
        "10-octet ESI.";
}
}
case auto-assigned {
    description
        "The ESI is auto-assigned.";
    container esi-auto {
        description
            "The ESI is auto-assigned.";
        choice auto-mode {
            description
                "Indicates the auto-assignment mode. ESI can be
                automatically assigned either with or without
                indicating a pool from which the ESI should be
                taken.

                For both cases, the server will auto-assign an
                ESI value 'auto-assigned-ESI' and use that value
                operationally.";
            case from-pool {
                leaf esi-pool-name {
                    type string;
                    description
                        "The auto-assignment will be made from the
                        pool identified by the ESI-pool-name.";
                }
            }
            case full-auto {
                leaf auto {
                    type empty;
                    description
                        "Indicates an ESI is fully auto-assigned.";
                }
            }
        }
    }
}
leaf auto-ethernet-segment-identifier {
    type yang:hex-string {
        length "29";
    }
    config false;
    description
        "The value of the auto-assigned ESI.";
}
}
}
```

```
}
leaf esi-redundancy-mode {
  type identityref {
    base es-redundancy-mode;
  }
  description
    "Indicates the ES redundancy mode.";
  reference
    "RFC 7432: BGP MPLS-Based Ethernet VPN, Section 14.1";
}
container df-election {
  description
    "Top container for the DF election method properties.";
  leaf df-election-method {
    type identityref {
      base df-election-methods;
    }
    default "default-7432";
    description
      "Specifies the DF election method.";
    reference
      "RFC 8584: Framework for Ethernet VPN Designated
        Forwarder Election Extensibility";
  }
  leaf revertive {
    when "derived-from-or-self(..df-election-method, "
      + "'preference') " {
      description
        "The revertive value is only applicable
          to the preference method.";
    }
    type boolean;
    default "true";
    description
      "The 'preempt' or 'revertive' behavior. This
        option allows a non-revertive behavior in the
        DF election.";
    reference
      "RFC 8584: Framework for Ethernet VPN Designated
        Forwarder Election Extensibility";
  }
  leaf election-wait-time {
    type uint32;
    units "seconds";
    default "3";
    description
      "Election wait timer.";
    reference
```

```
        "RFC 8584: Framework for Ethernet VPN Designated
          Forwarder Election Extensibility";
    }
}
leaf split-horizon-filtering {
    type boolean;
    description
        "Controls split-horizon filtering. It is enabled
         when set to 'true'."

        In order to achieve split-horizon filtering, every
        Broadcast, unknown unicast, or multicast (BUM)
        packet originating from a non-DF PE is encapsulated
        with an MPLS label that identifies the origin ES.";
    reference
        "RFC 7432: BGP MPLS-Based Ethernet VPN, Section 8.3";
}
container pbb {
    description
        "Provider Backbone Bridging (PBB) parameters .";
    reference
        "IEEE 802.1ah: Provider Backbone Bridge";
    leaf backbone-src-mac {
        type yang:mac-address;
        description
            "The PEs connected to the same CE must share the
             same Provider Backbone (B-MAC) address in
             All-Active mode.";
        reference
            "RFC 7623: Provider Backbone Bridging Combined with
             Ethernet VPN (PBB-EVPN), Section 6.2.1.1";
    }
}
list member {
    key "ne-id interface-id";
    description
        "Includes a list of ES members.";
    leaf ne-id {
        type string;
        description
            "An identifier of the network element where the ES
             is configured within a service provider network.";
    }
    leaf interface-id {
        type string;
        description
            "Identifier of a node interface.";
    }
}
```

```
    }  
  }  
}  
<CODE ENDS>
```

8.4. L2NM

The "ietf-l2vpn-ntw" YANG module uses types defined in [RFC6991], [RFC9181], [RFC8294], and [IEEE802.1Qcp-2018].

```
<CODE BEGINS>  
file "ietf-l2vpn-ntw@2022-04-29.yang"  
module ietf-l2vpn-ntw {  
  yang-version 1.1;  
  namespace "urn:ietf:params:xml:ns:yang:ietf-l2vpn-ntw";  
  prefix l2vpn-ntw;  
  
  import ietf-inet-types {  
    prefix inet;  
    reference  
      "RFC 6991: Common YANG Data Types, Section 4";  
  }  
  import ietf-yang-types {  
    prefix yang;  
    reference  
      "RFC 6991: Common YANG Data Types, Section 3";  
  }  
  import ietf-vpn-common {  
    prefix vpn-common;  
    reference  
      "RFC 9181: A Common YANG for Data Model for Layer 2  
        and Layer 3 VPNs";  
  }  
  import iana-bgp-l2-encaps {  
    prefix iana-bgp-l2-encaps;  
    reference  
      "RFC XXXX: A YANG Network Data Model for Layer 2 VPNs.";  
  }  
  import iana-pseudowire-types {  
    prefix iana-pw-types;  
    reference  
      "RFC XXXX: A YANG Network Data Model for Layer 2 VPNs.";  
  }  
  import ietf-routing-types {  
    prefix rt-types;  
    reference  
      "RFC 8294: Common YANG Data Types for the Routing Area";  
  }  
}
```

```
}
import ieee802-dot1q-types {
  prefix dot1q-types;
  reference
    "IEEE Std 802.1Qcp-2018: Bridges and Bridged Networks -
      Amendment: YANG Data Model";
}

organization
  "IETF OPSA (Operations and Management Area) Working Group";
contact
  "WG Web:  <https://datatracker.ietf.org/wg/opsawg/>
  WG List:  <mailto:opsawg@ietf.org>

  Editor:    Mohamed Boucadair
             <mailto:mohamed.boucadair@orange.com>
  Editor:    Samier Barguil
             <mailto:samier.barguilgiraldo.ext@telefonica.com>
  Author:    Oscar Gonzalez de Dios
             <mailto:oscar.gonzalezdedios@telefonica.com>";
description
  "This YANG module defines a network model for Layer 2 VPN
  services.

  Copyright (c) 2022 IETF Trust and the persons identified as
  authors of the code.  All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Revised BSD License
  set forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (https://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";

revision 2022-04-29 {
  description
    "Initial version.";
  reference
    "RFC XXXX: A YANG Network Data Model for Layer 2 VPNs.";
}

/* Features */

feature oam-3ah {
  description
```

```
        "Indicates the support of OAM 802.3ah.";
    reference
        "IEEE Std 802.3ah: Media Access Control Parameters, Physical
          Layers, and Management Parameters for
          Subscriber Access Networks";
}

/* Identities */

identity evpn-service-interface-type {
    description
        "Base identity for EVPN service interface type.";
}

identity vlan-based-service-interface {
    base evpn-service-interface-type;
    description
        "VLAN-Based Service Interface.";
    reference
        "RFC 7432: BGP MPLS-Based Ethernet VPN, Section 6.1";
}

identity vlan-bundle-service-interface {
    base evpn-service-interface-type;
    description
        "VLAN Bundle Service Interface.";
    reference
        "RFC 7432: BGP MPLS-Based Ethernet VPN, Section 6.2";
}

identity vlan-aware-bundle-service-interface {
    base evpn-service-interface-type;
    description
        "VLAN-Aware Bundle Service Interface.";
    reference
        "RFC 7432: BGP MPLS-Based Ethernet VPN, Section 6.3";
}

identity mapping-type {
    base vpn-common:multicast-gp-address-mapping;
    description
        "Identity for multicast group mapping type.";
}

identity loop-prevention-type {
    description
        "Identity of loop prevention.";
}
```

```
identity shut {
  base loop-prevention-type;
  description
    "Shut protection type.";
}

identity trap {
  base loop-prevention-type;
  description
    "Trap protection type.";
}

identity color-type {
  description
    "Identity of color types. A type is assigned to a service frame
    to identify its QoS profile conformance.";
}

identity green {
  base color-type;
  description
    "'green' color type. A service frame is 'green' if it is
    conformant with the committed rate of the bandwidth profile.";
}

identity yellow {
  base color-type;
  description
    "'yellow' color type. A service frame is 'yellow' if it exceeds
    the committed rate but is conformant with the excess rate
    of the bandwidth profile.";
}

identity red {
  base color-type;
  description
    "'red' color type. A service frame is 'red' if it is not
    conformant with both the committed and excess rates of the
    bandwidth profile.";
}

identity t-ldp-pw-type {
  description
    "Identity for t-ldp-pw-type.";
}

identity vpws-type {
  base t-ldp-pw-type;
```

```
description
  "Virtual Private Wire Service (VPWS) t-ldp-pw-type.";
reference
  "RFC 4664: Framework for Layer 2 Virtual Private Networks
    (L2VPNs), Section 3.3";
}

identity vpls-type {
  base t-ldp-pw-type;
  description
    "Virtual Private LAN Service (VPLS) t-ldp-pw-type.";
  reference
    "RFC 4762: Virtual Private LAN Service (VPLS) Using
      Label Distribution Protocol (LDP)
      Signaling, Section 6.1";
}

identity hvpls {
  base t-ldp-pw-type;
  description
    "Identity for Hierarchical Virtual Private LAN Service (H-VPLS)
      t-ldp-pw-type.";
  reference
    "RFC 4762: Virtual Private LAN Service (VPLS) Using
      Label Distribution Protocol (LDP)
      Signaling, Section 10";
}

identity lacp-mode {
  description
    "Identity of the LACP mode.";
}

identity lacp-active {
  base lacp-mode;
  description
    "LACP active mode.

    This mode refers to the mode where auto-speed negotiation
    is initiated followed by an establishment of an
    Ethernet channel with the other end.";
}

identity lacp-passive {
  base lacp-mode;
  description
    "LACP passive mode.
```

```
        This mode refers to the LACP mode where an endpoint does
        not initiate the negotiation, but only responds to LACP
        packets initiated by the other end (e.g., full duplex
        or half duplex)";
    }

    identity pm-type {
        description
            "Identity for performance monitoring type.";
    }

    identity loss {
        base pm-type;
        description
            "Loss measurement is the performance monitoring type.";
    }

    identity delay {
        base pm-type;
        description
            "Delay measurement is the performance monitoring type.";
    }

    identity mac-learning-mode {
        description
            "Media Access Control (MAC) learning mode.";
    }

    identity data-plane {
        base mac-learning-mode;
        description
            "User MAC addresses are learned through ARP broadcast.";
    }

    identity control-plane {
        base mac-learning-mode;
        description
            "User MAC addresses are advertised through EVPN-BGP.";
    }

    identity mac-action {
        description
            "Base identity for a MAC action.";
    }

    identity drop {
        base mac-action;
        description
```

```
    "Dropping a packet as the MAC action.";
}

identity flood {
    base mac-action;
    description
        "Packet flooding as the MAC action.";
}

identity warning {
    base mac-action;
    description
        "Log a warning message as the MAC action.";
}

identity precedence-type {
    description
        "Redundancy type. The service can be created
        with primary and secondary signalization.";
}

identity primary {
    base precedence-type;
    description
        "Identifies the main VPN network access.";
}

identity secondary {
    base precedence-type;
    description
        "Identifies the secondary VPN network access.";
}

identity ldp-pw-type {
    description
        "Identity for allowed LDP-based pseudowire (PW) type.";
    reference
        "RFC 4762: Virtual Private LAN Service (VPLS) Using
        Label Distribution Protocol (LDP)
        Signaling, Section 6.1.1";
}

identity ethernet {
    base ldp-pw-type;
    description
        "PW Ethernet type.";
}
```

```
identity ethernet-tagged {
    base ldp-pw-type;
    description
        "PW Ethernet tagged mode type.";
}

/* Typedefs */

typedef ccm-priority-type {
    type uint8 {
        range "0..7";
    }
    description
        "A 3-bit priority value to be used in the VLAN tag,
        if present in the transmitted frame. A larger value
        indicates a higher priority.";
}

/* Groupings */

grouping cfm-802 {
    description
        "Grouping for 802.1ag Connectivity Fault Management (CFM)
        attributes.";
    reference
        "IEEE Std 802-1ag: Virtual Bridged Local Area Networks
        Amendment 5: Connectivity Fault Management";
    leaf maid {
        type string;
        description
            "Maintenance Association Identifier (MAID).";
    }
    leaf mep-id {
        type uint32;
        description
            "Local Maintenance Entity Group End Point (MEP) ID.";
    }
    leaf mep-level {
        type uint32;
        description
            "MEP level.";
    }
    leaf mep-up-down {
        type enumeration {
            enum up {
                description
                    "MEP is up.";
            }
        }
    }
}
```

```
        enum down {
            description
                "MEP is down.";
        }
    }
    default "up";
    description
        "MEP up/down.";
}
leaf remote-mep-id {
    type uint32;
    description
        "Remote MEP ID.";
}
leaf cos-for-cfm-pdus {
    type uint32;
    description
        "Class of service for CFM PDUs.";
}
leaf ccm-interval {
    type uint32;
    units "milliseconds";
    default "10000";
    description
        "Continuity Check Message (CCM) interval.";
}
leaf ccm-holdtime {
    type uint32;
    units "milliseconds";
    default "35000";
    description
        "CCM hold time.";
}
leaf ccm-p-bits-pri {
    type ccm-priority-type;
    description
        "The priority parameter for Continuity Check Messages (CCMs)
        transmitted by the MEP.";
}
}

grouping y-1731 {
    description
        "Grouping for Y-1731";
    reference
        "ITU-T Y-1731: Operations, administration and maintenance
        (OAM) functions and mechanisms for
        Ethernet-based networks";
}
```

```
list y-1731 {
  key "maid";
  description
    "List of configured Y-1731 instances.";
  leaf maid {
    type string;
    description
      "MAID.";
  }
  leaf mep-id {
    type uint32;
    description
      "Local MEP ID.";
  }
  leaf pm-type {
    type identityref {
      base pm-type;
    }
    default "delay";
    description
      "Performance monitor types.";
  }
  leaf remote-mep-id {
    type uint32;
    description
      "Remote MEP ID.";
  }
  leaf message-period {
    type uint32;
    units "milliseconds";
    default "10000";
    description
      "Defines the interval between OAM messages.";
  }
  leaf measurement-interval {
    type uint32;
    units "seconds";
    description
      "Specifies the measurement interval for statistics.";
  }
  leaf cos {
    type uint32;
    description
      "Identifies the Class of Service.";
  }
  leaf loss-measurement {
    type boolean;
    default "false";
  }
}
```

```
    description
      "Controls whether loss measurement is enabled/disabled.";
  }
  leaf synthetic-loss-measurement {
    type boolean;
    default "false";
    description
      "Indicates whether synthetic loss measurement is enabled.";
  }
  container delay-measurement {
    description
      "Container for delay measurement";
    leaf enable-dm {
      type boolean;
      default "false";
      description
        "Controls whether delay measurement is enabled ('true')
        or disabled ('false').";
    }
    leaf two-way {
      type boolean;
      default "false";
      description
        "Whether delay measurement is two-way ('true') of one-
        way ('false').";
    }
  }
  leaf frame-size {
    type uint32;
    units "bytes";
    description
      "Indicates the frame size.";
  }
  leaf session-type {
    type enumeration {
      enum proactive {
        description
          "Proactive mode.";
      }
      enum on-demand {
        description
          "On-demand mode.";
      }
    }
    default "on-demand";
    description
      "Specifies the session type.";
  }
}
```

```
    }  
  }  
  
  grouping parameters-profile {  
    description  
      "Container for per-service parameters.";  
    leaf local-autonomous-system {  
      type inet:as-number;  
      description  
        "Indicates a local AS Number (ASN).";  
    }  
    leaf svc-mtu {  
      type uint32;  
      description  
        "Layer 2 service MTU.  
        It is also known as the maximum transmission  
        unit or maximum frame size.";  
    }  
    leaf ce-vlan-preservation {  
      type boolean;  
      description  
        "Preserve the CE-VLAN ID from ingress to egress, i.e.,  
        CE-VLAN tag of the egress frame is identical to  
        that of the ingress frame that yielded this egress  
        service frame. If all-to-one bundling within a site  
        is enabled, then preservation applies to all ingress  
        service frames. If all-to-one bundling is disabled,  
        then preservation applies to tagged ingress service  
        frames having CE-VLAN ID 1 through 4094.";  
    }  
    leaf ce-vlan-cos-preservation {  
      type boolean;  
      description  
        "CE VLAN CoS preservation. Priority Code Point (PCP) bits  
        in the CE-VLAN tag of the egress frame are identical to  
        those of the ingress frame that yielded this egress  
        service frame.";  
    }  
    leaf control-word-negotiation {  
      type boolean;  
      description  
        "Controls whether Control-word negotiation is enabled  
        (if set to true) or not (if set to false).";  
      reference  
        "RFC 8077: Pseudowire Setup and Maintenance  
        Using the Label Distribution Protocol (LDP),  
        Section 7";  
    }  
  }
```

```
container mac-policies {
  description
    "Container of MAC policies.";
  container mac-addr-limit {
    description
      "Container of MAC address limit configuration.";
    leaf limit-number {
      type uint16;
      description
        "Maximum number of MAC addresses learned from
         the customer for a single service instance.
         The default value is '2' when this grouping
         is used at the service level";
    }
    leaf time-interval {
      type uint32;
      units "milliseconds";
      description
        "The aging time of the mac address.
         The default value is '300' when this grouping
         is used at the service level";
    }
    leaf action {
      type identityref {
        base mac-action;
      }
      description
        "Specifies the action when the upper limit is
         exceeded: drop the packet, flood the packet,
         or log a warning message (without dropping
         the packet).
         The default value is 'warning' when this
         grouping is used at the service level";
    }
  }
}
container mac-loop-prevention {
  description
    "Container for MAC loop prevention.";
  leaf window {
    type uint32;
    units "seconds";
    description
      "The time interval over which a MAC mobility event
       is detected and checked.
       The default value is '180' when this grouping
       is used at the service level";
  }
  leaf frequency {
```

```
    type uint32;
    description
        "The number of times to detect MAC duplication, where
        a 'duplicate MAC address' situation has occurred
        within the 'window' time interval and the duplicate
        MAC address has been added to a list of duplicate
        MAC addresses.
        The default value is '5' when this grouping is
        called at the service level";
}
leaf retry-timer {
    type uint32;
    units "seconds";
    description
        "The retry timer. When the retry timer expires,
        the duplicate MAC address will be flushed from
        the MAC-VRF.";
}
leaf protection-type {
    type identityref {
        base loop-prevention-type;
    }
    description
        "Protection type.
        The default value is 'trap' when this grouping
        is used at the service level";
}
}
}
container multicast {
    if-feature "vpn-common:multicast";
    description
        "Multicast container.";
    leaf enabled {
        type boolean;
        default "false";
        description
            "Enables multicast.";
    }
    container customer-tree-flavors {
        description
            "Type of trees used by the customer.";
        leaf-list tree-flavor {
            type identityref {
                base vpn-common:multicast-tree-type;
            }
            description
                "Type of multicast tree to be used.";
        }
    }
}
```

```
    }  
  }  
}  
  
grouping bandwidth-parameters {  
  description  
    "A grouping for bandwidth parameters.";  
  leaf cir {  
    type uint64;  
    units "bps";  
    description  
      "Committed Information Rate. The maximum  
      number of bits that a port can receive or  
      send during one-second over an  
      interface.";  
  }  
  leaf cbs {  
    type uint64;  
    units "bytes";  
    description  
      "Committed Burst Size. CBS controls the  
      bursty nature of the traffic. Traffic  
      that does not use the configured CIR  
      accumulates credits until the credits  
      reach the configured CBS.";  
  }  
  leaf eir {  
    type uint64;  
    units "bps";  
    description  
      "Excess Information Rate, i.e., excess  
      frame delivery allowed not subject to  
      SLA. The traffic rate can be limited  
      by EIR.";  
  }  
  leaf ebs {  
    type uint64;  
    units "bytes";  
    description  
      "Excess Burst Size. The bandwidth  
      available for burst traffic from the  
      EBS is subject to the amount of  
      bandwidth that is accumulated during  
      periods when traffic allocated by the  
      EIR policy is not used.";  
  }  
  leaf pir {
```

```
    type uint64;
    units "bps";
    description
        "Peak Information Rate, i.e., maximum
        frame delivery allowed. It is equal
        to or less than sum of CIR and EIR.";
}
leaf pbs {
    type uint64;
    units "bytes";
    description
        "Peak Burst Size.";
}
}

/* Main L2NM Container */

container l2vpn-ntw {
    description
        "Container for the L2NM.";
    container vpn-profiles {
        description
            "Container for VPN profiles.";
        uses vpn-common:vpn-profile-cfg;
    }
    container vpn-services {
        description
            "Container for L2VPN services.";
        list vpn-service {
            key "vpn-id";
            description
                "Container of a VPN service.";
            uses vpn-common:vpn-description;
            leaf parent-service-id {
                type vpn-common:vpn-id;
                description
                    "Pointer to the parent service that
                    triggered the L2NM.";
            }
            leaf vpn-type {
                type identityref {
                    base vpn-common:service-type;
                }
                must "not (derived-from-or-self(current(), "
                    + "'vpn-common:l3vpn'))" {
                    error-message "L3VPN is only applicable in L3NM.";
                }
            }
            description

```

```
        "Service type.";
    }
    leaf vpn-service-topology {
        type identityref {
            base vpn-common:vpn-topology;
        }
        description
            "Defining service topology, such as
             any-to-any, hub-spoke, etc.";
    }
    leaf bgp-ad-enabled {
        type boolean;
        description
            "Indicates whether BGP auto-discovery is enabled
             or disabled.";
    }
    leaf signaling-type {
        type identityref {
            base vpn-common:vpn-signaling-type;
        }
        description
            "VPN signaling type.";
    }
    container global-parameters-profiles {
        description
            "Container for a list of global parameters
             profiles.";
        list global-parameters-profile {
            key "profile-id";
            description
                "List of global parameters profiles.";
            leaf profile-id {
                type string;
                description
                    "The identifier of the global parameters profile.";
            }
            uses vpn-common:route-distinguisher;
            uses vpn-common:vpn-route-targets;
            uses parameters-profile;
        }
    }
    container underlay-transport {
        description
            "Container for the underlay transport.";
        uses vpn-common:underlay-transport;
    }
    uses vpn-common:service-status;
    container vpn-nodes {
```

```
description
  "Set of VPN nodes that are involved in the L2NM.";
list vpn-node {
  key "vpn-node-id";
  description
    "Container of the VPN nodes.";
  leaf vpn-node-id {
    type vpn-common:vpn-id;
    description
      "Sets the identifier of the VPN node.";
  }
  leaf description {
    type string;
    description
      "Textual description of a VPN node.";
  }
  leaf ne-id {
    type string;
    description
      "An identifier of the network element where
       the VPN node is deployed. This identifier
       uniquely identifies the network element within
       an administrative domain.";
  }
  leaf role {
    type identityref {
      base vpn-common:role;
    }
    default "vpn-common:any-to-any-role";
    description
      "Role of the VPN node in the VPN.";
  }
  leaf router-id {
    type rt-types:router-id;
    description
      "A 32-bit number in the dotted-quad format that is
       used to uniquely identify a node within an
       autonomous system (AS).";
  }
  container active-global-parameters-profiles {
    description
      "Container for a list of global parameters
       profiles.";
    list global-parameters-profile {
      key "profile-id";
      description
        "List of active global parameters profiles.";
      leaf profile-id {
```

```
    type leafref {
      path "../../../global-parameters-profiles"
        + "/global-parameters-profile/profile-id";
    }
    description
      "Points to a global profile defined at the
       service level.";
  }
  uses parameters-profile;
}
}
uses vpn-common:service-status;
container bgp-auto-discovery {
  when "../../../bgp-ad-enabled = 'true'" {
    description
      "Only applies when BGP auto-discovery is enabled.";
  }
  description
    "BGP is used for auto-discovery.";
  choice bgp-type {
    description
      "Choice for the BGP type.";
    case l2vpn-bgp {
      description
        "Container for BGP L2VPN.";
      leaf vpn-id {
        type vpn-common:vpn-id;
        description
          "VPN Identifier. This identifier serves to
           unify components of a given VPN for the
           sake of auto-discovery.";
        reference
          "RFC 6624: Layer 2 Virtual Private Networks
           Using BGP for Auto-Discovery and
           Signaling";
      }
    }
  }
  case evpn-bgp {
    description
      "EVPN case.";
    leaf evpn-type {
      type leafref {
        path "../../../vpn-type";
      }
      description
        "EVPN type.";
    }
    leaf auto-rt-enable {
```

```
        type boolean;
        default "false";
        description
            "Enables/disabled RT auto-derivation based on
            the ASN and Ethernet Tag ID.";
        reference
            "RFC 7432: BGP MPLS-Based Ethernet VPN,
            Section 7.10.1";
    }
    leaf auto-route-target {
        when "../auto-rt-enable = 'true'" {
            description
                "Can only be used when auto-RD is enabled.";
        }
        type rt-types:route-target;
        config false;
        description
            "The value of the auto-assigned RT.";
    }
}
uses vpn-common:route-distinguisher;
uses vpn-common:vpn-route-targets;
}
container signaling-option {
    description
        "Container for the L2VPN signaling.";
    leaf advertise-mtu {
        type boolean;
        description
            "Controls whether MTU is advertised.";
        reference
            "RFC 4667: Layer 2 Virtual Private Network (L2VPN)
            Extensions for Layer 2 Tunneling
            Protocol (L2TP), Section 4.3";
    }
    leaf mtu-allow-mismatch {
        type boolean;
        description
            "When set to true, it allows MTU mismatch.";
        reference
            "RFC 4667: Layer 2 Virtual Private Network (L2VPN)
            Extensions for Layer 2 Tunneling
            Protocol (L2TP), Section 4.3";
    }
    leaf signaling-type {
        type leafref {
            path "../.../.../signaling-type";
        }
    }
}
```

```
    }
    description
      "VPN signaling type.";
  }
  choice signaling-option {
    description
      "Choice for the signaling-option.";
    case bgp {
      description
        "BGP is used as the signaling protocol.";
      choice bgp-type {
        description
          "Choice for the BGP type.";
        case l2vpn-bgp {
          description
            "Container for BGP L2VPN.";
          leaf ce-range {
            type uint16;
            description
              "Determines the number of remote CEs with
               which a given CE can communicate in the
               context of a VPN.";
            reference
              "RFC 6624: Layer 2 Virtual Private Networks
               Using BGP for Auto-Discovery and
               Signaling";
          }
          leaf pw-encapsulation-type {
            type identityref {
              base iana-bgp-l2-encaps:bgp-l2-encaps-type;
            }
            description
              "PW encapsulation type.";
          }
        }
        container vpls-instance {
          when "derived-from-or-self(..../..../..../"
            + "vpn-type, 'vpn-common:vpls') " {
            description
              "Only applies for VPLS.";
          }
          description
            "VPLS instance.";
          leaf vpls-edge-id {
            type uint16;
            description
              "VPLS Edge Identifier (VE ID). This is
               used when the same VE ID is configured
               for the PE.";
          }
        }
      }
    }
  }
}
```

```
        reference
        "RFC 4761: Virtual Private LAN Service
        (VPLS) Using BGP for Auto-
        Discovery and Signaling,
        Section 3.5";
    }
    leaf vpls-edge-id-range {
        type uint16;
        description
            "Specifies the size of the range of
            VE ID in a VPLS service. The range
            controls the size of the label
            block advertised in the context of
            a VPLS instance.";
        reference
        "RFC 4761: Virtual Private LAN Service
        (VPLS) Using BGP for Auto-
        Discovery and Signaling";
    }
}
case evpn-bgp {
    description
        "Used for EVPN.";
    leaf evpn-type {
        type leafref {
            path "../..//bgp-auto-discovery/evpn-type";
        }
        description
            "EVPN type.";
    }
    leaf service-interface-type {
        type identityref {
            base evpn-service-interface-type;
        }
        description
            "EVPN service interface type.";
    }
}
container evpn-policies {
    description
        "Includes a set of EVPN policies such
        as those related to handling MAC
        addresses.";
    leaf mac-learning-mode {
        type identityref {
            base mac-learning-mode;
        }
        description
```

```
        "Indicates through which plane MAC
        addresses are advertised.";
    }
    leaf ingress-replication {
        type boolean;
        description
            "Controls whether ingress replication is
            enabled/disabled.";
        reference
            "RFC 7432: BGP MPLS-Based Ethernet VPN,
            Section 8.3.1.1";
    }
    leaf p2mp-replication {
        type boolean;
        description
            "Controls whether P2MP replication is
            enabled/disabled.";
        reference
            "RFC 7432: BGP MPLS-Based Ethernet VPN,
            Section 8.3.1.2";
    }
    container arp-proxy {
        if-feature "vpn-common:ipv4";
        description
            "Top container for the ARP proxy.";
        leaf enable {
            type boolean;
            default "false";
            description
                "Enables (when set to 'true') or
                disables (when set to 'false')
                ARP proxy.";
            reference
                "RFC 7432: BGP MPLS-Based Ethernet VPN,
                Section 10";
        }
        leaf arp-suppression {
            type boolean;
            default "false";
            description
                "Enables (when set to 'true') or
                disables (when set to 'false') ARP
                suppression.";
            reference
                "RFC 7432: BGP MPLS-Based Ethernet
                VPN";
        }
        leaf ip-mobility-threshold {
```

```
type uint16;
description
    "It is possible for a given host (as
    defined by its IP address) to move
    from one ES to another.
    IP mobility threshold specifies the
    number of IP mobility events
    that are detected for a given IP
    address within the
    detection-threshold before it
    is identified as a duplicate IP
    address.
    Once the detection threshold is
    reached, updates for the IP address
    are suppressed.";
}
leaf duplicate-ip-detection-interval {
    type uint16;
    units "seconds";
    description
        "The time interval used in detecting a
        duplicate IP address. Duplicate IP
        address detection number of host moves
        are allowed within this interval
        period.";
}
}
container nd-proxy {
    if-feature "vpn-common:ipv6";
    description
        "Top container for the ND proxy.";
    leaf enable {
        type boolean;
        default "false";
        description
            "Enables (when set to 'true') or
            disables (when set to 'false') ND
            proxy.";
        reference
            "RFC 7432: BGP MPLS-Based Ethernet VPN,
            Section 10";
    }
    leaf nd-suppression {
        type boolean;
        default "false";
        description
            "Enables (when set to 'true') or
            disables (when set to 'false')";
    }
}
```

```
Neighbor Discovery (ND) message
suppression.
ND suppression is a technique that
is used to reduce the amount of ND
packets flooding within individual
segments, that is between hosts
connected to the same logical
switch.";
}
leaf ip-mobility-threshold {
  type uint16;
  description
    "It is possible for a given host (as
    defined by its IP address) to move
    from one ES to another.
    IP mobility threshold specifies the
    number of IP mobility events
    that are detected for a given IP
    address within the
    detection-threshold before it
    is identified as a duplicate IP
    address.
    Once the detection threshold is
    reached, updates for the IP address
    are suppressed.";
}
leaf duplicate-ip-detection-interval {
  type uint16;
  units "seconds";
  description
    "The time interval used in detecting a
    duplicate IP address. Duplicate IP
    address detection number of host moves
    are allowed within this interval
    period.";
}
}
leaf underlay-multicast {
  type boolean;
  default "false";
  description
    "Enables (when set to 'true') or disables
    (when set to 'false') underlay
    multicast.";
}
leaf flood-unknown-unicast-supression {
  type boolean;
  default "false";
```

```
description
  "Enables (when set to 'true') or disables
   (when set to 'false') unknown flood
   unicast suppression.";
}
leaf vpws-vlan-aware {
  type boolean;
  default "false";
  description
    "Enables (when set to 'true') or disables
     (when set to 'false') VPWS VLAN-aware.";
}
container bum-management {
  description
    "Broadcast-unknown-unicast-multicast
     management.";
  leaf discard-broadcast {
    type boolean;
    default "false";
    description
      "Discards broadcast, when enabled.";
  }
  leaf discard-unknown-multicast {
    type boolean;
    default "false";
    description
      "Discards unknown multicast, when
       enabled.";
  }
  leaf discard-unknown-unicast {
    type boolean;
    default "false";
    description
      "Discards unknown unicast, when
       enabled.";
  }
}
container pbb {
  when "derived-from-or-self("
    + "../..//evpn-type, 'pbb-evpn') " {
    description
      "Only applies for PBB EVPN.";
  }
  description
    "PBB parameters container.";
  reference
    "IEEE 802.1ah: Provider Backbone Bridge";
  leaf backbone-src-mac {
```

```
        type yang:mac-address;
        description
            "Includes provider backbone MAC (B-MAC)
            address.";
        reference
            "RFC 7623: Provider Backbone Bridging
            Combined with Ethernet VPN
            (PBB-EVPN), Section 8.1";
    }
}
}
}
}
}
container ldp-or-l2tp {
    description
        "Container for LDP or L2TP-signaled PWs.";
    leaf agi {
        type rt-types:route-distinguisher;
        description
            "Attachment Group Identifier. Also, called
            VPLS-Id.";
        reference
            "RFC 4667: Layer 2 Virtual Private Network
            (L2VPN) Extensions for Layer 2
            Tunneling Protocol (L2TP),
            Section 4.3
            RFC 4762: Virtual Private LAN Service (VPLS)
            Using Label Distribution Protocol
            (LDP) Signaling, Section 6.1.1";
    }
    leaf saii {
        type uint32;
        description
            "Source Attachment Individual Identifier
            (SAII).";
        reference
            "RFC 4667: Layer 2 Virtual Private Network
            (L2VPN) Extensions for Layer 2
            Tunneling Protocol (L2TP),
            Section 3";
    }
    list remote-targets {
        key "taii";
        description
            "List of allowed target Attachment Individual
            Identifier (AII) and peers.";
        reference
```

```
        "RFC 4667: Layer 2 Virtual Private Network
          (L2VPN) Extensions for Layer 2
          Tunneling Protocol (L2TP),
          Section 5";
    leaf taii {
        type uint32;
        description
            "Target Attachment Individual Identifier.";
        reference
            "RFC 4667: Layer 2 Virtual Private Network
              (L2VPN) Extensions for Layer 2
              Tunneling Protocol (L2TP),
              Section 3";
    }
    leaf peer-addr {
        type inet:ip-address;
        description
            "Indicates the peer forwarder's IP address.";
    }
}
choice ldp-or-l2tp {
    description
        "Choice of LDP or L2TP-signaled PWs.";
    case ldp {
        description
            "Container for T-LDP PW configurations.";
        leaf t-ldp-pw-type {
            type identityref {
                base t-ldp-pw-type;
            }
            description
                "T-LDP PW type.";
        }
        leaf pw-type {
            type identityref {
                base ldp-pw-type;
            }
            description
                "PW encapsulation type.";
            reference
                "RFC 4762: Virtual Private LAN Service
                  (VPLS) Using Label Distribution
                  Protocol (LDP) Signaling,
                  Section 6.1.1";
        }
        leaf pw-description {
            type string;
            description
```

```
        "Includes a human-readable description
        of the interface. This may be used when
        communicating with a remote peer.";
    reference
        "RFC 4762: Virtual Private LAN Service
        (VPLS) Using Label Distribution
        Protocol (LDP) Signaling,
        Section 6.1.1";
}
leaf mac-addr-withdraw {
    type boolean;
    description
        "If set to 'true', then MAC address
        withdrawal is enabled. If 'false',
        then MAC address withdrawal is
        disabled.";
    reference
        "RFC 4762: Virtual Private LAN Service
        (VPLS) Using Label Distribution
        Protocol (LDP) Signaling,
        Section 6.2";
}
list pw-peer-list {
    key "peer-addr vc-id";
    description
        "List of AC and PW bindings.";
    leaf peer-addr {
        type inet:ip-address;
        description
            "Indicates the peer's IP address.";
    }
    leaf vc-id {
        type string;
        description
            "VC label used to identify a PW.";
    }
    leaf pw-priority {
        type uint32;
        description
            "Defines the priority for the PW.
            The higher the pw-priority value, the
            higher the preference of the PW will
            be.";
    }
}
}
container qinq {
    when "derived-from-or-self("
        + "../t-ldp-pw-type, 'hvpls')\" {
```

```
        description
          "Only applies when t-ldp pw type
           is h-vpls.";
      }
      description
        "Container for QinQ.";
      leaf s-tag {
        type dot1q-types:vlanid;
        mandatory true;
        description
          "S-TAG.";
      }
      leaf c-tag {
        type dot1q-types:vlanid;
        mandatory true;
        description
          "C-TAG.";
      }
    }
  }
  case l2tp {
    description
      "Container for L2TP PWs.";
    leaf router-id {
      type rt-types:router-id;
      description
        "A 32-bit number in the dotted-quad format
         that is used to uniquely identify a node
         within a service provider network.";
      reference
        "RFC 4667: Layer 2 Virtual Private Network
         (L2VPN) Extensions for Layer 2
         Tunneling Protocol (L2TP),
         Section 4.2";
    }
    leaf pseudowire-type {
      type identityref {
        base iana-pw-types:iana-pw-types;
      }
      description
        "Encapsulation type.";
      reference
        "RFC 4667: Layer 2 Virtual Private Network
         (L2VPN) Extensions for Layer 2
         Tunneling Protocol (L2TP),
         Section 4.2";
    }
  }
}
```

```
    }
  }
}
container vpn-network-accesses {
  description
    "Main container for VPN network accesses.";
  list vpn-network-access {
    key "id";
    description
      "List of VPN network accesses.";
    leaf id {
      type vpn-common:vpn-id;
      description
        "Identifier of the network access";
    }
    leaf description {
      type string;
      description
        "A textual description of the VPN network
        access.";
    }
    leaf interface-id {
      type string;
      description
        "Refers to a physical or logical interface.";
    }
    leaf active-vpn-node-profile {
      type leafref {
        path "../.../"
          + "/active-global-parameters-profiles"
          + "/global-parameters-profile/profile-id";
      }
      description
        "An identifier of an active VPN instance
        profile.";
    }
  }
  uses vpn-common:service-status;
  container connection {
    description
      "Container for the bearer and AC.";
    leaf l2-termination-point {
      type string;
      description
        "Specifies a reference to a local Layer 2
        termination point such as a Layer 2
        sub-interface.";
    }
  }
}
```

```
leaf local-bridge-reference {
  type string;
  description
    "Specifies a local bridge reference to
    accommodate, for example, implementations
    that require internal bridging.
    A reference may be a local bridge domain.";
}
leaf bearer-reference {
  if-feature "vpn-common:bearer-reference";
  type string;
  description
    "This is an internal reference for the service
    provider to identify the bearer associated
    with this VPN.";
}
container encapsulation {
  description
    "Container for Layer 2 encapsulation.";
  leaf encap-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 {
    when "derived-from-or-self(../encap-type, "
      + "'vpn-common:dot1q') " {
      description
        "Only applies when the type of the
        tagged interface is 'dot1q'.";
    }
    description
      "Tagged interface.";
    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 dot1q-types:vlanid;
description
  "VLAN identifier.";
}
container tag-operations {
  description
    "Sets the tag manipulation policy for this
    VPN network access. It defines a set of
    tag manipulations that allow for the
    insertion, removal, or rewriting
    of 802.1Q VLAN tags. These operations are
    indicated for the CE-PE direction.
    By default, tag operations are symmetric.
    As such, the reverse tag operation is
    assumed on the PE-CE direction.";
  choice op-choice {
    description
      "Selects the tag rewriting policy for a
      VPN network access.";
    leaf pop {
      type empty;
      description
        "Pop the outer tag.";
    }
    leaf push {
      type empty;
      description
        "Push one or two tags defined by the
        tag-1 and tag-2 leaves. It is
        assumed that, absent any policy, the
        default value of 0 will be used for
        PCP setting.";
    }
    leaf translate {
      type empty;
      description
        "Translate the outer tag to one or two
        tags. PCP bits are preserved.";
    }
  }
}
leaf tag-1 {
  when 'not(..pop)';
  type dot1q-types:vlanid;
  description
    "A first tag to be used for push or
    translate operations. This tag will be
    used as the outermost tag as a result
    of the tag operation.";
```

```
    }
    leaf tag-1-type {
      type dot1q-types:dot1q-tag-type;
      default "dot1q-types:s-vlan";
      description
        "Specifies a specific 802.1Q tag type
         of tag-1.";
    }
    leaf tag-2 {
      when '../translate';
      type dot1q-types:vlanid;
      description
        "A second tag to be used for
         translation.";
    }
    leaf tag-2-type {
      type dot1q-types:dot1q-tag-type;
      default "dot1q-types:c-vlan";
      description
        "Specifies a specific 802.1Q tag type
         of tag-2.";
    }
  }
}
container priority-tagged {
  when "derived-from-or-self(../encap-type, "
    + "'vpn-common:priority-tagged') " {
    description
      "Only applies when the type of the
       tagged interface is 'priority-tagged'.";
  }
  description
    "Priority tagged container.";
  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'.";
  }
}
container qinq {
  when "derived-from-or-self(../encap-type, "
    + "'vpn-common:qinq') " {
    description
      "Only applies when the type of the tagged
```

```
        interface is QinQ.";
    }
    description
        "Includes QinQ parameters.";
    leaf tag-type {
        type identityref {
            base vpn-common:tag-type;
        }
        default "vpn-common:s-c-vlan";
        description
            "Tag type. By default, the tag type is
             's-c-vlan'.";
    }
    leaf svlan-id {
        type dot1q-types:vlanid;
        mandatory true;
        description
            "S-VLAN identifier.";
    }
    leaf cvlan-id {
        type dot1q-types:vlanid;
        mandatory true;
        description
            "C-VLAN identifier.";
    }
    container tag-operations {
        description
            "Sets the tag manipulation policy for this
             VPN network access. It defines a set of
             tag manipulations that allow for the
             insertion, removal, or rewriting
             of 802.1Q VLAN tags. These operations are
             indicated for the CE-PE direction.
             By default, tag operations are symmetric.
             As such, the reverse tag operation is
             assumed on the PE-CE direction.";
        choice op-choice {
            description
                "Selects the tag rewriting policy for a
                 VPN network access.";
            leaf pop {
                type uint8 {
                    range "1|2";
                }
                description
                    "Pop one or two tags as a function
                     of the indicated pop value.";
            }
        }
    }
}
```

```
leaf push {
  type empty;
  description
    "Push one or two tags defined by the
    tag-1 and tag-2 leaves. It is
    assumed that, absent any policy, the
    default value of 0 will be used for
    PCP setting.";
}
leaf translate {
  type uint8 {
    range "1|2";
  }
  description
    "Translate one or two outer tags. PCP
    bits are preserved.

    The following operations are
    supported:

    - translate 1 with tag-1 leaf is
      provided: only the outermost tag is
      translated to the value in tag-1.

    - translate 2 with both tag-1 and
      tag-2 leaves are provided: both
      outer and inner tags are translated
      to the values in tag-1 and tag-2,
      respectively.

    - translate 2 with tag-1 leaf is
      provided: the outer tag is popped
      while the inner tag is translated
      to the value in tag-1.";
}
}
leaf tag-1 {
  when 'not(..../pop)';
  type dot1q-types:vlanid;
  description
    "A first tag to be used for push or
    translate operations. This tag will be
    used as the outermost tag as a result
    of the tag operation.";
}
leaf tag-1-type {
  type dot1q-types:dot1q-tag-type;
  default "dot1q-types:s-vlan";
}
```

```
        description
        "Specifies a specific 802.1Q tag type
        of tag-1.";
    }
    leaf tag-2 {
        when 'not(..pop)';
        type dot1q-types:vlanid;
        description
        "A second tag to be used for push or
        translate operations.";
    }
    leaf tag-2-type {
        type dot1q-types:dot1q-tag-type;
        default "dot1q-types:c-vlan";
        description
        "Specifies a specific 802.1Q tag type
        of tag-2.";
    }
}
}
}
container lag-interface {
    if-feature "vpn-common:lag-interface";
    description
    "Container of LAG interface attributes
    configuration.";
    leaf lag-interface-id {
        type string;
        description
        "LAG interface identifier.";
    }
}
container lacp {
    description
    "Container for LACP.";
    leaf lacp-state {
        type boolean;
        default "false";
        description
        "Controls whether LACP is enabled.";
    }
    leaf mode {
        type identityref {
            base lacp-mode;
        }
        description
        "Indicates the LACP mode.";
    }
    leaf speed {
```

```
    type uint32;
    units "mbps";
    default "10";
    description
        "LACP speed. This low default value
         is inherited from the L2SM.";
}
leaf mini-link-num {
    type uint32;
    description
        "Defines the minimum number of links that
         must be active before the aggregating
         link is put into service.";
}
leaf system-id {
    type yang:mac-address;
    description
        "Indicates the System ID used by LACP.";
}
leaf admin-key {
    type uint16;
    description
        "Indicates the value of the key used for
         the aggregate interface.";
}
leaf system-priority {
    type uint16 {
        range "0..65535";
    }
    default "32768";
    description
        "Indicates the LACP priority for the
         system.";
}
container member-link-list {
    description
        "Container of Member link list.";
    list member-link {
        key "name";
        description
            "Member link.";
        leaf name {
            type string;
            description
                "Member link name.";
        }
        leaf speed {
            type uint32;
        }
    }
}
```

```
        units "mbps";
        default "10";
        description
            "Port speed.";
    }
    leaf mode {
        type identityref {
            base vpn-common:neg-mode;
        }
        description
            "Negotiation mode.";
    }
    leaf link-mtu {
        type uint32;
        description
            "Link MTU size.";
    }
    container oam-802.3ah-link {
        if-feature "oam-3ah";
        description
            "Container for oam 802.3ah link.";
        leaf enable {
            type boolean;
            default "false";
            description
                "Indicates support of OAM 802.3ah
                 link.";
        }
    }
}

leaf flow-control {
    type boolean;
    default "false";
    description
        "Indicates whether flow control is
         supported.";
}

leaf lldp {
    type boolean;
    default "false";
    description
        "Indicates whether Link Layer Discovery
         Protocol (LLDP) is supported.";
}

container split-horizon {
    description
```

```
        "Configuration with split horizon enabled.";
    leaf group-name {
        type string;
        description
            "Group name of the Split Horizon.";
    }
}
}
choice signaling-option {
    description
        "Choice for the signaling-option.";
    case bgp {
        description
            "BGP is used as the signaling protocol.";
        choice bgp-type {
            description
                "Choice for the BGP type.";
            case l2vpn-bgp {
                description
                    "Container for BGP L2VPN.";
                leaf ce-id {
                    type uint16;
                    description
                        "Identifies the CE within the VPN.";
                    reference
                        "RFC 6624: Layer 2 Virtual Private
                        Networks Using BGP for
                        Auto-Discovery and
                        Signaling";
                }
                leaf remote-ce-id {
                    type uint16;
                    description
                        "Indicates the identifier of the remote
                        CE.";
                }
            }
            case vpls {
                description
                    "VPLS instance."
                    when "derived-from-or-self ../../../../.."
                        + "vpn-type, 'vpn-common:vpls'" {
                        description
                            "Only applies for VPLS.";
                    }
                leaf vpls-edge-id {
                    type uint16;
                    description

```

```
        "VPLS Edge Identifier (VE ID).";
    reference
        "RFC 4761: Virtual Private LAN Service
        (VPLS) Using BGP for Auto-
        Discovery and Signaling,
        Section 3.2.1";
    }
}
}
case evpn-bgp {
    description
        "Used for EVPN.";
    leaf df-preference {
        type uint16;
        default "32767";
        description
            "Defines a 2-octet value that indicates
            the PE preference to become the DF in
            the ES.

            The preference value is only applicable
            to the preference based method.";
        reference
            "RFC 8584: Framework for Ethernet VPN
            Designated Forwarder Election
            Extensibility";
    }
    container vpws-service-instance {
        when "derived-from-or-self ../../../../"
            + "vpn-type, 'vpn-common:vpws-evpn'" {
            description
                "Only applies for EVPN-VPWS.";
        }
        description
            "Local and remote VPWS Service Instance
            (VSI)";
        reference
            "RFC 8214: Virtual Private Wire Service
            Support in Ethernet VPN";
        choice local-vsi-choice {
            description
                "Choices for assigning local VSI.";
            case directly-assigned {
                description
                    "Explicitly assign a local VSI.";
                leaf local-vpws-service-instance {
                    type uint32 {
                        range "1..16777215";
                    }
                }
            }
        }
    }
}
```

```
    }
    description
      "Indicates the assigned local
       VSI.";
  }
}
case auto-assigned {
  description
    "The local VSI is auto-assigned.";
  container local-vsi-auto {
    description
      "The local VSI is auto-assigned.";
    choice auto-mode {
      description
        "Indicates the auto-assignment
         mode of local VSI. VSI can be
         automatically assigned either
         with or without indicating a
         pool from which the VSI
         should be taken.

         For both cases, the server
         will auto-assign a local VSI
         value and use that value.";
      case from-pool {
        leaf vsi-pool-name {
          type string;
          description
            "The auto-assignment will be
             made from this pool.";
        }
      }
    }
    case full-auto {
      leaf auto {
        type empty;
        description
          "Indicates that a local VSI
           is fully auto-assigned.";
      }
    }
  }
}
leaf auto-local-vsi {
  type uint32 {
    range "1..16777215";
  }
  config false;
  description
    "The value of the auto-assigned
```

```
        local VSI.";
    }
}
}
choice remote-vsi-choice {
    description
        "Choice for assigning the remote VSI.";
    case directly-assigned {
        description
            "Explicitly assign a remote VSI.";
        leaf remote-vpws-service-instance {
            type uint32 {
                range "1..16777215";
            }
            description
                "Indicates the value of the remote
                VSI.";
        }
    }
    case auto-assigned {
        description
            "The remote VSI is auto-assigned.";
        container remote-vsi-auto {
            description
                "The remote VSI is auto-assigned.";
            choice auto-mode {
                description
                    "Indicates the auto-assignment
                    mode of remote VSI. VSI can be
                    automatically assigned either
                    with or without indicating a
                    pool from which the VSI
                    should be taken.

                    For both cases, the server
                    will auto-assign a remote VSI
                    value and use that value.";
                case from-pool {
                    leaf vsi-pool-name {
                        type string;
                        description
                            "The auto-assignment will be
                            made from this pool.";
                    }
                }
                case full-auto {
                    leaf auto {
```

```
        type empty;
        description
            "Indicates that a remote VSI
             is fully auto-assigned.";
    }
}
}
leaf auto-remote-vsi {
    type uint32 {
        range "1..16777215";
    }
    config false;
    description
        "The value of the auto-assigned
         remote VSI.";
}
}
}
}
}
}
}
}
}
list group {
    key "group-id";
    description
        "List of group-ids.";
    leaf group-id {
        type string;
        description
            "Indicates the group-id to which the network
             access belongs to.";
    }
    leaf precedence {
        type identityref {
            base precedence-type;
        }
        description
            "Defining service redundancy in transport
             network.";
    }
    leaf ethernet-segment-identifier {
        type string;
        description
            "Reference to the ESI associated with the VPN
             network access.";
    }
}
```

```
}
container ethernet-service-oam {
  description
    "Container for Ethernet service OAM.";
  leaf md-name {
    type string;
    description
      "Maintenance domain name.";
  }
  leaf md-level {
    type uint8;
    description
      "Maintenance domain level.";
  }
  container cfm-802.1-ag {
    description
      "Container of 802.1ag CFM configurations.";
    list n2-uni-c {
      key "maid";
      description
        "List of UNI-N to UNI-C.";
      uses cfm-802;
    }
    list n2-uni-n {
      key "maid";
      description
        "List of UNI-N to UNI-N.";
      uses cfm-802;
    }
  }
  uses y-1731;
}
container service {
  description
    "Container for service";
  leaf mtu {
    type uint32;
    description
      "Layer 2 MTU, it is also known as the maximum
        transmission unit or maximum frame size.";
  }
  container svc-pe-to-ce-bandwidth {
    if-feature "vpn-common:inbound-bw";
    description
      "From the customer site's perspective, the
        service inbound bandwidth of the connection
        or download bandwidth from the service
        provider the site. Note that the L2SM uses
```

```
        'input-bandwidth' to refer to the same
        concept.";
list pe-to-ce-bandwidth {
  key "bw-type";
  description
    "List for PE-to-CE bandwidth data nodes.";
  leaf bw-type {
    type identityref {
      base vpn-common:bw-type;
    }
    description
      "Indicates the bandwidth type.";
  }
  choice type {
    description
      "Choice based upon bandwidth type.";
    case per-cos {
      description
        "Bandwidth per CoS.";
      list cos {
        key "cos-id";
        description
          "List of class of services.";
        leaf cos-id {
          type uint8;
          description
            "Identifier of the CoS, indicated by
            DSCP or a CE-CLAN CoS (802.1p) value
            in the service frame.";
          reference
            "IEEE Std 802.1Q: Bridges and Bridged
            Networks";
        }
        uses bandwidth-parameters;
      }
    }
    case other {
      description
        "Other bandwidth types.";
      uses bandwidth-parameters;
    }
  }
}
}
}
container svc-ce-to-pe-bandwidth {
  if-feature "vpn-common:outbound-bw";
  description
    "From the customer site's perspective,
```

```
the service outbound bandwidth of the
connection or upload bandwidth from
the CE to the PE. Note that the L2SM uses
'output-bandwidth' to refer to the same
concept.";
list ce-to-pe-bandwidth {
  key "bw-type";
  description
    "List for CE-to-PE bandwidth.";
  leaf bw-type {
    type identityref {
      base vpn-common:bw-type;
    }
    description
      "Indicates the bandwidth type.";
  }
  choice type {
    description
      "Choice based upon bandwidth type.";
    case per-cos {
      description
        "Bandwidth per CoS.";
      list cos {
        key "cos-id";
        description
          "List of class of services.";
        leaf cos-id {
          type uint8;
          description
            "Identifier of the CoS, indicated by
            DSCP or a CE-CLAN CoS (802.1p) value
            in the service frame.";
          reference
            "IEEE Std 802.1Q: Bridges and Bridged
            Networks";
        }
        uses bandwidth-parameters;
      }
    }
    case other {
      description
        "Other non CoS-aware bandwidth types.";
      uses bandwidth-parameters;
    }
  }
}
}
container qos {
```

```
if-feature "vpn-common:qos";
description
  "QoS configuration.";
container qos-classification-policy {
  description
    "Configuration of the traffic classification
    policy.";
  list rule {
    key "id";
    ordered-by user;
    description
      "List of classification rules.";
    leaf id {
      type string;
      description
        "A description identifying the QoS
        classification policy rule.";
    }
    choice match-type {
      default "match-flow";
      description
        "Choice for classification.";
      case match-flow {
        container match-flow {
          description
            "Describes flow-matching criteria.";
          leaf dscp {
            type inet:dscp;
            description
              "DSCP value.";
          }
          leaf dot1q {
            type uint16;
            description
              "802.1Q matching. It is a VLAN tag
              added into a frame.";
            reference
              "IEEE Std 802.1Q: Bridges and
              Bridged
              Networks";
          }
          leaf pcps {
            type uint8 {
              range "0..7";
            }
            description
              "Priority Code Point (PCP) value.";
          }
        }
      }
    }
  }
}
```

```
    leaf src-mac-address {
      type yang:mac-address;
      description
        "Source MAC address.";
    }
    leaf dst-mac-address {
      type yang:mac-address;
      description
        "Destination MAC address.";
    }
    leaf color-type {
      type identityref {
        base color-type;
      }
      description
        "Color type.";
    }
    leaf any {
      type empty;
      description
        "Allows all.";
    }
  }
}
case match-application {
  leaf match-application {
    type identityref {
      base vpn-common:customer-application;
    }
    description
      "Defines the application to match.";
  }
}
}
leaf target-class-id {
  type string;
  description
    "Identification of the CoS.
    This identifier is internal to the
    administration.";
}
}
}
container qos-profile {
  description
    "QoS profile configuration.";
  list qos-profile {
    key "profile";
```

```
    description
      "QoS profile.
       Can be standard profile or customized
       profile.";
    leaf profile {
      type leafref {
        path "/l2vpn-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.";
    }
  }
}
}
container mac-policies {
  description
    "Container for MAC-related policies.";
  list access-control-list {
    key "name";
    description
      "Container for access control List.";
    leaf name {
      type string;
      description
        "Specifies the name of the ACL.";
    }
    leaf-list src-mac-address {
      type yang:mac-address;
      description
        "Specifies the source MAC address.";
    }
    leaf-list src-mac-address-mask {
      type yang:mac-address;
      description
        "Specifies the source MAC address mask.";
    }
    leaf-list dst-mac-address {
```

```
    type yang:mac-address;
    description
      "Specifies the destination MAC address.";
  }
  leaf-list dst-mac-address-mask {
    type yang:mac-address;
    description
      "Specifies the destination MAC address
      mask.";
  }
  leaf action {
    type identityref {
      base mac-action;
    }
    default "drop";
    description
      "Specifies the filtering action.";
  }
  leaf rate-limit {
    when "derived-from-or-self(.. / action, "
      + "'flood') " {
      description
        "Rate-limit is valid only when the action
        is to accept the matching frame.";
    }
    type decimal64 {
      fraction-digits 2;
    }
    units "bytes per second";
    description
      "Specifies how to rate-limit the traffic.";
  }
}
container mac-loop-prevention {
  description
    "Container of MAC loop prevention.";
  leaf window {
    type uint32;
    units "seconds";
    default "180";
    description
      "The timer when a MAC mobility event is
      detected.";
  }
  leaf frequency {
    type uint32;
    default "5";
    description
```

```
        "The number of times to detect MAC
        duplication, where a 'duplicate MAC
        address' situation has occurred and
        the duplicate MAC address has been
        added to a list of duplicate MAC
        addresses.";
    }
    leaf retry-timer {
        type uint32;
        units "seconds";
        description
            "The retry timer. When the retry timer
            expires, the duplicate MAC address will
            be flushed from the MAC-VRF.";
    }
    leaf protection-type {
        type identityref {
            base loop-prevention-type;
        }
        default "trap";
        description
            "Protection type";
    }
}
container mac-addr-limit {
    description
        "Container of MAC-Addr limit configurations";
    leaf limit-number {
        type uint16;
        default "2";
        description
            "Maximum number of MAC addresses learned
            from the subscriber for a single service
            instance.";
    }
    leaf time-interval {
        type uint32;
        units "milliseconds";
        default "300";
        description
            "The aging time of the mac address.";
    }
    leaf action {
        type identityref {
            base mac-action;
        }
        default "warning";
        description
```

```
        "Specifies the action when the upper limit
        is exceeded: drop the packet, flood the
        packet, or log a warning message (without
        dropping the packet).";
    }
}
}
container broadcast-unknown-unicast-multicast {
  description
    "Container of broadcast, unknown unicast, and
    multicast configurations";
  leaf multicast-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 the multicast site.";
  }
  list multicast-gp-address-mapping {
    key "id";
    description
      "List of Port to group mappings.";
    leaf id {
      type uint16;
      description
        "Unique identifier for the mapping.";
    }
    leaf vlan-id {
      type uint32;
      mandatory true;
      description
        "The VLAN ID of the multicast group.";
    }
    leaf mac-gp-address {
      type yang:mac-address;
    }
  }
}
```

```

        mandatory true;
        description
            "The MAC address of the multicast group.";
    }
    leaf port-lag-number {
        type uint32;
        description
            "The port/LAG belonging to the multicast
             group.";
    }
}
leaf bum-overall-rate {
    type uint64;
    units "bps";
    description
        "Overall rate for BUM.";
}
}
}
}
}
}
}
}
}
}
<CODE ENDS>
```

9. Security Considerations

The YANG modules specified in this document defines schemas for data that are designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The 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.

There are a number of data nodes defined in "ietf-l2vpn-ntw" and "ietf-ethernet-segment" YANG modules 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-l2vpn-ntw" and "ietf-ethernet-segment" modules:

- * 'vpn-profiles': This container includes a set of sensitive data that influence how the L3VPN service is delivered. For example, an attacker who has access to these data nodes may be able to manipulate routing policies, QoS policies, or encryption properties. These data nodes are defined with "nacm:default-deny-write" tagging [RFC9181].
- * 'ethernet-segments' and 'vpn-services': An attacker who is able to access network nodes can undertake various attacks, such as deleting a running L2VPN service, interrupting all the traffic of a client. In addition, an attacker may modify the attributes of a running service (e.g., QoS, bandwidth) or an ES, leading to malfunctioning of the service and therefore to SLA violations. In addition, an attacker could attempt to create an L2VPN service, add a new network access, or intercept/redirect the traffic to a non-authorized node. In addition to using NACM to prevent unauthorized access, such activity can be detected by adequately monitoring and tracking network configuration changes.

Some of the readable data nodes in the "ietf-l2vpn-ntw" YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

- * '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.

Both "iana-bgp-l2-encaps" and "iana-pseudowire-types" modules define YANG identities for encapsulation/pseudowires types. These identities are intended to be referenced by other YANG modules, and by themselves do not expose any nodes which are writable, contain read-only state, or RPCs.

10. IANA Considerations

10.1. Registering YANG Modules

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

URI: urn:ietf:params:xml:ns:yang:iana-bgp-l2-encaps
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

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

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

URI: urn:ietf:params:xml:ns:yang:ietf-l2vpn-ntw
Registrant Contact: The IESG.
XML: N/A; the requested URI is an XML namespace.

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

name: iana-bgp-l2-encaps
namespace: urn:ietf:params:xml:ns:yang:iana-bgp-l2-encaps
maintained by IANA: Y
prefix: iana-bgp-l2-encaps
reference: RFC XXXX

name: iana-pseudowire-types
namespace: urn:ietf:params:xml:ns:yang:iana-pseudowire-types
maintained by IANA: Y
prefix: iana-pw-types
reference: RFC XXXX

name: ietf-ethernet-segment
namespace: urn:ietf:params:xml:ns:yang:ietf-ethernet-segment
maintained by IANA: N
prefix: l2vpn-es
reference: RFC XXXX

name: ietf-l2vpn-ntw
namespace: urn:ietf:params:xml:ns:yang:ietf-l2vpn-ntw
maintained by IANA: N
prefix: l2vpn-ntw
reference: RFC XXXX

10.2. BGP Layer 2 Encapsulation Types

This document defines the initial version of the IANA-maintained "iana-bgp-l2-encaps" YANG module (Section 8.1). IANA is requested to add this note to the registry:

BGP Layer 2 encapsulation types must not be directly added to the "iana-bgp-l2-encaps" YANG module. They must instead be added to the "BGP Layer 2 Encapsulation Types" registry [IANA-BGP-L2].

When a Layer 2 encapsulation type is added to the "BGP Layer 2 Encapsulation Types" registry, a new "identity" statement must be added to the "iana-bgp-l2-encaps" YANG module. The name of the "identity" is a lower-case version of the encapsulation name provided in the description. The "identity" statement should have the following sub-statements defined:

"base": Contains 'bgp-l2-encaps-type'.

"description": Replicates the description from the registry.

"reference": Replicates the reference from the registry with the title of the document added.

Unassigned or reserved values are not present in the module.

When the "iana-bgp-l2-encaps" YANG module is updated, a new "revision" statement with a unique revision date must be added in front of the existing revision statements.

IANA is requested to add this note to [IANA-BGP-L2]:

When this registry is modified, the YANG module "iana-bgp-l2-encaps" must be updated as defined in RFCXXXX.

10.3. Pseudowire Types

This document defines the initial version of the IANA-maintained "iana-pseudowire-types" YANG module (Section 8.2). IANA is requested to add this note to the registry:

MPLS pseudowire types must not be directly added to the "iana-bgp-l2-encaps" YANG module. They must instead be added to the "MPLS Pseudowire Types" registry [IANA-PW-Types].

When a pseudowire type is added to the "iana-pseudowire-types" registry, a new "identity" statement must be added to the "iana-pseudowire-types" YANG module. The name of the "identity" is a

lower-case version of the encapsulation name provided in the description. The "identity" statement should have the following sub-statements defined:

"base": Contains 'iana-pw-types'.

"description": Replicates the description from the registry.

"reference": Replicates the reference from the registry with the title of the document added

Unassigned or reserved values are not present in the module.

When the "iana-pseudowire-types" YANG module is updated, a new "revision" statement with a unique revision date must be added in front of the existing revision statements.

IANA is requested to add this note to [IANA-PW-Types]:

When this registry is modified, the YANG module "iana-pseudowire-types" must be updated as defined in RFCXXXX.

11. References

11.1. Normative References

[IANA-BGP-L2]

IANA, "BGP Layer 2 Encapsulation Types",
<<https://www.iana.org/assignments/bgp-parameters/bgp-parameters.xhtml#bgp-l2-encapsulation-types-registry>>.

[IANA-PW-Types]

IANA, "MPLS Pseudowire Types Registry",
<<http://www.iana.org/assignments/pwe3-parameters/pwe3-parameters.xhtml#pwe3-parameters-2>>.

[IEEE-802-1ag]

IEEE, "802.1ag - 2007 - IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 5: Connectivity Fault Management", 2007, <DOI 10.1109/IEEESTD.2007.4431836>.

[IEEE802.1Qcp-2018]

IEEE, "IEEE Standard for Local and metropolitan area networks--Bridges and Bridged Networks--Amendment 30: YANG Data Model", September 2018,
<<https://ieeexplore.ieee.org/document/8467507>>.

- [ITU-T-Y-1731] Union, I. T., "Operations, administration and maintenance (OAM) functions and mechanisms for Ethernet-based networks", August 2015, <<https://www.itu.int/rec/T-REC-Y.1731/en>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC4667] Luo, W., "Layer 2 Virtual Private Network (L2VPN) Extensions for Layer 2 Tunneling Protocol (L2TP)", RFC 4667, DOI 10.17487/RFC4667, September 2006, <<https://www.rfc-editor.org/info/rfc4667>>.
- [RFC4761] Kompella, K., Ed. and Y. Rekhter, Ed., "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling", RFC 4761, DOI 10.17487/RFC4761, January 2007, <<https://www.rfc-editor.org/info/rfc4761>>.
- [RFC4762] Lasserre, M., Ed. and V. Kompella, Ed., "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling", RFC 4762, DOI 10.17487/RFC4762, January 2007, <<https://www.rfc-editor.org/info/rfc4762>>.
- [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>>.
- [RFC6074] Rosen, E., Davie, B., Radoaca, V., and W. Luo, "Provisioning, Auto-Discovery, and Signaling in Layer 2 Virtual Private Networks (L2VPNs)", RFC 6074, DOI 10.17487/RFC6074, January 2011, <<https://www.rfc-editor.org/info/rfc6074>>.
- [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>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.

- [RFC7432] Sajassi, A., Ed., Aggarwal, R., Bitar, N., Isaac, A., Uttaro, J., Drake, J., and W. Henderickx, "BGP MPLS-Based Ethernet VPN", RFC 7432, DOI 10.17487/RFC7432, February 2015, <<https://www.rfc-editor.org/info/rfc7432>>.
- [RFC7623] Sajassi, A., Ed., Salam, S., Bitar, N., Isaac, A., and W. Henderickx, "Provider Backbone Bridging Combined with Ethernet VPN (PBB-EVPN)", RFC 7623, DOI 10.17487/RFC7623, September 2015, <<https://www.rfc-editor.org/info/rfc7623>>.
- [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>>.
- [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>>.
- [RFC8077] Martini, L., Ed. and G. Heron, Ed., "Pseudowire Setup and Maintenance Using the Label Distribution Protocol (LDP)", STD 84, RFC 8077, DOI 10.17487/RFC8077, February 2017, <<https://www.rfc-editor.org/info/rfc8077>>.
- [RFC8214] Boutros, S., Sajassi, A., Salam, S., Drake, J., and J. Rabadan, "Virtual Private Wire Service Support in Ethernet VPN", RFC 8214, DOI 10.17487/RFC8214, August 2017, <<https://www.rfc-editor.org/info/rfc8214>>.
- [RFC8294] Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger, "Common YANG Data Types for the Routing Area", RFC 8294, DOI 10.17487/RFC8294, December 2017, <<https://www.rfc-editor.org/info/rfc8294>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8365] Sajassi, A., Ed., Drake, J., Ed., Bitar, N., Shekhar, R., Uttaro, J., and W. Henderickx, "A Network Virtualization Overlay Solution Using Ethernet VPN (EVPN)", RFC 8365, DOI 10.17487/RFC8365, March 2018, <<https://www.rfc-editor.org/info/rfc8365>>.

- [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>>.
- [RFC9181] Barguil, S., Gonzalez de Dios, O., Ed., Boucadair, M., Ed., and Q. Wu, "A Common YANG Data Model for Layer 2 and Layer 3 VPNs", RFC 9181, DOI 10.17487/RFC9181, February 2022, <<https://www.rfc-editor.org/info/rfc9181>>.

11.2. Informative References

- [I-D.ietf-bess-evpn-pref-df]
Rabadan, J., Sathappan, S., Przygienda, T., Lin, W., Drake, J., Sajassi, A., and S. Mohanty, "Preference-based EVPN DF Election", Work in Progress, Internet-Draft, draft-ietf-bess-evpn-pref-df-08, 23 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-bess-evpn-pref-df-08.txt>>.
- [I-D.ietf-bess-evpn-yang]
Brissette, P., Shah, H., Hussain, I., Tiruveedhula, K., and J. Rabadan, "Yang Data Model for EVPN", Work in Progress, Internet-Draft, draft-ietf-bess-evpn-yang-07, 11 March 2019, <<https://www.ietf.org/archive/id/draft-ietf-bess-evpn-yang-07.txt>>.
- [I-D.ietf-idr-bgp-model]
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", Work in Progress, Internet-Draft, draft-ietf-idr-bgp-model-13, 6 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-idr-bgp-model-13.txt>>.
- [I-D.ietf-opsawg-sap]
Boucadair, M., Dios, O. G. D., Barguil, S., Wu, Q., and V. Lopez, "A Network YANG Model for Service Attachment Points (SAPs)", Work in Progress, Internet-Draft, draft-ietf-opsawg-sap-04, 11 April 2022, <<https://www.ietf.org/archive/id/draft-ietf-opsawg-sap-04.txt>>.

[I-D.ietf-teas-enhanced-vpn]

Dong, J., Bryant, S., Li, Z., Miyasaka, T., and Y. Lee, "A Framework for Enhanced Virtual Private Network (VPN+) Services", Work in Progress, Internet-Draft, draft-ietf-teas-enhanced-vpn-10, 6 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-teas-enhanced-vpn-10.txt>>.

[I-D.ietf-teas-ietf-network-slices]

Farrel, A., Drake, J., Rokui, R., Homma, S., Makhijani, K., Contreras, L. M., and J. Tantsura, "Framework for IETF Network Slices", Work in Progress, Internet-Draft, draft-ietf-teas-ietf-network-slices-10, 27 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-teas-ietf-network-slices-10.txt>>.

[I-D.ietf-teas-te-service-mapping-yang]

Lee, Y., Dhody, D., Fioccola, G., Wu, Q., Ceccarelli, D., and J. Tantsura, "Traffic Engineering (TE) and Service Mapping YANG Model", Work in Progress, Internet-Draft, draft-ietf-teas-te-service-mapping-yang-10, 7 March 2022, <<https://www.ietf.org/archive/id/draft-ietf-teas-te-service-mapping-yang-10.txt>>.

[IEEE-802-1ah]

IEEE, "IEEE Standard for Local and metropolitan area networks -- Virtual Bridged Local Area Networks Amendment 7: Provider Backbone Bridges", IEEE Std 801.3AH-2008, 2008, <https://standards.ieee.org/standard/802_1ah-2008.html>.

[IEEE-802-3ah]

IEEE, "802.3ah - 2004 - IEEE Standard for Information technology-- Local and metropolitan area networks-- Part 3: CSMA/CD Access Method and Physical Layer Specifications Amendment: Media Access Control Parameters, Physical Layers, and Management Parameters for Subscriber Access Networks", IEEE Std 802.3AH-2004, 2004, <DOI 10.1109/IEEESTD.2004.94617>.

[IEEE802.1AX]

"Link Aggregation", IEEE Std 802.1AX-2020, 2020.

[IEEE802.1Q]

"Bridges and Bridged Networks", IEEE Std 802.1Q-2018, 6 July 2018, <<https://ieeexplore.ieee.org/document/8403927>>.

- [MFA] "The Use of Virtual Trunks for ATM/MPLS Control Plane Interworking Specification", MFA Forum 9.0.0 , February 2006.
- [PYANG] "pyang", November 2020,
<<https://github.com/mbj4668/pyang>>.
- [RFC2507] Degermark, M., Nordgren, B., and S. Pink, "IP Header Compression", RFC 2507, DOI 10.17487/RFC2507, February 1999, <<https://www.rfc-editor.org/info/rfc2507>>.
- [RFC2508] Casner, S. and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", RFC 2508, DOI 10.17487/RFC2508, February 1999, <<https://www.rfc-editor.org/info/rfc2508>>.
- [RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack Encoding", RFC 3032, DOI 10.17487/RFC3032, January 2001, <<https://www.rfc-editor.org/info/rfc3032>>.
- [RFC3545] Koren, T., Casner, S., Geevarghese, J., Thompson, B., and P. Ruddy, "Enhanced Compressed RTP (CRTP) for Links with High Delay, Packet Loss and Reordering", RFC 3545, DOI 10.17487/RFC3545, July 2003, <<https://www.rfc-editor.org/info/rfc3545>>.
- [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>>.
- [RFC4446] Martini, L., "IANA Allocations for Pseudowire Edge to Edge Emulation (PWE3)", BCP 116, RFC 4446, DOI 10.17487/RFC4446, April 2006, <<https://www.rfc-editor.org/info/rfc4446>>.
- [RFC4448] Martini, L., Ed., Rosen, E., El-Aawar, N., and G. Heron, "Encapsulation Methods for Transport of Ethernet over MPLS Networks", RFC 4448, DOI 10.17487/RFC4448, April 2006, <<https://www.rfc-editor.org/info/rfc4448>>.

- [RFC4553] Vainshtein, A., Ed. and YJ. Stein, Ed., "Structure-Agnostic Time Division Multiplexing (TDM) over Packet (SAToP)", RFC 4553, DOI 10.17487/RFC4553, June 2006, <<https://www.rfc-editor.org/info/rfc4553>>.
- [RFC4618] Martini, L., Rosen, E., Heron, G., and A. Malis, "Encapsulation Methods for Transport of PPP/High-Level Data Link Control (HDLC) over MPLS Networks", RFC 4618, DOI 10.17487/RFC4618, September 2006, <<https://www.rfc-editor.org/info/rfc4618>>.
- [RFC4619] Martini, L., Ed., Kawa, C., Ed., and A. Malis, Ed., "Encapsulation Methods for Transport of Frame Relay over Multiprotocol Label Switching (MPLS) Networks", RFC 4619, DOI 10.17487/RFC4619, September 2006, <<https://www.rfc-editor.org/info/rfc4619>>.
- [RFC4664] Andersson, L., Ed. and E. Rosen, Ed., "Framework for Layer 2 Virtual Private Networks (L2VPNs)", RFC 4664, DOI 10.17487/RFC4664, September 2006, <<https://www.rfc-editor.org/info/rfc4664>>.
- [RFC4717] Martini, L., Jayakumar, J., Bocci, M., El-Aawar, N., Brayley, J., and G. Koleyni, "Encapsulation Methods for Transport of Asynchronous Transfer Mode (ATM) over MPLS Networks", RFC 4717, DOI 10.17487/RFC4717, December 2006, <<https://www.rfc-editor.org/info/rfc4717>>.
- [RFC4816] Malis, A., Martini, L., Brayley, J., and T. Walsh, "Pseudowire Emulation Edge-to-Edge (PWE3) Asynchronous Transfer Mode (ATM) Transparent Cell Transport Service", RFC 4816, DOI 10.17487/RFC4816, February 2007, <<https://www.rfc-editor.org/info/rfc4816>>.
- [RFC4842] Malis, A., Pate, P., Cohen, R., Ed., and D. Zelig, "Synchronous Optical Network/Synchronous Digital Hierarchy (SONET/SDH) Circuit Emulation over Packet (CEP)", RFC 4842, DOI 10.17487/RFC4842, April 2007, <<https://www.rfc-editor.org/info/rfc4842>>.
- [RFC4863] Martini, L. and G. Swallow, "Wildcard Pseudowire Type", RFC 4863, DOI 10.17487/RFC4863, May 2007, <<https://www.rfc-editor.org/info/rfc4863>>.
- [RFC4901] Ash, J., Ed., Hand, J., Ed., and A. Malis, Ed., "Protocol Extensions for Header Compression over MPLS", RFC 4901, DOI 10.17487/RFC4901, June 2007, <<https://www.rfc-editor.org/info/rfc4901>>.

- [RFC5086] Vainshtein, A., Ed., Sasson, I., Metz, E., Frost, T., and P. Pate, "Structure-Aware Time Division Multiplexed (TDM) Circuit Emulation Service over Packet Switched Network (CESoPSN)", RFC 5086, DOI 10.17487/RFC5086, December 2007, <<https://www.rfc-editor.org/info/rfc5086>>.
- [RFC5087] Stein, Y(J)., Shashoua, R., Insler, R., and M. Anavi, "Time Division Multiplexing over IP (TDMoIP)", RFC 5087, DOI 10.17487/RFC5087, December 2007, <<https://www.rfc-editor.org/info/rfc5087>>.
- [RFC5143] Malis, A., Brayley, J., Shirron, J., Martini, L., and S. Vogelsang, "Synchronous Optical Network/Synchronous Digital Hierarchy (SONET/SDH) Circuit Emulation Service over MPLS (CEM) Encapsulation", RFC 5143, DOI 10.17487/RFC5143, February 2008, <<https://www.rfc-editor.org/info/rfc5143>>.
- [RFC5795] Sandlund, K., Pelletier, G., and L-E. Jonsson, "The RObust Header Compression (ROHC) Framework", RFC 5795, DOI 10.17487/RFC5795, March 2010, <<https://www.rfc-editor.org/info/rfc5795>>.
- [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>>.
- [RFC6307] Black, D., Ed., Dunbar, L., Ed., Roth, M., and R. Solomon, "Encapsulation Methods for Transport of Fibre Channel Traffic over MPLS Networks", RFC 6307, DOI 10.17487/RFC6307, April 2012, <<https://www.rfc-editor.org/info/rfc6307>>.
- [RFC6624] Kompella, K., Kothari, B., and R. Cherukuri, "Layer 2 Virtual Private Networks Using BGP for Auto-Discovery and Signaling", RFC 6624, DOI 10.17487/RFC6624, May 2012, <<https://www.rfc-editor.org/info/rfc6624>>.
- [RFC7209] Sajassi, A., Aggarwal, R., Uttaro, J., Bitar, N., Henderickx, W., and A. Isaac, "Requirements for Ethernet VPN (EVPN)", RFC 7209, DOI 10.17487/RFC7209, May 2014, <<https://www.rfc-editor.org/info/rfc7209>>.
- [RFC7267] Martini, L., Ed., Bocci, M., Ed., and F. Balus, Ed., "Dynamic Placement of Multi-Segment Pseudowires", RFC 7267, DOI 10.17487/RFC7267, June 2014, <<https://www.rfc-editor.org/info/rfc7267>>.

- [RFC7297] Boucadair, M., Jacquenet, C., and N. Wang, "IP Connectivity Provisioning Profile (CPP)", RFC 7297, DOI 10.17487/RFC7297, July 2014, <<https://www.rfc-editor.org/info/rfc7297>>.
- [RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.
- [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>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [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>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [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>>.
- [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>>.
- [RFC8584] Rabadan, J., Ed., Mohanty, S., Ed., Sajassi, A., Drake, J., Nagaraj, K., and S. Sathappan, "Framework for Ethernet VPN Designated Forwarder Election Extensibility", RFC 8584, DOI 10.17487/RFC8584, April 2019, <<https://www.rfc-editor.org/info/rfc8584>>.

- [RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <<https://www.rfc-editor.org/info/rfc8792>>.
- [RFC8960] Saad, T., Raza, K., Gandhi, R., Liu, X., and V. Beeram, "A YANG Data Model for MPLS Base", RFC 8960, DOI 10.17487/RFC8960, December 2020, <<https://www.rfc-editor.org/info/rfc8960>>.
- [RFC8969] Wu, Q., Ed., Boucadair, M., Ed., Lopez, D., Xie, C., and L. Geng, "A Framework for Automating Service and Network Management with YANG", RFC 8969, DOI 10.17487/RFC8969, January 2021, <<https://www.rfc-editor.org/info/rfc8969>>.

Appendix A. Examples

This section includes a non-exhaustive list of examples to illustrate the use of the L2NM.

In the following subsections, only the content of the message bodies is shown using JSON notations [RFC7951].

The examples use the folding defined in [RFC8792] for long lines.

A.1. BGP-based VPLS

This section provides an example to illustrate how the L2NM can be used to manage BGP-based VPLS. We consider the sample VPLS service delivered using the architecture depicted in Figure 23. In accordance with [RFC4761], we assume that a full mesh is established between all PEs. The details about such full mesh are not detailed here.

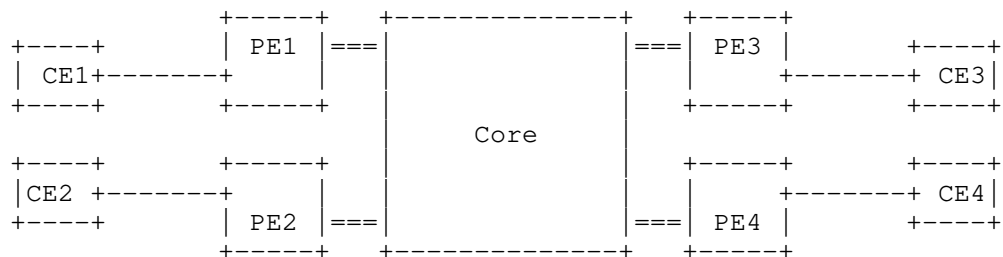


Figure 23: An Example of VPLS

Figure 24 show an example of a message body used to configure a VPLS instance using the L2NM. In this example, BGP is used for both auto-discovery and signaling. The 'signaling-type' data node is set to 'vpn-common:bgp-signaling'.

===== NOTE: '\ ' line wrapping per RFC 8792 =====

```
{
  "ietf-l2vpn-ntw:l2vpn-ntw": {
    "vpn-services": {
      "vpn-service": [
        {
          "vpn-id": "vpls7714825356",
          "vpn-description": "Sample BGP-based VPLS",
          "customer-name": "customer-7714825356",
          "vpn-type": "ietf-vpn-common:vpls",
          "bgp-ad-enabled": true,
          "signaling-type": "ietf-vpn-common:bgp-signaling",
          "global-parameters-profiles": {
            "global-parameters-profile": [
              {
                "profile-id": "simple-profile",
                "local-autonomous-system": 65535,
                "svc-mtu": 1518,
                "rd-suffix": 1,
                "vpn-target": [
                  {
                    "id": 1,
                    "route-targets": [
                      {
                        "route-target": "0:65535:1"
                      }
                    ],
                    "route-target-type": "both"
                  }
                ]
              }
            ]
          }
        },
        {
          "vpn-node-id": "pe1",
          "ne-id": "198.51.100.1",
          "active-global-parameters-profiles": {
            "global-parameters-profile": [
              {
                "profile-id": "simple-profile"
              }
            ]
          }
        }
      ]
    }
  }
}
```

```

    }
  ]
},
"bgp-auto-discovery": {
  "vpn-id": "1"
},
"signaling-option": {
  "pw-encapsulation-type": "iana-bgp-l2-encaps:ethernet\
-tagged-mode",
  "vpls-instance": {
    "vpls-edge-id": 1,
    "vpls-edge-id-range": 100
  }
},
"vpn-network-accesses": {
  "vpn-network-access": [
    {
      "id": "1/1/1.1",
      "interface-id": "1/1/1",
      "description": "Interface to CE1",
      "active-vpn-node-profile": "simple-profile",
      "status": {
        "admin-status": {
          "status": "ietf-vpn-common:admin-up"
        }
      },
      "connection": {
        "encapsulation": {
          "encap-type": "ietf-vpn-common:dot1q",
          "dot1q": {
            "cvlan-id": 1
          }
        }
      }
    }
  ]
}
},
{
  "vpn-node-id": "pe2",
  "ne-id": "198.51.100.2",
  "active-global-parameters-profiles": {
    "global-parameters-profile": [
      {
        "profile-id": "simple-profile"
      }
    ]
  }
},

```

```

    "bgp-auto-discovery": {
      "vpn-id": "1"
    },
    "signaling-option": {
      "pw-encapsulation-type": "iana-bgp-l2-encaps:ethernet\
-tagged-mode",
      "vpls-instance": {
        "vpls-edge-id": 2,
        "vpls-edge-id-range": 100
      }
    },
    "vpn-network-accesses": {
      "vpn-network-access": [
        {
          "id": "1/1/1.1",
          "interface-id": "1/1/1",
          "description": "Interface to CE2",
          "active-vpn-node-profile": "simple-profile",
          "status": {
            "admin-status": {
              "status": "ietf-vpn-common:admin-up"
            }
          },
          "connection": {
            "encapsulation": {
              "encap-type": "ietf-vpn-common:dot1q",
              "dot1q": {
                "cvlan-id": 1
              }
            }
          }
        }
      ]
    }
  },
  {
    "vpn-node-id": "pe3",
    "ne-id": "198.51.100.3",
    "active-global-parameters-profiles": {
      "global-parameters-profile": [
        {
          "profile-id": "simple-profile"
        }
      ]
    }
  },
  "bgp-auto-discovery": {
    "vpn-id": "1"
  },
},

```

```
    "signaling-option": {
      "pw-encapsulation-type": "iana-bgp-l2-encaps:ethernet\
-tagged-mode",
      "vpls-instance": {
        "vpls-edge-id": 3,
        "vpls-edge-id-range": 100
      }
    },
    "vpn-network-accesses": {
      "vpn-network-access": [
        {
          "id": "1/1/1.1",
          "interface-id": "1/1/1",
          "description": "Interface to CE3",
          "active-vpn-node-profile": "simple-profile",
          "status": {
            "admin-status": {
              "status": "ietf-vpn-common:admin-up"
            }
          },
          "connection": {
            "encapsulation": {
              "encap-type": "ietf-vpn-common:dot1q",
              "dot1q": {
                "cvlan-id": 1
              }
            }
          }
        }
      ]
    }
  },
  {
    "vpn-node-id": "pe4",
    "ne-id": "198.51.100.4",
    "active-global-parameters-profiles": {
      "global-parameters-profile": [
        {
          "profile-id": "simple-profile"
        }
      ]
    }
  },
  "bgp-auto-discovery": {
    "vpn-id": "1"
  },
  "signaling-option": {
    "pw-encapsulation-type": "iana-bgp-l2-encaps:ethernet\
-tagged-mode",
```

```

        "vpls-instance": {
            "vpls-edge-id": 4,
            "vpls-edge-id-range": 100
        },
        "vpn-network-accesses": {
            "vpn-network-access": [
                {
                    "id": "1/1/1.1",
                    "interface-id": "1/1/1",
                    "description": "Interface to CE4",
                    "active-vpn-node-profile": "simple-profile",
                    "status": {
                        "admin-status": {
                            "status": "ietf-vpn-common:admin-up"
                        }
                    },
                    "connection": {
                        "encapsulation": {
                            "encap-type": "ietf-vpn-common:dot1q",
                            "dot1q": {
                                "cvlan-id": 1
                            }
                        }
                    }
                }
            ]
        }
    ]
}

```

Figure 24: Example of L2NM Message Body to Configure a BGP-based VPLS

A.2. BGP-based VPWS with LDP Signaling

Let's consider the simple architecture depicted in Figure 25 to offer a VPWS between CE1 and CE2. The service uses BGP for auto-discovery and LDP for signaling.

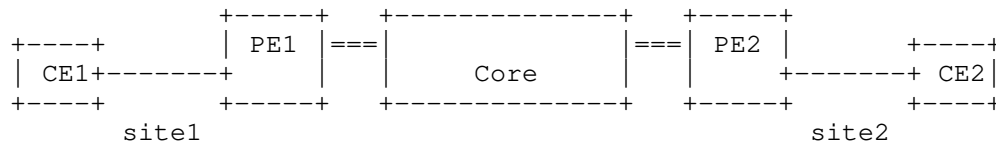


Figure 25: An Example of VPLS

```

{
  "ietf-l2vpn-ntw:l2vpn-ntw": {
    "vpn-services": {
      "vpn-service": [
        {
          "vpn-id": "vpws12345",
          "vpn-description": "Sample VPWS",
          "customer-name": "customer-12345",
          "vpn-type": "ietf-vpn-common:vpws",
          "bgp-ad-enabled": true,
          "signaling-type": "ietf-vpn-common:ldp-signaling",
          "global-parameters-profiles": {
            "global-parameters-profile": [
              {
                "profile-id": "simple-profile",
                "local-autonomous-system": 65550,
                "rd-auto": {
                  "auto": [
                    null
                  ]
                },
                "vpn-target": [
                  {
                    "id": 1,
                    "route-targets": [
                      {
                        "route-target": "0:65535:1"
                      }
                    ],
                    "route-target-type": "both"
                  }
                ]
              }
            ]
          },
          "vpn-nodes": {
            "vpn-node": [
              {
                "vpn-node-id": "pe1",
                "ne-id": "2001:db8:100::1",

```

```
"active-global-parameters-profiles": {
  "global-parameters-profile": [
    {
      "profile-id": "simple-profile"
    }
  ]
},
"bgp-auto-discovery": {
  "vpn-id": "587"
},
"signaling-option": {
  "advertise-mtu": true,
  "ldp-or-l2tp": {
    "saii": 1,
    "remote-targets": [
      {
        "taii": 2
      }
    ]
  },
  "ldp-pw-type": "ethernet"
},
"vpn-network-accesses": {
  "vpn-network-access": [
    {
      "id": "1/1/1.1",
      "interface-id": "1/1/1",
      "description": "Interface to CE1",
      "active-vpn-node-profile": "simple-profile",
      "status": {
        "admin-status": {
          "status": "ietf-vpn-common:admin-up"
        }
      }
    }
  ]
},
{
  "vpn-node-id": "pe2",
  "ne-id": "2001:db8:200::1",
  "active-global-parameters-profiles": {
    "global-parameters-profile": [
      {
        "profile-id": "simple-profile"
      }
    ]
  }
},
```

```

    "bgp-auto-discovery": {
      "vpn-id": "587"
    },
    "signaling-option": {
      "advertise-mtu": true,
      "ldp-or-l2tp": {
        "saii": 2,
        "remote-targets": [
          {
            "taii": 1
          }
        ],
        "ldp-pw-type": "ethernet"
      }
    },
    "vpn-network-accesses": {
      "vpn-network-access": [
        {
          "id": "5/1/1.1",
          "interface-id": "5/1/1",
          "description": "Interface to CE2",
          "active-vpn-node-profile": "simple-profile",
          "status": {
            "admin-status": {
              "status": "ietf-vpn-common:admin-up"
            }
          }
        }
      ]
    }
  ]
}

```

Figure 26: Example of L2NM Message Body to Configure a BGP-based VPWS with LDP Signaling

A.3. LDP-based VPLS

This section provides an example to illustrate how the L2NM can be used to manage a VPLS with LDP signaling. The connectivity between the CE and the PE is direct using Dot1q encapsulation [IEEE802.1Q]. We consider the sample service delivered using the architecture depicted in Figure 27.

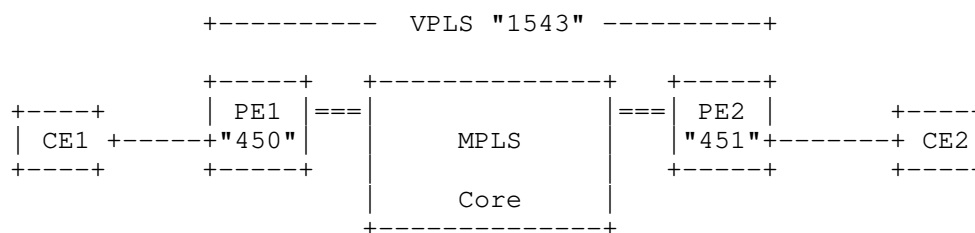


Figure 27: An Example of VPLS topology

Figure 28 shows how the L2NM is used to instruct both PE1 and PE2 to use the targeted LDP session between them to establish the VPLS "1543" between the ends. A single VPN service is created for this purpose. Additionally, two VPN Nodes and each with a corresponding VPN network access is also created.

===== NOTE: '\ ' line wrapping per RFC 8792 =====

```
{
  "ietf-l2vpn-ntw:l2vpn-ntw": {
    "vpn-services": {
      "vpn-service": [
        {
          "vpn-id": "450",
          "vpn-name": "CORPO-EXAMPLE",
          "vpn-description": "SEDE_CENTRO_450",
          "customer-name": "EXAMPLE",
          "vpn-type": "ietf-vpn-common:vpls",
          "vpn-service-topology": "ietf-vpn-common:hub-spoke",
          "bgp-ad-enabled": false,
          "signaling-type": "ietf-vpn-common:ldp-signaling",
          "global-parameters-profiles": {
            "global-parameters-profile": [
              {
                "profile-id": "simple-profile",
                "ce-vlan-preservation": true,
                "ce-vlan-cos-preservation": true
              }
            ]
          }
        }
      ]
    }
  }
}
```

```
    ]
  },
  "vpn-nodes": {
    "vpn-node": [
      {
        "vpn-node-id": "450",
        "description": "SEDE_CENTRO_450",
        "ne-id": "2001:db8:5::1",
        "role": "ietf-vpn-common:hub-role",
        "status": {
          "admin-status": {
            "status": "ietf-vpn-common:admin-up"
          }
        },
        "active-global-parameters-profiles": {
          "global-parameters-profile": [
            {
              "profile-id": "simple-profile"
            }
          ]
        },
        "signaling-option": {
          "ldp-or-l2tp": {
            "t-ldp-pw-type": "vpls-type",
            "pw-peer-list": [
              {
                "peer-addr": "2001:db8:50::1",
                "vc-id": "1543"
              }
            ]
          }
        }
      ]
    },
    "vpn-network-accesses": {
      "vpn-network-access": [
        {
          "id": "4508671287",
          "description": "VPN_450_SNA",
          "interface-id": "gigabithethernet0/0/1",
          "status": {
            "admin-status": {
              "status": "ietf-vpn-common:admin-up"
            }
          }
        },
        "connection": {
          "l2-termination-point": "550",
          "encapsulation": {
            "encap-type": "ietf-vpn-common:dot1q",
            "dot1q": {
```

```

        "tag-type": "ietf-vpn-common:c-vlan",
        "cvlan-id": 550
    }
}
},
"service": {
    "mtu": 1550,
    "svc-pe-to-ce-bandwidth": {
        "pe-to-ce-bandwidth": [
            {
                "bw-type": "ietf-vpn-common:bw-per-port",
                "cir": "20480000"
            }
        ]
    },
    "svc-ce-to-pe-bandwidth": {
        "ce-to-pe-bandwidth": [
            {
                "bw-type": "ietf-vpn-common:bw-per-port",
                "cir": "20480000"
            }
        ]
    },
    "qos": {
        "qos-profile": {
            "qos-profile": [
                {
                    "profile": "QoS_Profile_A",
                    "direction": "ietf-vpn-common:both"
                }
            ]
        }
    }
}
}
}
}
}
}
}
{
    "vpn-node-id": "451",
    "description": "SEDE_CHAPINERO_451",
    "ne-id": "2001:db8:50::1",
    "role": "ietf-vpn-common:spoke-role",
    "status": {
        "admin-status": {
            "status": "ietf-vpn-common:admin-up"
        }
    }
},

```

```
"active-global-parameters-profiles": {
  "global-parameters-profile": [
    {
      "profile-id": "simple-profile"
    }
  ]
},
"signaling-option": {
  "ldp-or-l2tp": {
    "t-ldp-pw-type": "vpls-type",
    "pw-peer-list": [
      {
        "peer-addr": "2001:db8:5::1",
        "vc-id": "1543"
      }
    ]
  }
},
"vpn-network-accesses": {
  "vpn-network-access": [
    {
      "id": "4508671288",
      "description": "VPN_450_SNA",
      "interface-id": "gigabithethernet0/0/1",
      "status": {
        "admin-status": {
          "status": "ietf-vpn-common:admin-up"
        }
      },
      "connection": {
        "l2-termination-point": "550",
        "encapsulation": {
          "encap-type": "ietf-vpn-common:dot1q",
          "dot1q": {
            "tag-type": "ietf-vpn-common:c-vlan",
            "cvlan-id": 550
          }
        }
      },
      "service": {
        "mtu": 1550,
        "svc-pe-to-ce-bandwidth": {
          "pe-to-ce-bandwidth": [
            {
              "bw-type": "ietf-vpn-common:bw-per-port",
              "cir": "20480000"
            }
          ]
        }
      }
    }
  ]
}
```

```

    },
    "svc-ce-to-pe-bandwidth": {
      "ce-to-pe-bandwidth": [
        {
          "bw-type": "ietf-vpn-common:bw-per-port",
          "cir": "20480000"
        }
      ]
    },
    "qos": {
      "qos-profile": {
        "qos-profile": [
          {
            "profile": "QoS_Profile_A",
            "direction": "ietf-vpn-common:both"
          }
        ]
      }
    }
  }
}

```

Figure 28: Example of L2NM Message Body for LDP-based VPLS

A.4. VPWS-EVPN Service Instance

Figure 29 depicts a sample architecture to offer VPWS-EVPN service between CE1 and CE2. Both CEs are multi-homed. BGP sessions are maintained between these PEs as per [RFC8214]. In this EVPN instance, an All-Active redundancy mode is used.

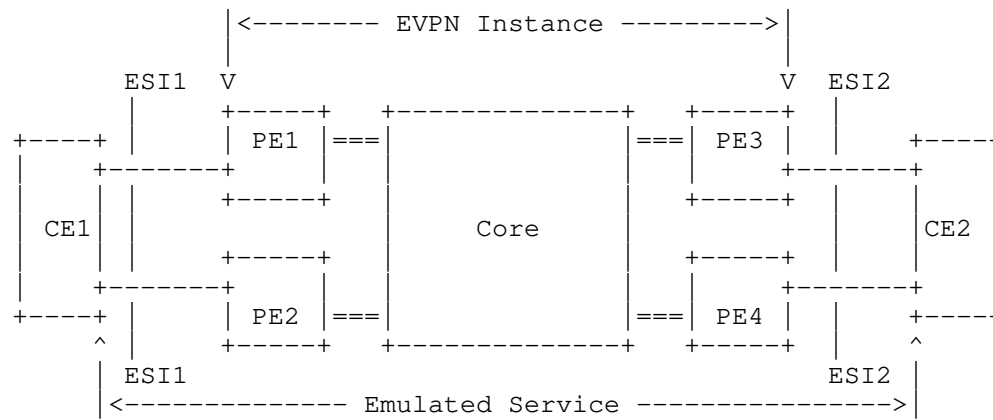


Figure 29: An Example of VPWS-EVPN

Let's first suppose that the following ES was created (Figure 30).

===== NOTE: '\ ' line wrapping per RFC 8792 =====

```
{
  "ietf-ethernet-segment:ethernet-segments": {
    "ethernet-segment": [
      {
        "name": "esi1",
        "ethernet-segment-identifier": "00:11:11:11:11:11:\
11:11:11",
        "esi-redundancy-mode": "all-active"
      },
      {
        "name": "esi2",
        "ethernet-segment-identifier": "00:22:22:22:22:22:\
22:22:22",
        "esi-redundancy-mode": "all-active"
      }
    ]
  }
}
```

Figure 30: Example of L2NM Message Body to Configure an Ethernet Segment

Figure 29 shows a simplified configuration to illustrate the use of the L2NM to configured VPWS-EVPN instance.

```
{
  "ietf-l2vpn-ntw:l2vpn-ntw": {
    "vpn-services": {
      "vpn-service": [
        {
          "vpn-id": "vpws15432855",
          "vpn-description": "Sample VPWS-EVPN",
          "customer-name": "customer_15432855",
          "vpn-type": "ietf-vpn-common:vpws-evpn",
          "bgp-ad-enabled": true,
          "signaling-type": "ietf-vpn-common:bgp-signaling",
          "global-parameters-profiles": {
            "global-parameters-profile": [
              {
                "profile-id": "simple-profile",
                "local-autonomous-system": 65535,
                "rd-suffix": 1,
                "vpn-target": [
                  {
                    "id": 1,
                    "route-targets": [
                      {
                        "route-target": "0:65535:1"
                      }
                    ],
                    "route-target-type": "both"
                  }
                ]
              }
            ]
          }
        }
      ],
      "vpn-nodes": {
        "vpn-node": [
          {
            "vpn-node-id": "pe1",
            "ne-id": "198.51.100.1",
            "active-global-parameters-profiles": {
              "global-parameters-profile": [
                {
                  "profile-id": "simple-profile"
                }
              ]
            },
            "vpn-network-accesses": {
              "vpn-network-access": [
                {
                  "id": "1/1/1.1",
                  "interface-id": "1/1/1",

```

```

      "description": "Interface to CE1",
      "active-vpn-node-profile": "simple-profile",
      "status": {
        "admin-status": {
          "status": "ietf-vpn-common:admin-up"
        }
      },
      "connection": {
        "encapsulation": {
          "encap-type": "ietf-vpn-common:dot1q",
          "dot1q": {
            "cvlan-id": 1
          }
        }
      },
      "vpws-service-instance": {
        "local-vpws-service-instance": 1111,
        "remote-vpws-service-instance": 1112
      },
      "group": [
        {
          "group-id": "gr1",
          "ethernet-segment-identifier": "es1"
        }
      ]
    }
  ],
},
{
  "vpn-node-id": "pe2",
  "ne-id": "198.51.100.2",
  "active-global-parameters-profiles": {
    "global-parameters-profile": [
      {
        "profile-id": "simple-profile"
      }
    ]
  },
  "vpn-network-accesses": {
    "vpn-network-access": [
      {
        "id": "1/1/1.1",
        "interface-id": "1/1/1",
        "description": "Interface to CE1",
        "active-vpn-node-profile": "simple-profile",
        "status": {
          "admin-status": {

```

```
        "status": "ietf-vpn-common:admin-up"
      }
    },
    "connection": {
      "encapsulation": {
        "encap-type": "ietf-vpn-common:dot1q",
        "dot1q": {
          "cvlan-id": 1
        }
      }
    },
    "vpws-service-instance": {
      "local-vpws-service-instance": 1111,
      "remote-vpws-service-instance": 1112
    },
    "group": [
      {
        "group-id": "gr1",
        "ethernet-segment-identifier": "es1"
      }
    ]
  }
}
},
{
  "vpn-node-id": "pe3",
  "ne-id": "198.51.100.3",
  "active-global-parameters-profiles": {
    "global-parameters-profile": [
      {
        "profile-id": "simple-profile"
      }
    ]
  },
  "vpn-network-accesses": {
    "vpn-network-access": [
      {
        "id": "1/1/1.1",
        "interface-id": "1/1/1",
        "description": "Interface to CE2",
        "active-vpn-node-profile": "simple-profile",
        "status": {
          "admin-status": {
            "status": "ietf-vpn-common:admin-up"
          }
        }
      },
      "connection": {
```

```
        "encapsulation": {
          "encap-type": "ietf-vpn-common:dot1q",
          "dot1q": {
            "cvlan-id": 1
          }
        },
        "vpws-service-instance": {
          "local-vpws-service-instance": 1112,
          "remote-vpws-service-instance": 1111
        },
        "group": [
          {
            "group-id": "gr1",
            "ethernet-segment-identifier": "esi2"
          }
        ]
      }
    ],
    {
      "vpn-node-id": "pe4",
      "ne-id": "198.51.100.4",
      "active-global-parameters-profiles": {
        "global-parameters-profile": [
          {
            "profile-id": "simple-profile"
          }
        ]
      },
      "vpn-network-accesses": {
        "vpn-network-access": [
          {
            "id": "1/1/1.1",
            "interface-id": "1/1/1",
            "description": "Interface to CE2",
            "active-vpn-node-profile": "simple-profile",
            "status": {
              "admin-status": {
                "status": "ietf-vpn-common:admin-up"
              }
            }
          }
        ],
        "connection": {
          "encapsulation": {
            "encap-type": "ietf-vpn-common:dot1q",
            "dot1q": {
              "cvlan-id": 1
            }
          }
        }
      }
    }
  ]
}
```

```
    }  
  },  
  "vpws-service-instance": {  
    "local-vpws-service-instance": 1112,  
    "remote-vpws-service-instance": 1111  
  },  
  "group": [  
    {  
      "group-id": "gr1",  
      "ethernet-segment-identifier": "esi2"  
    }  
  ]  
}  
]  
}  
]  
}  
]  
}  
]  
}
```

Figure 31: Example of L2NM Message Body to Configure a VPWS-EVPN Instance

A.5. Automatic ESI Assignment

This section provides an example to illustrate how the L2NM can be used to manage ESI auto-assignment. We consider the sample EVPN service delivered using the architecture depicted in Figure 32.

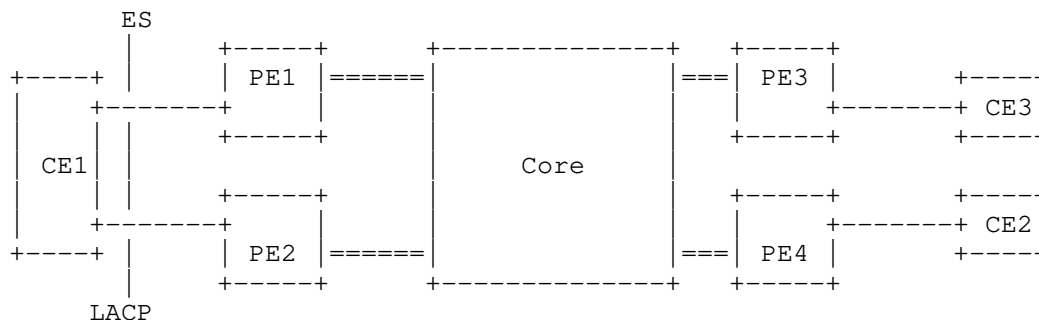


Figure 32: An Example of Automatic ESI Assignment

Figure 33 and Figure 34 show how the L2NM is used to instruct both PE1 and PE2 to auto-assign the ESI to identify the ES used with CE1. In this example, we suppose that LACP is enabled and that a Type 1 (T=0x01) is used as per Section 5 of [RFC7432]. Note that this example does not include all the details to configure the EVPN service, but focuses only on the ESI management part.

```
{
  "ietf-ethernet-segment:ethernet-segments": {
    "ethernet-segment": [
      {
        "name": "esi1",
        "esi-type": "esi-type-1-lacp",
        "esi-redundancy-mode": "all-active"
      }
    ]
  }
}
```

Figure 33: Example of L2NM Message Body to Auto-Assign Ethernet Segment Identifiers

```
{
  "ietf-l2vpn-ntw:l2vpn-ntw": {
    "ietf-l2vpn-ntw:vpn-services": {
      "vpn-service": [
        {
          "vpn-id": "auto-esi-lacp",
          "vpn-description": "Sample to illustrate auto-ESI",
          "vpn-type": "ietf-vpn-common:vpws-evpn",
          "vpn-nodes": {
            "vpn-node": [
              {
                "vpn-node-id": "pe1",
                "ne-id": "198.51.100.1",
                "vpn-network-accesses": {
                  "vpn-network-access": [
                    {
                      "id": "1/1/1.1",
                      "interface-id": "1/1/1",
                      "description": "Interface to CE1",
                      "status": {
                        "admin-status": {
                          "status": "ietf-vpn-common:admin-up"
                        }
                      }
                    }
                  ]
                }
              }
            ]
          }
        }
      ]
    }
  }
}
```

```
    "connection": {
      "lag-interface": {
        "lag-interface-id": "1",
        "lcp": {
          "lcp-state": true,
          "system-id": "11:00:11:00:11:11",
          "admin-key": 154
        }
      }
    },
    "group": [
      {
        "group-id": "gr1",
        "ethernet-segment-identifier": "es1"
      }
    ]
  }
},
{
  "vpn-node-id": "pe2",
  "ne-id": "198.51.100.2",
  "vpn-network-accesses": {
    "vpn-network-access": [
      {
        "id": "2/2/2.5",
        "interface-id": "2/2/2",
        "description": "Interface to CE1",
        "status": {
          "admin-status": {
            "status": "ietf-vpn-common:admin-up"
          }
        }
      },
      {
        "connection": {
          "lag-interface": {
            "lag-interface-id": "1",
            "lcp": {
              "lcp-state": true,
              "system-id": "11:00:11:00:11:11",
              "admin-key": 154
            }
          }
        }
      }
    ]
  },
  "group": [
    {
      "group-id": "gr1",
      "ethernet-segment-identifier": "es1"
    }
  ]
}
```

Figure 34: An Example of L2NM Message Body for ESI Auto-Assignment

The auto-assigned ESI can be retrieved using, e.g., a GET RESTCONF method. The assigned value will be then returned as shown in the 'esi-auto' data node in Figure 35.

===== NOTE: '\ ' line wrapping per RFC 8792 =====

```
{
  "ietf-ethernet-segment:ethernet-segments": {
    "ethernet-segment": [
      {
        "name": "esi1",
        "ethernet-segment-identifier": "esi-type-1-lacp",
        "esi-auto": {
          "auto-ethernet-segment-identifier": "01:11:00:11:00:11:\n11:9a:00:00"
        },
        "esi-redundancy-mode": "all-active"
      }
    ]
  }
}
```

Figure 35: An Example of L2NM Message Body to Retrieve the Assigned ESI

A.6. VPN Network Access Precedence

In reference to the example depicted in Figure 36, an L2VPN service involves two VPN network accesses to sites that belong to the same customer.

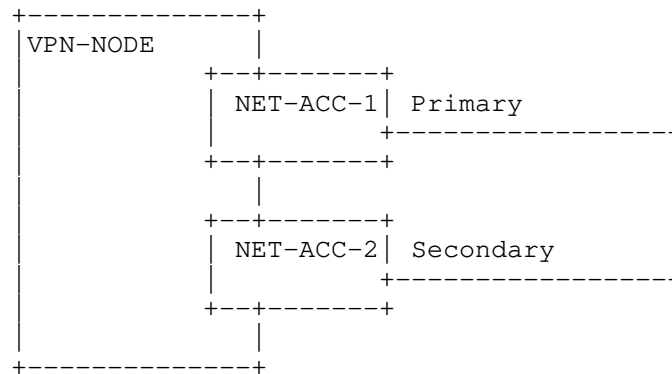


Figure 36: Example of Multiple VPN Network Accesses

In order to tag one of these VPN network accesses as "primary" and the other one as "secondary", Figure 37 shows an excerpt of the corresponding L2NM configuration. In such a configuration, both accesses are bound to the same "group-id" and the "precedence" data node set as function of the intended role of each access (primary or secondary).

```

{
  "ietf-l2vpn-ntw:l2vpn-ntw": {
    "vpn-services": {
      "vpn-service": [
        {
          "vpn-id": "Sample-Service",
          "vpn-nodes": {
            "vpn-node": [
              {
                "vpn-node-id": "VPN-NODE",
                "vpn-network-accesses": {
                  "vpn-network-access": [
                    {
                      "id": "NET-ACC-1",
                      "connection": {
                        "bearer-reference": "br1"
                      },
                      "group": [
                        {
                          "group-id": "1",
                          "precedence": "primary"
                        }
                      ]
                    },
                    {
                      "id": "NET-ACC-2",
                      "connection": {
                        "bearer-reference": "br2"
                      },
                      "group": [
                        {
                          "group-id": "1",
                          "precedence": "secondary"
                        }
                      ]
                    }
                  ]
                }
              }
            ]
          }
        }
      ]
    }
  }
}

```

Figure 37: Example of Message Body to Associate Priority Levels
with VPN Network Accesses

Acknowledgements

During the discussions of this work, helpful comments, suggestions, and reviews were received from: Sergio Belotti, Italo Busi, Miguel Cros Cecilia, Joe Clarke, Dhruv Dhody, Adrian Farrel, Roque Gagliano, Christian Jacquenet, Kireeti Kompella, Julian Lucek, Moti Morgenstern, Erez Segev, and Tom Petch. Many thanks to them.

Luay Jalil, Jichun Ma, Daniel King, and Zhang Guiyu contributed to an early version of this document.

Thanks to Yingzhen Qu and Himanshu Shah for the rtgdir reviews, Ladislav Lhotka for the yangdoctors review, Chris Lonvick for the secdir review, and Dale Worley for the gen-art review. Special thanks to Adrian Farrel for the careful Shepherd review. Thanks to Robert Wilton for the careful AD review and various suggestions to enhance the model.

A YANG module for Ethernet segments was first defined in the context of the EVPN device module [I-D.ietf-bess-evpn-yang].

This work is partially supported by the European Commission under Horizon 2020 grant agreement number 101015857 Secured autonomic traffic management for a Tera of SDN flows (Teraflow).

Contributors

Victor Lopez
Nokia
Email: victor.lopez@nokia.com

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

Raul Arco
Nokia
Email: raul.arco@nokia.com

Authors' Addresses

Mohamed Boucadair (editor)
Orange
Rennes
France
Email: mohamed.boucadair@orange.com

Oscar Gonzalez de Dios (editor)
Telefonica
Madrid
Spain
Email: oscar.gonzalezdedios@telefonica.com

Samier Barguil
Telefonica
Madrid
Spain
Email: samier.barguilgiraldo.ext@telefonica.com

Luis Angel Munoz
Vodafone
Spain
Email: luis-angel.munoz@vodafone.com

OPSAWG
Internet-Draft
Intended status: Standards Track
Expires: 11 April 2022

S. Barguil
O. Gonzalez de Dios, Ed.
Telefonica
M. Boucadair, Ed.
Orange
L. Munoz
Vodafone
A. Aguado
Nokia
8 October 2021

A Layer 3 VPN Network YANG Model
draft-ietf-opsawg-l3sm-l3nm-18

Abstract

As a complement to the Layer 3 Virtual Private Network Service YANG data Model (L3SM), used for communication between customers and service providers, this document defines an L3VPN Network YANG Model (L3NM) that can be used for the provisioning of Layer 3 Virtual Private Network (VPN) services within a service provider 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

Please update "RFC UUUU" to the RFC number to be assigned to I-D.ietf-opsawg-vpn-common.

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 11 April 2022.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. Terminology	5
3. Acronyms	6
4. L3NM Reference Architecture	7
5. Relation with other YANG Models	11
6. Sample Uses of the L3NM Data Model	12
6.1. Enterprise Layer 3 VPN Services	12
6.2. Multi-Domain Resource Management	13
6.3. Management of Multicast Services	13
7. Description of the L3NM YANG Module	13
7.1. Overall Structure of the Module	14
7.2. VPN Profiles	15
7.3. VPN Services	16
7.4. VPN Instance Profiles	20
7.5. VPN Nodes	22

7.6.	VPN Network Accesses	25
7.6.1.	Connection	28
7.6.2.	IP Connection	30
7.6.3.	CE-PE Routing Protocols	33
7.6.3.1.	Static Routing	35
7.6.3.2.	BGP	37
7.6.3.3.	OSPF	40
7.6.3.4.	IS-IS	42
7.6.3.5.	RIP	44
7.6.3.6.	VRRP	45
7.6.4.	OAM	47
7.6.5.	Security	48
7.6.6.	Services	49
7.6.6.1.	Overview	49
7.6.6.2.	QoS	50
7.7.	Multicast	55
8.	L3NM YANG Module	59
9.	Security Considerations	121
10.	IANA Considerations	122
11.	References	123
11.1.	Normative References	123
11.2.	Informative References	127
Appendix A.	L3VPN Examples	132
A.1.	4G VPN Provisioning Example	132
A.2.	Loopback Interface	137
A.3.	Overriding VPN Instance Profile Parameters	138
A.4.	Multicast VPN Provisioning Example	141
Appendix B.	Implementation Status	145
B.1.	Nokia Implementation	145
B.2.	Huawei Implementation	145
B.3.	Infinera Implementation	145
B.4.	Ribbon-ECI Implementation	145
B.5.	Juniper Implementation	146
Acknowledgements		146
Contributors		146
Authors' Addresses		147

1. Introduction

[RFC8299] defines a Layer 3 Virtual Private Network Service YANG data Model (L3SM) that can be used for communication between customers and service providers. Such a model focuses 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 to the role of a customer service model (as per [RFC8309]).

This document defines 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 and the network controller/orchestrator by allowing for more network-centric information to be included. It enables further capabilities such as resource management or serves as a multi-domain orchestration interface, where logical resources (such as route targets or route distinguishers) must be coordinated.

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

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

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

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

Section 5.1 of [RFC8969] illustrates how the L3NM can be used within the network management automation architecture.

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 an end-to-end VPN connectivity are addressed by complementary YANG modules, e.g., [I-D.evenwu-opsawg-yang-composed-vpn].

The L3NM focuses on BGP Provider Edge (PE) based Layer 3 VPNs as described in [RFC4026][RFC4110][RFC4364] and Multicast VPNs as described in [RFC6037][RFC6513].

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

2. Terminology

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.

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.

This document uses the term "network model" defined in Section 2.1 of [RFC8969].

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

This document makes use of the following terms:

Layer 3 VPN Customer Service Model (L3SM): A YANG module that describes the service requirements of an 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 an L3VPN. The service orchestrator interacts with the customer using the L3SM. The service orchestrator is responsible for the Customer Edge (CE) - 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 a 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 when both models are used in a given deployment.

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: A service provider that offers VPN-related services [RFC4176].

Service provider network: A network that is able to provide VPN-related services.

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.

3. Acronyms

The following acronyms are used in the document:

ACL	Access Control List
AS	Autonomous System
ASM	Any-Source Multicast
ASN	AS Number
BSR	Bootstrap Router
BFD	Bidirectional Forwarding Detection
BGP	Border Gateway Protocol
CE	Customer Edge

CsC	Carriers' Carriers
IGMP	Internet Group Management Protocol
L3VPN	Layer 3 Virtual Private Network
L3SM	L3VPN Service Model
L3NM	L3VPN Network Model
MLD	Multicast Listener Discovery
MSDP	Multicast Source Discovery Protocol
MVPN	Multicast VPN
NAT	Network Address Translation
OAM	Operations, Administration, and Maintenance
OSPF	Open Shortest Path First
PE	Provider Edge
PIM	Protocol Independent Multicast
QoS	Quality of Service
RD	Route Distinguisher
RP	Rendezvous Point
RT	Route Target
SA	Security Association
SSM	Source-Specific Multicast
VPN	Virtual Private Network
VRF	Virtual Routing and Forwarding

4. L3NM Reference Architecture

Figure 1 depicts the reference architecture for the L3NM. The figure is an expansion of the architecture presented in Section 5 of [RFC8299]; it decomposes the box marked "orchestration" in that section into three separate functional components: 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]. This separation relies upon a dedicated communication interface between these components and appropriate YANG modules that reflect network-related information. Such information is hidden to customers.

The intelligence for translating customer-facing information into network-centric one (and vice versa) 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".

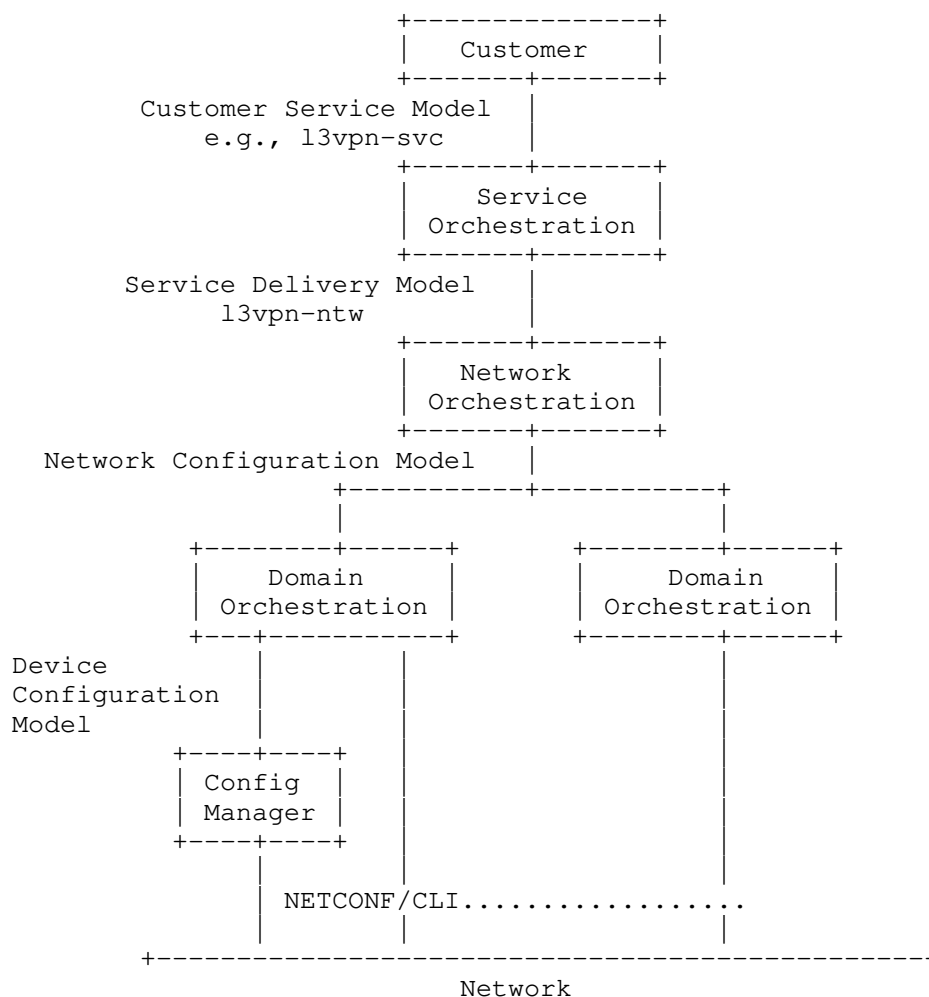


Figure 1: L3NM Reference Architecture

The customer may use a variety of means to request a service that may trigger the instantiation of an L3NM. The customer may use the L3SM or more abstract models to request a service that relies upon an L3VPN service. For example, the customer may supply an IP Connectivity Provisioning Profile (CPP) that characterizes the

requested service [RFC7297], an enhanced VPN (VPN+) service [I-D.ietf-teas-enhanced-vpn], or an IETF network slice service [I-D.ietf-teas-ietf-network-slices].

Note also that both the L3SM and the L3NM may be used in the context of the Abstraction and Control of TE Networks (ACTN) Framework [RFC8453]. Figure 2 shows the Customer Network Controller (CNC), the Multi-Domain Service Coordinator (MDSC), and the Provisioning Network Controller (PNC) components and the interfaces where L3SM/L3NM are used.

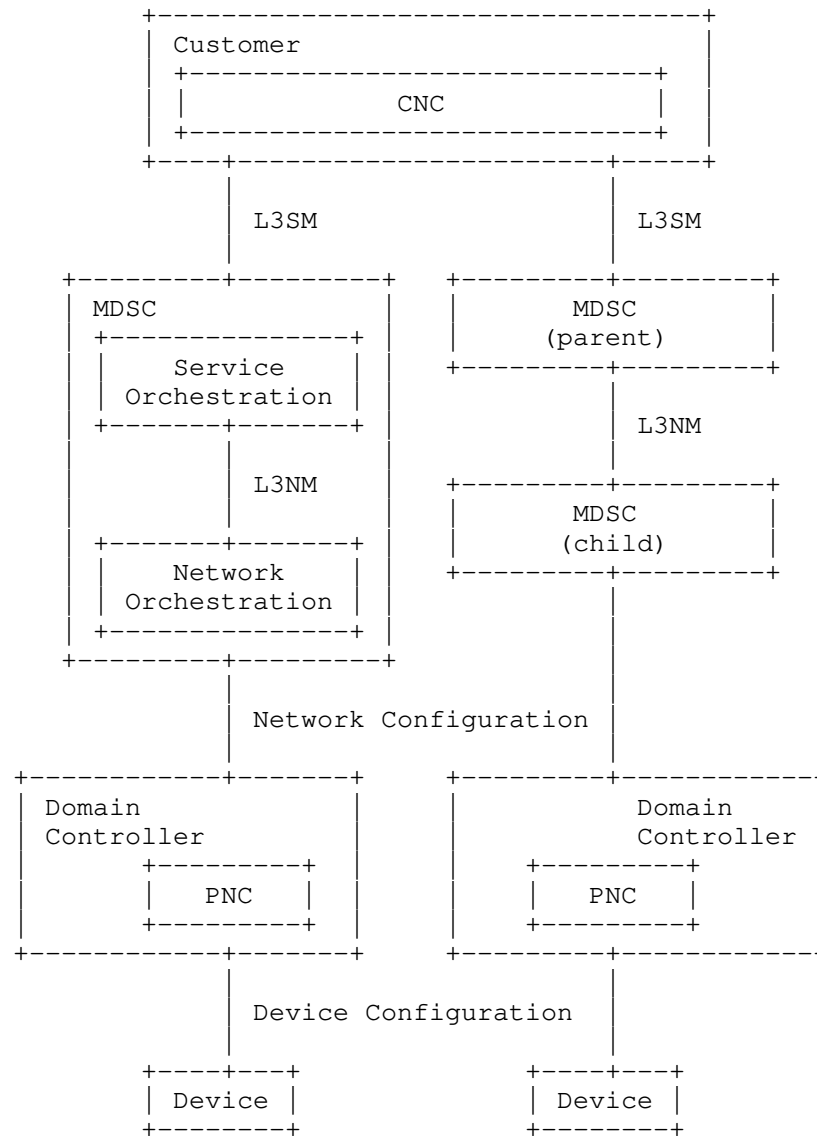


Figure 2: L3SM and L3NM in the Context of ACTN

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 groupings.

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 initial 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 is meant to manage L3VPN services within a service provider network. The module provides a network view of the service. Such a 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 (i.e., L3NM) may be mapped to an external view which is visible to customers: L3VPN Service YANG data Model (L3SM) [RFC8299].

The L3NM can be fed with inputs that are requested by customers, typically, relying upon an 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 the 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: An L3VPN involves nodes that are part of a

topology managed by the service provider network. The topology can be represented using the network topology YANG module defined in [RFC8345] or its extension such as a User-Network Interface (UNI) topology module (e.g., [I-D.ogondio-opsawg-uni-topology]).

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

This section provides a non-exhaustive list of examples to illustrate contexts where the L3NM can be used.

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 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 Software-Defined Networking (SDN) [RFC7149][RFC7426] layer, but some manual intervention will still be required.

A common function that is supported by VPNs is the addition or removal of VPN nodes. Workflows can use the L3NM in these scenarios to add or prune nodes from the network data model as required.

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, resources must be adequately managed in each domain to avoid broken configuration.

For example, route targets (RTs) shall be synchronized between PEs. When all PEs are controlled by the same management system, RT allocation can be performed by that management system. In cases where the service spans across multiple management systems, the task of allocating RTs has to be aligned across the domains, therefore, the network model must provide a way to specify RTs. In addition, route distinguishers (RDs) must also be synchronized to avoid collisions in RD allocation between separate management systems. An incorrect allocation might lead to the same RD and IP prefixes being exported by different PEs.

6.3. Management of Multicast Services

Multicast services over L3VPN can be implemented using dual PIM MVPNs (also known as, Draft Rosen model) [RFC6037] or Multiprotocol BGP (MP-BGP)-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 network. 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 PEs using the same architecture that is used for unicast VPNs.

On the other hand, [RFC6037] 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 these limitations, MBGP MVPN is the architectural model that has been taken as the base for implementing multicast service in L3VPNs. In this scenario, BGP is used to auto-discover MVPN PE members and the customer PIM signaling is sent across the provider's core through MP-BGP. The multicast traffic is transported on MPLS P2MP LSPs.

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.

The full tree diagram of the module can be generated using the "pyang" tool [PYANG]. That tree is not included here because it is too long (Section 3.3 of [RFC8340]). Instead, subtrees are provided for the reader's convenience.

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-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).

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

```
module: ietf-l3vpn-ntw
  +--rw l3vpn-ntw
    +--rw vpn-profiles
      | ...
    +--rw vpn-services
      +--rw vpn-service* [vpn-id]
        ...
      +--rw vpn-nodes
        +--rw vpn-node* [vpn-node-id]
          ...
        +--rw vpn-network-accesses
          +--rw vpn-network-access* [id]
            ...
```

Figure 3: Overall L3NM Tree Structure

Some of the data nodes are keyed by the address-family. For the sake of data representation compactness, It is RECOMMENDED to use the dual-stack address-family for data nodes that have the same value for both IPv4 and IPv6. If, for some reasons, a data node is present for both dual-stack and IPv4 (or IPv6), the value that is indicated under dual-stack takes precedence over the one that is indicated under IPv4 (or IPv6).

7.2. VPN Profiles

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

```

+--rw l3vpn-ntw
  +--rw vpn-profiles
    +--rw valid-provider-identifiers
      +--rw external-connectivity-identifier* [id]
        {external-connectivity}?
        +--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

This document does not make any assumption about the exact definition of these profiles. The exact definition of the profiles is local to each VPN service provider. The model only includes an identifier to these profiles in order to facilitate identifying and binding local policies when building a VPN service. As shown in Figure 4, the following identifiers can be included:

'external-connectivity-identifier': This identifier refers to a profile that defines the external connectivity provided to a VPN service (or a subset of VPN sites). An external connectivity may be an access to the Internet or a restricted connectivity such as access to a public/private cloud.

'encryption-profile-identifier': An encryption profile refers to a set of policies related to the encryption schemes and setup that can be applied when building and offering a VPN service.

'qos-profile-identifier': A Quality of Service (QoS) profile refers to a 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, for example, of 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) when delivering the VPN service.

7.3. VPN Services

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 (e.g., the network controller). The subtree of the 'vpn-services' is shown in Figure 5.

```

+--rw l3vpn-ntw
  +--rw vpn-profiles
  |   ...
  +--rw vpn-services
    +--rw vpn-service* [vpn-id]
      +--rw vpn-id                vpn-common:vpn-id
      +--rw vpn-name?             string
      +--rw vpn-description?      string
      +--rw customer-name?        string
      +--rw parent-service-id?    vpn-common:vpn-id
      +--rw vpn-type?             identityref
      +--rw vpn-service-topology? identityref
      +--rw status
        +--rw admin-status
        |   +--rw status?         identityref
        |   +--rw last-change?    yang:date-and-time
        +--ro oper-status
        |   +--ro status?         identityref
        |   +--ro last-change?    yang:date-and-time
      +--rw vpn-instance-profiles
      |   ...
      +--rw underlay-transport
      |   +-- (type)?
      |   |   +--:(abstract)
      |   |   |   +-- transport-instance-id? string
      |   |   +--:(protocol)
      |   |   |   +-- protocol*         identityref
      +--rw external-connectivity
      |   {external-connectivity}
      |   +--rw (profile)?
      |   |   +--:(profile)
      |   |   |   +--rw profile-name?    leafref
      +--rw vpn-nodes
      |   ...

```

Figure 5: VPN Services Subtree Structure

The description of the VPN service data nodes that are depicted in Figure 5 are as follows:

- 'vpn-id': Is an identifier that is used to uniquely identify the L3VPN service within L3NM scope.
- 'vpn-name': Associates a name with the service in order to facilitate the identification of the service.
- 'vpn-description': Includes a textual description of the service.

The internal structure of a VPN description is local to each VPN service provider.

'customer-name': Indicates the name of the customer who ordered the service.

'parent-service-id': Refers to an identifier of the parent service (e.g., L3SM, IETF network slice, VPN+) that triggered the creation of the VPN service. This identifier is used to easily correlate the (network) service as built in the network with a service order. A controller can use that correlation to enrich or populate some fields (e.g., description fields) as a function of local deployments.

'vpn-type': Indicates the VPN type. The values are taken from [I-D.ietf-opsawg-vpn-common]. For the L3NM, this is typically set to BGP/MPLS L3VPN, but other values may be defined in the future to support specific Layer 3 VPN capabilities (e.g., [I-D.ietf-bess-evpn-prefix-advertisement]).

'vpn-service-topology': Indicates the network topology for the service: hub-spoke, any-to-any, or custom. The network implementation of this attribute is defined by the correct usage of import and export profiles (Section 4.3.5 of [RFC4364]).

'status': Is used to track the service status of a given VPN service. Both operational and administrative status are maintained together with a timestamp. For example, a service can be created, but not put into effect.

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

'vpn-instance-profiles': Defines reusable parameters for the same 'vpn-service'.

More details are provided in Section 7.4.

'underlay-transport': Describes the preference for the transport

technology to carry the traffic of the VPN service. This preference is especially useful in networks with multiple domains and Network-to-Network Interface (NNI) types. The underlay transport can be expressed as an abstract transport instance (e.g., an identifier of a VPN+ instance, a virtual network identifier, or a network slice name) or as an ordered list of the actual protocols to be enabled in the network.

A rich set of protocol identifiers that can be used to refer to an underlay transport are defined in [I-D.ietf-opsawg-vpn-common].

'external-connectivity': Indicates whether/how external connectivity is provided to the VPN service. For example, a service provider may provide an external connectivity to a VPN customer (e.g., to a public cloud). Such service may involve tweaking both filtering and NAT rules (e.g., bind a Virtual Routing and Forwarding (VRF) interface with a NAT instance as discussed in Section 2.10 of [RFC8512]). These added value features may be bound to all or a subset of network accesses. Some of these added value features may be implemented in a PE or in other nodes than PEs (e.g., a P node or even a dedicated node that hosts the NAT function).

Only a pointer to a local profile that defines the external connectivity feature is supported in this document.

'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. Whether that information is included in the L3NM, e.g., to populate the various 'description' data node is implementation specific.

More details are provided in Section 7.5.

7.4. VPN Instance Profiles

VPN instance profiles are meant to factorize data nodes that are used at many levels of the model. Generic VPN instance profiles are defined at the VPN service level and then called at the VPN node and VPN network access levels. Each VPN instance profile is identified by 'profile-id'. This identifier is then referenced for one or multiple VPN nodes (Section 7.5) so that the controller can identify generic resources (e.g., RTs and RDs) to be configured for a given VRF.

The subtree of 'vpn-instance-profile' is shown in Figure 6.

```

+--rw l3vpn-ntw
  +--rw vpn-profiles
  |   ...
  +--rw vpn-services
    +--rw vpn-service* [vpn-id]
      +--rw vpn-id                               vpn-common:vpn-id
      ...
      +--rw vpn-instance-profiles
        +--rw vpn-instance-profile* [profile-id]
          +--rw profile-id                       string
          +--rw role?                             identityref
          +--rw local-as?                         inet:as-number
          |   {vpn-common:rtg-bgp}?
          +--rw (rd-choice)?
          |   +--:(directly-assigned)
          |   |   +--rw rd?
          |   |   |   rt-types:route-distinguisher
          |   +--:(directly-assigned-suffix)
          |   |   +--rw rd-suffix?                uint16
          |   +--:(auto-assigned)
          |   |   +--rw rd-auto
          |   |   |   +--rw (auto-mode)?
          |   |   |   |   +--:(from-pool)
          |   |   |   |   |   +--rw rd-pool-name?    string
          |   |   |   |   +--:(full-auto)
          |   |   |   |   |   +--rw auto?            empty
          |   |   |   +--ro auto-assigned-rd?
          |   |   |   |   rt-types:route-distinguisher
          |   +--:(auto-assigned-suffix)
          |   |   +--rw rd-auto-suffix
          |   |   |   +--rw (auto-mode)?
          |   |   |   |   +--:(from-pool)
          |   |   |   |   |   +--rw rd-pool-name?    string
          |   |   |   |   +--:(full-auto)
          |   |   |   |   |   +--rw auto?            empty

```

```

|         +---ro auto-assigned-rd-suffix?   uint16
|         +---:(no-rd)
|         +---rw no-rd?                      empty
+---rw address-family* [address-family]
+---rw address-family          identityref
+---rw vpn-targets
|   +---rw vpn-target* [id]
|   |   +---rw id                      uint8
|   |   +---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 maximum-routes* [protocol]
|   +---rw protocol          identityref
|   +---rw maximum-routes?   uint32
+---rw multicast {vpn-common:multicast}?
...

```

Figure 6: Subtree Structure of VPN Instance Profiles

The description of the listed data nodes is as follows:

'profile-id': Is used to uniquely identify a VPN instance profile.

'role': Indicates the role of the VPN instance profile in the VPN. Role values are defined in [I-D.ietf-opsawg-vpn-common] (e.g., any-to-any-role, spoke-role, hub-role).

'local-as': Indicates the Autonomous System Number (ASN) that is configured for the VPN node.

'rd': As defined in [I-D.ietf-opsawg-vpn-common], the following RD assignment modes are supported: direct assignment, automatic assignment from a given pool, automatic assignment, and no assignment. For illustration purposes, the following modes can be used in the deployment cases:

'directly-assigned': The VPN service provider (service orchestrator) assigns explicitly RDs. This case will fit with a brownfield scenario where some existing services need to be updated by the VPN service provider.

'full-auto': The network controller auto-assigns RDs. This can apply for the deployment of new services.

'no-rd': The VPN service provider (service orchestrator) explicitly wants no RD to be assigned. This case can be used for CE testing within the network or for troubleshooting proposes.

Also, the module accommodates deployments where only the Assigned Number subfield of RDs (Section 4.2 of [RFC4364]) is assigned from a pool while the Administrator subfield is set to, e.g., the Router ID that is assigned to a VPN node. The module supports these modes for managing the Assigned Number subfield: explicit assignment, auto-assignment from a pool, and full auto-assignment.

'address-family': Includes a set of per-address family data nodes:

'address-family': Identifies the address family. It can be set to IPv4, IPv6, or dual-stack.

'vpn-targets': Specifies RT import/export rules for the VPN service (Section 4.3 of [RFC4364]).

'maximum-routes': Indicates the maximum number of prefixes that the VPN node can accept for a given routing protocol. If 'protocol' is set to 'any', this means that the maximum value applies to each active routing protocol.

'multicast': Enables multicast traffic in the VPN service. Refer to Section 7.7.

7.5. VPN Nodes

The 'vpn-node' is an abstraction that represents a set of common policies applied on a given network node (typically, a PE) and belong to one L3VPN service. 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 pointing to an abstract node. In this case, the network controller will decide how to split the 'vpn-node' into VRFs.

```

+--rw l3vpn-ntw
  +--rw vpn-profiles
  |   ...
  +--rw vpn-services
    +--rw vpn-service* [vpn-id]
    |   ...
    +--rw vpn-nodes
      +--rw vpn-node* [vpn-node-id]

```

```

+--rw vpn-node-id                vpn-common:vpn-id
+--rw description?              string
+--rw ne-id?                    string
+--rw local-as?                 inet:as-number
|                               {vpn-common:rtg-bgp}?
+--rw router-id?                rt-types:router-id
+--rw active-vpn-instance-profiles
|   +--rw vpn-instance-profile* [profile-id]
|       +--rw profile-id        leafref
|       +--rw router-id* [address-family]
|           +--rw address-family identityref
|           +--rw router-id?    inet:ip-address
|       +--rw local-as?        inet:as-number
|           {vpn-common:rtg-bgp}?
|       +--rw (rd-choice)?
|           ....
|       +--rw address-family* [address-family]
|           +--rw address-family identityref
|           ...
|       +--rw vpn-targets
|           ...
|       +--rw maximum-routes* [protocol]
|           ...
|       +--rw multicast {vpn-common:multicast}?
|           ...
+--rw msdp {msdp}?
|   +--rw peer?                inet:ipv4-address
|   +--rw local-address?      inet:ipv4-address
|   +--rw status
|       +--rw admin-status
|           +--rw status?      identityref
|           +--rw last-change? yang:date-and-time
|       +--ro oper-status
|           +--ro status?      identityref
|           +--ro last-change? yang:date-and-time
+--rw groups
|   +--rw group* [group-id]
|       +--rw group-id        string
+--rw status
|   +--rw admin-status
|       +--rw status?          identityref
|       +--rw last-change?    yang:date-and-time
|   +--ro oper-status
|       +--ro status?          identityref
|       +--ro last-change?    yang:date-and-time
+--rw vpn-network-accesses
...

```

Figure 7: VPN Node Subtree Structure

In reference to the subtree shown in Figure 7, the description of VPN node data nodes is as follows:

'vpn-node-id': Is an identifier that uniquely identifies a node that enables a VPN network access.

'description': Provides a textual description of the VPN node.

'ne-id': Includes a unique identifier of the network element where the VPN node is deployed.

'local-autonomous-system': Indicates the ASN that is configured for the VPN node.

'router-id': Indicates a 32-bit number that is used to uniquely identify a router within an Autonomous System.

'active-vpn-instance-profiles': Lists the set of active VPN instance profiles for this VPN node. Concretely, one or more VPN instance profiles that are defined at the VPN service level can be enabled at the VPN node level; each of these profiles is uniquely identified by means of 'profile-id'. The structure of 'active-vpn-instance-profiles' is the same as the one discussed in Section 7.4 except 'router-id'. The value of 'router-id' indicated under 'active-vpn-instance-profiles' takes precedence over the 'router-id' under the 'vpn-node' for the indicated address family. For example, Router IDs can be configured per address family. This capability can be used, for example, to configure an IPv6 address as a Router ID when such capability is supported by involved routers.

Values defined in 'active-vpn-instance-profiles' overrides the ones defined in the VPN service level. An example is shown in Appendix A.3.

'msdp': For redundancy purposes, Multicast Source Discovery Protocol (MSDP) [RFC3618] may be enabled and used to share the state about sources between multiple Rendezvous Points (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.

'groups': Lists the groups to which a VPN node belongs to

[I-D.ietf-opsawg-vpn-common]. The 'group-id' is used to associate, e.g., redundancy or protection constraints with VPN nodes.

'status': Tracks the status of a node involved in a VPN service. Both operational and administrative status are maintained. A mismatch between the administrative status vs. the operational status can be used as a trigger to detect anomalies.

'vpn-network-accesses': Represents the point to which sites are connected.

Note that, unlike in the 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 that can be applied during the service creation.

See Section 7.6 for more details.

7.6. VPN Network Accesses

The 'vpn-network-access' includes a set of data nodes that describe the access information for the traffic that belongs to a particular L3VPN (Figure 8).

```

...
+--rw vpn-nodes
  +--rw vpn-node* [vpn-node-id]
    ...
    +--rw vpn-network-accesses
      +--rw vpn-network-access* [id]
        +--rw id                               vpn-common:vpn-id
        +--rw interface-id?                    string
        +--rw description?                     string
        +--rw vpn-network-access-type?         identityref
        +--rw vpn-instance-profile?            leafref
        +--rw status
          +--rw admin-status
            +--rw status?                      identityref
            +--rw last-change?                 yang:date-and-time
          +--ro oper-status
            +--ro status?                     identityref
            +--ro last-change?                 yang:date-and-time
        +--rw connection
          | ...
        +--rw ip-connection
          | ...
        +--rw routing-protocols
          | ...
        +--rw oam
          | ...
        +--rw security
          | ...
        +--rw service
          ...

```

Figure 8: VPN Network Access Subtree Structure

In reference to the subtree depicted in Figure 8, a 'vpn-network-access' includes the following data nodes:

'id': Is an identifier of the VPN network access.

'interface-id': Indicates the physical or logical interface on which the VPN network access is bound.

'description': Includes a textual description of the VPN network access.

'vpn-network-access-type': Is used to select the type of network interface to be deployed in the devices. The available defined values are:

- 'point-to-point': Represents a direct connection between the endpoints. The controller must keep the association between a logical or physical interface on the device with the 'id' of the 'vpn-network-access'.
- 'multipoint': Represents a multipoint connection between the customer site and the PEs. The controller must keep the association between a logical or physical interface on the device with the 'id' of the 'vpn-network-access'.
- 'irb': Represents a connection coming from an L2VPN service. An identifier of such service ('l2vpn-id') may be included in the 'connection' container as depicted in Figure 9. The controller must keep the relationship between the logical tunnels or bridges on the devices with the 'id' of the 'vpn-network-access'.
- 'loopback': Represents the creation of a logical interface on a device. An example to illustrate how a loopback interface can be used in the L3NM is provided in Appendix A.2.
- 'vpn-instance-profile': Provides a pointer to an active VPN instance profile at the VPN node level. Referencing an active VPN instance profile implies that all associated data nodes will be inherited by the VPN network access. However, some inherited data nodes (e.g., multicast) can be overridden at the VPN network access level. In such case, adjusted values take precedence over inherited ones.
- 'status': Indicates both operational and administrative status of a VPN network access.
- 'connection': Represents and groups the set of Layer 2 connectivity from where the traffic of the L3VPN in a particular VPN Network access is coming. See Section 7.6.1.
- 'ip-connection': Contains Layer 3 connectivity information of a VPN network access (e.g., IP addressing). See Section 7.6.2.
- 'routing-protocols': Includes the CE-PE routing configuration information. See Section 7.6.3.
- 'oam': Specifies the Operations, Administration, and Maintenance (OAM) mechanisms used for a VPN network access. See Section 7.6.4.
- 'security': Specifies the authentication and the encryption to be applied for a given VPN network access. See Section 7.6.5.

'service': Specifies the service parameters (e.g., QoS, multicast) to apply for a given VPN network access. See Section 7.6.6.

7.6.1. Connection

The 'connection' container represents the layer 2 connectivity to the L3VPN for a particular VPN network access. As shown in the tree depicted in Figure 9, the 'connection' container defines protocols and parameters to enable such connectivity at layer 2.

The traffic can enter the VPN with or without encapsulation (e.g., VLAN, QinQ). The 'encapsulation' container specifies the layer 2 encapsulation to use (if any) and allows to configure the relevant tags.

The interface that is attached to the L3VPN is identified by the 'interface-id' at the 'vpn-network-access' level. From a network model perspective, it is expected that the 'interface-id' is sufficient to identify the interface. However, specific layer 2 sub-interfaces may be required to be configured in some implementations/deployments. Such a layer 2 specific interface can be included in 'l2-termination-point'.

If a layer 2 tunnel is needed to terminate the service in the CE-PE connection, the 'l2-tunnel-service' container is used to specify the required parameters to set such tunneling service (e.g., VPLS, VXLAN). An identity, called 'l2-tunnel-type', is defined for layer 2 tunnel selection. The container can also identify the pseudowire (Section 6.1 of [RFC8077]).

As discussed in Section 7.6, 'l2vpn-id' is used to identify the L2VPN service that is associated with an IRB interface.

To accommodate implementations that require internal bridging, a local bridge reference can be specified in 'local-bridge-reference'. Such a reference may be a local bridge domain.

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 connection with the site. In the L3NM, it is assumed that the bearer has been allocated by the service provider at the service orchestration stage. The bearer is associated to a network element and a port. Hence, a bearer is just a 'bearer-reference' to allow the association between a service request (e.g., L3SM) and L3NM.

The L3NM can be used to create a LAG interface for a given L3VPN service ('lag-interface') [IEEE802.1AX]. Such a LAG interface can be referenced under 'interface-id' (Section 7.6).

```

...
+--rw connection
|
|   +--rw encapsulation
|   |   +--rw type?          identityref
|   |   +--rw dot1q
|   |   |   +--rw tag-type?    identityref
|   |   |   +--rw cvlan-id?    uint16
|   |   +--rw priority-tagged
|   |   |   +--rw tag-type?    identityref
|   |   +--rw qinq
|   |   |   +--rw tag-type?    identityref
|   |   |   +--rw svlan-id     uint16
|   |   |   +--rw cvlan-id     uint16
|   +--rw (l2-service)?
|   |   +--:(l2-tunnel-service)
|   |   |   +--rw l2-tunnel-service
|   |   |   |   +--rw type?          identityref
|   |   |   |   +--rw pseudowire
|   |   |   |   |   +--rw vcid?      uint32
|   |   |   |   |   +--rw far-end?   union
|   |   |   |   +--rw vpls
|   |   |   |   |   +--rw vcid?      uint32
|   |   |   |   |   +--rw far-end*   union
|   |   |   +--rw vxlan
|   |   |   |   +--rw vni-id          uint32
|   |   |   |   +--rw peer-mode?      identityref
|   |   |   |   +--rw peer-ip-address* inet:ip-address
|   |   +--:(l2vpn)
|   |   |   +--rw l2vpn-id?          vpn-common:vpn-id
|   +--rw l2-termination-point?      string
|   +--rw local-bridge-reference?     string
|   +--rw bearer-reference?           string
|   |   {vpn-common:bearer-reference}?
|   +--rw lag-interface {vpn-common:lag-interface}?
|   |   +--rw lag-interface-id?      string
|   |   +--rw member-link-list
|   |   |   +--rw member-link* [name]
|   |   |   +--rw name           string
...

```

Figure 9: Connection Subtree Structure

7.6.2. IP Connection

This container is used to group Layer 3 connectivity information, particularly the IP addressing information, of a VPN network access. The allocated address represents the PE interface address configuration. Note that a distinct layer 3 interface other than the one indicated under the 'connection' container may be needed to terminate the layer 3 service. The identifier of such interface is included in 'l3-termination-point'. For example, this data node can be used to carry the identifier of a bridge domain interface.

As shown in Figure 10, the 'ip-connection' container can include IPv4, IPv6, or both if dual-stack is enabled.

```

...
+--rw vpn-network-accesses
  +--rw vpn-network-access* [id]
    ...
    +--rw ip-connection
      +--rw l3-termination-point?      string
      +--rw ipv4 {vpn-common:ipv4}?
      |   ...
      +--rw ipv6 {vpn-common:ipv6}?
      |   ...
    ...
  ...

```

Figure 10: IP Connection Subtree Structure

For both IPv4 and IPv6, the IP connection supports three IP address assignment modes for customer addresses: provider DHCP, DHCP relay, and static addressing. Note that for the IPv6 case, SLAAC [RFC4862] can be used. For both IPv4 and IPv6, 'address-allocation-type' is used to indicate the IP address allocation mode to activate for a given VPN network access.

When 'address-allocation-type' is set to 'provider-dhcp', DHCP assignments can be made locally or by an external DHCP server. Such as behavior is controlled by setting 'dhcp-service-type'.

Figure 11 shows the structure of the dynamic IPv4 address assignment (i.e., by means of DHCP).

```

...
+--rw ip-connection
|   +--rw l3-termination-point?      string
|   +--rw ipv4 {vpn-common:ipv4}?
|   |   +--rw local-address?          inet:ipv4-address
|   |   +--rw prefix-length?          uint8
|   |   +--rw address-allocation-type? identityref
|   |   +--rw (allocation-type)?
|   |   |   +--:(provider-dhcp)
|   |   |   |   +--rw dhcp-service-type? enumeration
|   |   |   |   +--rw (service-type)?
|   |   |   |   |   +--:(relay)
|   |   |   |   |   |   +--rw server-ip-address*
|   |   |   |   |   |   |   inet:ipv4-address
|   |   |   |   |   +--:(server)
|   |   |   |   |   |   +--rw (address-assign)?
|   |   |   |   |   |   |   +--:(number)
|   |   |   |   |   |   |   |   +--rw number-of-dynamic-address?
|   |   |   |   |   |   |   |   |   uint16
|   |   |   |   |   |   |   +--:(explicit)
|   |   |   |   |   |   |   |   +--rw customer-addresses
|   |   |   |   |   |   |   |   |   +--rw address-pool* [pool-id]
|   |   |   |   |   |   |   |   |   |   +--rw pool-id      string
|   |   |   |   |   |   |   |   |   |   +--rw start-address
|   |   |   |   |   |   |   |   |   |   |   inet:ipv4-address
|   |   |   |   |   |   |   |   |   |   +--rw end-address?
|   |   |   |   |   |   |   |   |   |   |   inet:ipv4-address
|   |   |   |   |   +--:(dhcp-relay)
|   |   |   |   |   |   +--rw customer-dhcp-servers
|   |   |   |   |   |   |   +--rw server-ip-address*  inet:ipv4-address
|   |   |   |   +--:(static-addresses)
|   |   |   ...
|   ...
...

```

Figure 11: IP Connection Subtree Structure (IPv4)

Figure 12 shows the structure of the dynamic IPv6 address assignment (i.e., DHCPv6 and/or SLAAC). Note that if 'address-allocation-type' is set to 'slaac', the Prefix Information option of Router Advertisements that will be issued for SLAAC purposes, will carry the IPv6 prefix that is determined by 'local-address' and 'prefix-length'. For example, if 'local-address' is set to '2001:db8:0:1::1' and 'prefix-length' is set to '64', the IPv6 prefix that will be used is '2001:db8:0:1::/64'.

```

...
+--rw ip-connection
|   +--rw l3-termination-point?      string
|   +--rw ipv4 {vpn-common:ipv4}?
|   |   ...
|   +--rw ipv6 {vpn-common:ipv6}?
|   |   +--rw local-address?          inet:ipv6-address
|   |   +--rw prefix-length?          uint8
|   |   +--rw address-allocation-type? identityref
|   |   +--rw (allocation-type)?
|   |   |   +--:(provider-dhcp)
|   |   |   |   +--rw provider-dhcp
|   |   |   |   |   +--rw dhcp-service-type?
|   |   |   |   |   |   enumeration
|   |   |   |   |   +--rw (service-type)?
|   |   |   |   |   |   +--:(relay)
|   |   |   |   |   |   |   +--rw server-ip-address*
|   |   |   |   |   |   |   |   inet:ipv6-address
|   |   |   |   |   |   +--:(server)
|   |   |   |   |   |   |   +--rw (address-assign)?
|   |   |   |   |   |   |   |   +--:(number)
|   |   |   |   |   |   |   |   |   +--rw number-of-dynamic-address?
|   |   |   |   |   |   |   |   |   |   uint16
|   |   |   |   |   |   |   |   +--:(explicit)
|   |   |   |   |   |   |   |   |   +--rw customer-addresses
|   |   |   |   |   |   |   |   |   |   +--rw address-pool* [pool-id]
|   |   |   |   |   |   |   |   |   |   |   +--rw pool-id      string
|   |   |   |   |   |   |   |   |   |   |   +--rw start-address
|   |   |   |   |   |   |   |   |   |   |   |   inet:ipv6-address
|   |   |   |   |   |   |   |   |   |   |   +--rw end-address?
|   |   |   |   |   |   |   |   |   |   |   |   inet:ipv6-address
|   |   |   |   |   |   +--:(dhcp-relay)
|   |   |   |   |   |   |   +--rw customer-dhcp-servers
|   |   |   |   |   |   |   |   +--rw server-ip-address*
|   |   |   |   |   |   |   |   |   inet:ipv6-address
|   |   |   +--:(static-addresses)
|   |   |   ...

```

Figure 12: IP Connection Subtree Structure (IPv6)

In the case of the static addressing (Figure 13), 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 ip-connection
|   +--rw l3-termination-point?      string
|   +--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 customer-address?     inet:ipv4-address
|   +--rw ipv6 {vpn-common:ipv6}?
|   |   +--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 customer-address?     inet:ipv6-address
...

```

Figure 13: IP Connection Subtree Structure (Static Mode)

7.6.3. CE-PE Routing Protocols

A VPN service provider can configure one or more routing protocols associated with a particular 'vpn-network-access'. Such routing protocols are enabled between the PE and the CE. Each instance is uniquely identified to accommodate scenarios where multiple instances of the same routing protocol have to be configured on the same link.

The subtree of the 'routing-protocols' is shown in Figure 14.

```

...
+--rw vpn-network-accesses
  +--rw vpn-network-access* [id]
    ...
    +--rw routing-protocols
      +--rw routing-protocol* [id]
        +--rw id      string
        +--rw type?   identityref
        +--rw routing-profiles* [id]
          +--rw id      leafref
          +--rw type?   identityref
        +--rw static
          |
          | ...
          +--rw bgp
            |
            | ...
            +--rw ospf
              |
              | ...
              +--rw isis
                |
                | ...
                +--rw rip
                  |
                  | ...
                  +--rw vrrp
                    |
                    | ...
                    +--rw security
                      ...

```

Figure 14: Routing Subtree Structure

Multiple routing instances can be defined; each uniquely identified by an 'id'. The type of routing instance is indicated in 'type'. The values of these attributes are those defined in [I-D.ietf-opsawg-vpn-common] ('routing-protocol-type' identity).

Configuring multiple instances of the same routing protocol does not automatically imply that, from a device configuration perspective, there will be parallel instances (e.g., multiple processes) running on the PE-CE link. It is up to each implementation (typically, network orchestration shown in Figure 1) to decide about the appropriate configuration as a function of underlying capabilities and service provider operational guidelines. 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 thereof.

Routing configuration does not include low-level policies. Such policies are handled at the device configuration level. Local policies of a service provider (e.g., filtering) are implemented as part of the device configuration; these are not captured in the L3NM, but the model allows local profiles to be associated with routing instances ('routing-profiles'). Note that these routing profiles can be scoped to capture parameters that are globally applied to all L3VPN services within a service provider network, while customized L3VPN parameters are captured by means of the L3NM. The provisioning of an L3VPN service will, thus, rely upon the instantiation of these global routing profiles and the customized L3NM.

7.6.3.1. Static Routing

The L3NM supports the configuration of one or more IPv4/IPv6 static routes. Since the same structure is used for both IPv4 and IPv6, it was considered to have one single container to group both static entries independently of their address family, but that design was abandoned to ease the mapping with the structure in [RFC8299].

```

...
+--rw routing-protocols
|   +--rw routing-protocol* [id]
|   |   ...
|   |   +--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       union
|   |   |   |   |   +--rw bfd-enable?    boolean
|   |   |   |   |   +--rw metric?        uint32
|   |   |   |   |   +--rw preference?    uint32
|   |   |   |   |   +--rw status
|   |   |   |   |   |   +--rw admin-status
|   |   |   |   |   |   |   +--rw status?          identityref
|   |   |   |   |   |   |   +--rw last-change?     yang:date-and-time
|   |   |   |   |   |   +--ro oper-status
|   |   |   |   |   |   |   +--ro status?          identityref
|   |   |   |   |   |   |   +--ro last-change?     yang:date-and-time
|   |   |   |   +--rw ipv6-lan-prefixes*
|   |   |   |   |   [lan next-hop]
|   |   |   |   |   {vpn-common:ipv6}?
|   |   |   |   |   +--rw lan          inet:ipv6-prefix
|   |   |   |   |   +--rw lan-tag?       string
|   |   |   |   |   +--rw next-hop       union
|   |   |   |   |   +--rw bfd-enable?    boolean
|   |   |   |   |   +--rw metric?        uint32
|   |   |   |   |   +--rw preference?    uint32
|   |   |   |   |   +--rw status
|   |   |   |   |   |   +--rw admin-status
|   |   |   |   |   |   |   +--rw status?          identityref
|   |   |   |   |   |   |   +--rw last-change?     yang:date-and-time
|   |   |   |   |   |   +--ro oper-status
|   |   |   |   |   |   |   +--ro status?          identityref
|   |   |   |   |   |   |   +--ro last-change?     yang:date-and-time
|   |   |   |
|   |   |   +--rw ...
|   |   |
|   |   +--rw ...
|   |
|   +--rw ...
|
+--rw ...

```

Figure 15: Static Routing Subtree Structure

As depicted in Figure 15, the following data nodes can be defined for a given IP prefix:

'lan-tag': Indicates a local tag (e.g., "myfavourite-lan") that is used to enforce local policies.

- 'next-hop': Indicates the next-hop to be used for the static route. It can be identified by an IP address, an interface, etc.
- 'bfd-enable': Indicates whether BFD is enabled or disabled for this static route entry.
- 'metric': Indicates the metric associated with the static route entry.
- 'preference': Indicates the preference associated with the static route entry. This preference is used to selecting a preferred route among routes to the same destination prefix.
- 'status': Used to convey the status of a static route entry. This data node can also be used to control the (de)activation of individual static route entries.

7.6.3.2. BGP

The L3NM allows the configuration of a BGP neighbor, including a set for parameters that are pertinent to be tweaked at the network level for service customization purposes. The 'bgp' container does not aim to include every BGP parameter; a comprehensive set of parameters belongs more to the BGP device model.

```

...
+--rw routing-protocols
|   +--rw routing-protocol* [id]
|   |   ...
|   |   +--rw bgp
|   |   |   +--rw description?          string
|   |   |   +--rw local-as?             inet:as-number
|   |   |   +--rw peer-as               inet:as-number
|   |   |   +--rw address-family?       identityref
|   |   |   +--rw local-address?        union
|   |   |   +--rw neighbor*             inet:ip-address
|   |   |   +--rw multihop?             uint8
|   |   |   +--rw as-override?          boolean
|   |   |   +--rw allow-own-as?         uint8
|   |   |   +--rw prepend-global-as?    boolean
|   |   |   +--rw send-default-route?   boolean
|   |   |   +--rw site-of-origin?       rt-types:route-origin
|   |   |   +--rw ipv6-site-of-origin?  rt-types:ipv6-route-origin
|   |   |   +--rw redistribute-connected* [address-family]
|   |   |   |   +--rw address-family    identityref
|   |   |   |   +--rw enable?          boolean
|   |   +--rw bgp-max-prefix

```

```

| | | | | +--rw max-prefix?          uint32
| | | | | +--rw warning-threshold?  decimal64
| | | | | +--rw violate-action?      enumeration
| | | | | +--rw restart-timer?       uint32
| | | | +--rw bgp-timers
| | | | | +--rw keepalive?          uint16
| | | | | +--rw hold-time?         uint16
| | | | +--rw authentication
| | | | | +--rw enable?             boolean
| | | | | +--rw keying-material
| | | | |   +--rw (option)?
| | | | |     +--:(ao)
| | | | |       +--rw enable-ao?      boolean
| | | | |       +--rw ao-keychain?    key-chain:key-chain-ref
| | | | |     +--:(md5)
| | | | |       +--rw md5-keychain?   key-chain:key-chain-ref
| | | | |     +--:(explicit)
| | | | |       +--rw key-id?         uint32
| | | | |       +--rw key?           string
| | | | |       +--rw crypto-algorithm? identityref
| | | | |     +--:(ipsec)
| | | | |       +--rw sa?             string
| | | | +--rw status
| | | | | +--rw admin-status
| | | | |   +--rw status?             identityref
| | | | |   +--rw last-change?       yang:date-and-time
| | | | +--ro oper-status
| | | | | +--rw status?             identityref
| | | | | +--ro last-change?       yang:date-and-time
| | | ...

```

Figure 16: BGP Routing Subtree Structure

The following data nodes are captured in Figure 16. It is up to the implementation (e.g., network orchestrator) to derive the corresponding BGP device configuration:

'description': Includes a description of the BGP session.

'local-as': Indicates a local AS Number (ASN) if a distinct ASN is required, other than the one configured at the VPN node level.

'peer-as': Conveys the customer's ASN.

'address-family': Indicates the address-family of the peer. It can be set to IPv4, IPv6, or dual-stack.

This address family will be used together with the 'vpn-type' to derive the appropriate Address Family Identifiers (AFIs)/Subsequent Address Family Identifiers (SAFIs) that will be part of the derived device configurations (e.g., Unicast IPv4 MPLS L3VPN (AFI,SAFI = 1,128) defined in Section 4.3.4 of [RFC4364]).

- 'local-address': Specifies an address or a reference to an interface to use when establishing the BGP transport session.
- 'neighbor': Can indicate two neighbors (each for a given address-family) or one neighbor (if 'address-family' attribute is set to dual-stack). A list of IP address(es) of the BGP neighbors can be then conveyed in this data node.
- 'multihop': Indicates the number of allowed IP hops between a PE and its BGP peer.
- 'as-override': If set, this parameter indicates whether ASN override is enabled, i.e., replace the ASN of the customer specified in the AS_PATH BGP attribute with the ASN identified in the 'local-as' attribute.
- 'allow-own-as': Is used in some topologies (e.g., hub-and-spoke) to allow the provider's ASN to be included in the AS_PATH BGP attribute received from a CE. Loops are prevented by setting 'allow-own-as' to a maximum number of provider's ASN occurrences. This parameter is set by default to '0' (that is, reject any AS_PATH attribute that includes the provider's ASN).
- 'prepend-global-as': When distinct ASNs are configured in the VPN node and network access levels, this parameter controls whether the ASN provided at the VPN node level is prepended to the AS_PATH attribute.
- 'send-default-route': Controls whether default routes can be advertised to the peer.
- 'site-of-origin': Is meant to uniquely identify the set of routes learned from a site via a particular CE/PE connection and is used to prevent routing loops (Section 7 of [RFC4364]). The Site of Origin attribute is encoded as a Route Origin Extended Community.
- 'ipv6-site-of-origin': Carries an IPv6 Address Specific BGP Extended Community that is used to indicate the Site of Origin for VRF information [RFC5701]. It is used to prevent routing loops.
- 'redistribute-connected': Controls whether the PE-CE link is advertised to other PEs.

- 'bgp-max-prefix': Controls the behavior when a prefix maximum is reached.
- 'max-prefix': Indicates the maximum number of BGP prefixes allowed in the BGP session. If the limit is reached, the action indicated in 'violate-action' will be followed.
- 'warning-threshold': A warning notification is triggered when this limit is reached.
- 'violate-action': Indicates which action to execute when the maximum number of BGP prefixes is reached. Examples of such actions are: send a warning message, discard extra paths from the peer, or restart the session.
- 'restart-timer': Indicates, in seconds, the time interval after which the BGP session will be reestablished.
- 'bgp-timers': 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) 'keepalive' which is the time interval for the KeepAlive timer between a PE and a BGP peer (Section 4.4 of [RFC4271]). Both timers are expressed in seconds.
- 'authentication': The module adheres to the recommendations in Section 13.2 of [RFC4364] as it allows enabling TCP-AO [RFC5925] and accommodates the installed base that makes use of MD5. In addition, the module includes a provision for the use of IPsec.
- This version of the L3NM assumes that TCP-AO specific parameters are preconfigured as part of the key-chain that is referenced in the L3NM. No assumption is made about how such a key-chain is pre-configured. However, the structure of the key-chain should cover data nodes beyond those in [RFC8177], mainly SendID and RecvID (Section 3.1 of [RFC5925]).
- 'status': Indicates the status of the BGP routing instance.

7.6.3.3. OSPF

OSPF can be configured to run as a routing protocol on the 'vpn-network-access'.

```

...
+--rw routing-protocols
|   +--rw routing-protocol* [id]
|   |   ...
|   |   +--rw ospf
|   |   |   +--rw address-family?    identityref
|   |   |   +--rw area-id             yang:dotted-quad
|   |   |   +--rw metric?             uint16
|   |   |   +--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 max-lsa?          uint32
|   |   |   +--rw authentication
|   |   |   |   +--rw enable?        boolean
|   |   |   |   +--rw keying-material
|   |   |   |   |   +--rw (option)?
|   |   |   |   |   |   +--:(auth-key-chain)
|   |   |   |   |   |   |   +--rw key-chain?
|   |   |   |   |   |   |   |   key-chain:key-chain-ref
|   |   |   |   |   |   +--:(auth-key-explicit)
|   |   |   |   |   |   |   +--rw key-id?    uint32
|   |   |   |   |   |   |   +--rw key?      string
|   |   |   |   |   |   |   +--rw crypto-algorithm?
|   |   |   |   |   |   |   |   identityref
|   |   |   |   |   +--:(ipsec)
|   |   |   |   |   |   +--rw sa?    string
|   |   |   +--rw status
|   |   |   |   +--rw admin-status
|   |   |   |   |   +--rw status?    identityref
|   |   |   |   |   +--rw last-change? yang:date-and-time
|   |   |   +--ro oper-status
|   |   |   |   +--ro status?    identityref
|   |   |   |   +--ro last-change? yang:date-and-time
|   |   ...
|   ...
...

```

Figure 17: OPSF Routing Subtree Structure

The following data nodes are captured in Figure 17:

'address-family': Indicates whether IPv4, IPv6, or both address families are to be activated.

When the IPv4 or dual-stack address-family is requested, it is up to the implementation (e.g., network orchestrator) to decide whether OSPFv2 [RFC4577] or OSPFv3 [RFC6565] is used to announce IPv4 routes. Such decision will be typically reflected in the device configurations that will be derived to implement the L3VPN.

- 'area-id': Indicates the OSPF Area ID.
- 'metric': Associates a metric with OSPF routes.
- 'sham-links': Is used to create OSPF sham links between two VPN network accesses sharing the same area and having a backdoor link (Section 4.2.7 of [RFC4577] and Section 5 of [RFC6565]).
- 'max-lsa': Sets the maximum number of LSAs that the OSPF instance will accept.
- 'authentication': Controls the authentication schemes to be enabled for the OSPF instance. The following options are supported: IPsec for OSPFv3 authentication [RFC4552], authentication trailer for OSPFv2 [RFC5709] [RFC7474] and OSPFv3 [RFC7166].
- 'status': Indicates the status of the OSPF routing instance.

7.6.3.4. IS-IS

The model (Figure 18) allows the user to configure IS-IS [ISO10589][RFC1195][RFC5308] to run on the 'vpn-network-access' interface.

```

...
+--rw routing-protocols
|   +--rw routing-protocol* [id]
|   |   ...
|   |   +--rw isis
|   |   |   +--rw address-family?    identityref
|   |   |   +--rw area-address        area-address
|   |   |   +--rw level?              identityref
|   |   |   +--rw metric?             uint16
|   |   |   +--rw mode?               enumeration
|   |   |   +--rw authentication
|   |   |   |   +--rw enable?          boolean
|   |   |   |   +--rw keying-material
|   |   |   |   |   +--rw (option)?
|   |   |   |   |   |   +--:(auth-key-chain)
|   |   |   |   |   |   |   +--rw key-chain?
|   |   |   |   |   |   |   |   key-chain:key-chain-ref
|   |   |   |   |   |   |   +--:(auth-key-explicit)
|   |   |   |   |   |   |   |   +--rw key-id?          uint32
|   |   |   |   |   |   |   |   +--rw key?            string
|   |   |   |   |   |   |   |   +--rw crypto-algorithm? identityref
|   |   |   +--rw status
|   |   |   |   +--rw admin-status
|   |   |   |   |   +--rw status?      identityref
|   |   |   |   |   +--rw last-change? yang:date-and-time
|   |   |   +--ro oper-status
|   |   |   |   +--rw status?          identityref
|   |   |   |   +--ro last-change?    yang:date-and-time
|   |   ...
|   ...
...

```

Figure 18: IS-IS Routing Subtree Structure

The following IS-IS data nodes are supported:

- 'address-family': Indicates whether IPv4, IPv6, or both address families are to be activated.
- 'area-address': Indicates the IS-IS area address.
- 'level': Indicates the IS-IS level: Level 1, Level 2, or both.
- 'metric': Associates a metric with IS-IS routes.
- 'mode': Indicates the IS-IS interface mode type. It can be set to 'active' (that is, send or receive IS-IS protocol control packets) or 'passive' (that is, suppress the sending of IS-IS updates through the interface).

'authentication': Controls the authentication schemes to be enabled for the IS-IS instance. Both the specification of a key-chain [RFC8177] and the direct specification of key and authentication algorithm are supported.

'status': Indicates the status of the IS-IS routing instance.

7.6.3.5. RIP

The model (Figure 19) allows the user to configure RIP to run on the 'vpn-network-access' interface.

```

...
+--rw routing-protocols
|   +--rw routing-protocol* [id]
|   |   ...
|   |   +--rw rip
|   |   |   +--rw address-family?    identityref
|   |   |   +--rw timers
|   |   |   |   +--rw update-interval?    uint16
|   |   |   |   +--rw invalid-interval?    uint16
|   |   |   |   +--rw holddown-interval?    uint16
|   |   |   |   +--rw flush-interval?    uint16
|   |   |   +--rw neighbor*            inet:ip-address
|   |   |   +--rw default-metric?    uint8
|   |   |   +--rw authentication
|   |   |   |   +--rw enable?            boolean
|   |   |   |   +--rw keying-material
|   |   |   |   |   +--rw (option)?
|   |   |   |   |   |   +--:(auth-key-chain)
|   |   |   |   |   |   |   +--rw key-chain?
|   |   |   |   |   |   |   |   key-chain:key-chain-ref
|   |   |   |   |   |   +--:(auth-key-explicit)
|   |   |   |   |   |   |   +--rw key?            string
|   |   |   |   |   |   |   +--rw crypto-algorithm? identityref
|   |   |   +--rw status
|   |   |   |   +--rw admin-status
|   |   |   |   |   +--rw status?            identityref
|   |   |   |   |   +--rw last-change?    yang:date-and-time
|   |   |   +--ro oper-status
|   |   |   |   +--ro status?            identityref
|   |   |   |   +--ro last-change?    yang:date-and-time
|   |   ...
|   ...
...

```

Figure 19: RIP Subtree Structure

As shown in Figure 19, the following RIP data nodes are supported:

'address-family': Indicates whether IPv4, IPv6, or both address families are to be activated. This parameter is used to determine whether RIPv2 [RFC2453] and/or RIPv6 [RFC2080] are to be enabled [RFC2080].

'timers': Indicates the following timers:

- 'update-interval': Is the interval at which RIP updates are sent.
- 'invalid-interval': Is the interval before a RIP route is declared invalid.
- 'holddown-interval': Is the interval before better RIP routes are released.
- 'flush-interval': Is the interval before a route is removed from the routing table.

These timers are expressed in seconds.

'default-metric': Sets the default RIP metric.

'authentication': Controls the authentication schemes to be enabled for the RIP instance.

'status': Indicates the status of the RIP routing instance.

7.6.3.6. VRRP

The model (Figure 20) allows enabling VRRP on the 'vpn-network-access' interface.

```

...
+--rw routing-protocols
|   +--rw routing-protocol* [id]
|   |   ...
|   |   +--rw vrrp
|   |   |   +--rw address-family*    identityref
|   |   |   +--rw vrrp-group?        uint8
|   |   |   +--rw backup-peer?       inet:ip-address
|   |   |   +--rw virtual-ip-address* inet:ip-address
|   |   |   +--rw priority?          uint8
|   |   |   +--rw ping-reply?        boolean
|   |   |   +--rw status
|   |   |   |   +--rw admin-status
|   |   |   |   |   +--rw status?      identityref
|   |   |   |   |   +--rw last-change? yang:date-and-time
|   |   |   +--ro oper-status
|   |   |   |   +--ro status?          identityref
|   |   |   |   +--ro last-change?    yang:date-and-time
|   |   ...
|   ...
...

```

Figure 20: VRRP Subtree Structure

The following data nodes are supported:

'address-family': Indicates whether IPv4, IPv6, or both address families are to be activated. Note that VRRP version 3 [RFC5798] supports both IPv4 and IPv6.

'vrrp-group': Is used to identify the VRRP group.

'backup-peer': Carries the IP address of the peer.

'virtual-ip-address': Includes virtual IP addresses for a single VRRP group.

'priority': Assigns the VRRP election priority for the backup virtual router.

'ping-reply': Controls whether ping requests can be replied to.

'status': Indicates the status of the VRRP instance.

Note that no authentication data node is included for VRRP as there isn't currently any type of VRRP authentication (see Section 9 of [RFC5798]).

7.6.4. OAM

This container (Figure 21) defines the Operations, Administration, and Maintenance (OAM) mechanisms used for a VPN network access. In the current version of the L3NM, only BFD is supported.

```

...
+--rw oam
|   +--rw bfd {vpn-common:bfd}?
|       +--rw session-type?      identityref
|       +--rw desired-min-tx-interval?  uint32
|       +--rw required-min-rx-interval?  uint32
|       +--rw local-multiplier?    uint8
|       +--rw holdtime?            uint32
|       +--rw profile?             leafref
|       +--rw authentication!
|           +--rw key-chain?      key-chain:key-chain-ref
|           +--rw meticulous?     boolean
|       +--rw status
|           +--rw admin-status
|               +--rw status?      identityref
|               +--rw last-change? yang:date-and-time
|           +--ro oper-status
|               +--ro status?      identityref
|               +--ro last-change? yang:date-and-time
|
...

```

Figure 21: IP Connection Subtree Structure (OAM)

The following OAM data nodes can be specified:

'session-type': Indicates which BFD flavor is used to set up the session (e.g., classic BFD [RFC5880], Seamless BFD [RFC7880]). By default, the BFD session is assumed to follow the behavior specified in [RFC5880].

'desired-min-tx-interval': Is the minimum interval, in microseconds, that a PE would like to use when transmitting BFD Control packets less any jitter applied.

'required-min-rx-interval': Is the minimum interval, in microseconds, between received BFD Control packets that a PE is capable of supporting, less any jitter applied by the sender.

'local-multiplier': The negotiated transmit interval, multiplied by this value, provides the detection time for the peer.

'holdtime': Is used to indicate the expected BFD holddown time, in

milliseconds. This value may be inherited from the service request (see Section 6.3.2.2.2 of [RFC8299]).

'profile': Refers to a BFD profile (Section 7.2). Such a profile can be set by the provider or inherited from the service request (see Section 6.3.2.2.2 of [RFC8299]).

'authentication': Includes the required information to enable the BFD authentication modes discussed in Section 6.7 of [RFC5880]. In particular 'meticulous' controls the activation of the meticulous mode discussed in Sections 6.7.3 and 6.7.4 of [RFC5880].

'status': Indicates the status of BFD.

7.6.5. Security

The 'security' container specifies the authentication and the encryption to be applied for a given VPN network access traffic. As depicted in the subtree shown in Figure 22, the L3NM can be used to directly control the encryption to put in place (e.g., Layer 2 or Layer 3 encryption) or invoke a local encryption profile.

```

...
+--rw vpn-services
  +--rw vpn-service* [vpn-id]
    ...
    +--rw vpn-nodes
      +--rw vpn-node* [vpn-node-id]
        ...
        +--rw vpn-network-accesses
          +--rw vpn-network-access* [id]
            ...
            +--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 customer-key-chain?
                        kc:key-chain-ref
              +--rw service
            ...

```

Figure 22: Security Subtree Structure

7.6.6. Services

7.6.6.1. Overview

The 'service' container specifies the service parameters to apply for a given VPN network access (Figure 23).

```

...
+--rw vpn-network-accesses
  +--rw vpn-network-access* [id]
    ...
    +--rw service
      +--rw inbound-bandwidth?   uint64 {vpn-common:inbound-bw}?
      +--rw outbound-bandwidth?  uint64 {vpn-common:outbound-bw}?
      +--rw mtu?                  uint32
      +--rw qos {vpn-common:qos}?
      |   ...
      +--rw carriers-carrier
      |   {vpn-common:carriers-carrier}?
      |   +--rw signaling-type?  enumeration
      +--rw ntp
      |   +--rw broadcast?       enumeration
      |   +--rw auth-profile
      |   |   +--rw profile-id?  string
      |   +--rw status
      |   |   +--rw admin-status
      |   |   |   +--rw status?      identityref
      |   |   |   +--rw last-change? yang:date-and-time
      |   |   +--ro oper-status
      |   |   |   +--ro status?      identityref
      |   |   |   +--ro last-change? yang:date-and-time
      +--rw multicast {vpn-common:multicast}?
    ...

```

Figure 23: Services Subtree Structure

The following data nodes are defined:

'inbound-bandwidth': Indicates, in bits per second (bps), the inbound bandwidth of the connection (i.e., download bandwidth from the service provider to the site).

'outbound-bandwidth': Indicates, in bps, the outbound bandwidth of the connection (i.e., upload bandwidth from the site to the service provider).

'mtu': Indicates the MTU at the service level.

'qos': Is used to define a set of QoS policies to apply on a given connection (refer to Section 7.6.6.2 for more details).

'carriers-carrier': Groups a set of parameters that are used when Carriers' Carriers (CsC) is enabled such the use of BGP for signaling purposes [RFC8277].

'ntp': Time synchronization may be needed in some VPNs such as infrastructure and management VPNs. This container is used to enable the NTP service [RFC5905].

'multicast': Specifies the multicast mode and other data nodes such as the address-family. Refer to Section 7.7.

7.6.6.2. QoS

'qos' container is used to define a set of QoS policies to apply on a given connection (Figure 24). A QoS policy may be a classification or an action policy. For example, a QoS action can be defined to rate limit inbound/outbound traffic of a given class of service.

```

...
+--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-action
|   |   +--rw rule* [id]
|   |   |   +--rw id string
|   |   |   +--rw target-class-id? string
|   |   |   +--rw inbound-rate-limit? decimal64
|   |   |   +--rw outbound-rate-limit? decimal64
|   +--rw qos-profile
|   |   +--rw qos-profile* [profile]
|   |   |   +--rw profile leafref
|   |   |   +--rw direction? identityref
...

```

Figure 24: Services Subtree Structure

QoS classification can be based on many criteria such as:

Layer 3: As shown in Figure 25, classification can be 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)
|   |   |   |   |   |   |   +--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
|   |   |   |   |   |   |   ...

```

Figure 25: QoS Subtree Structure (L3)

Layer 4: As discussed in [I-D.ietf-opsawg-vpn-common], any layer 4

protocol can be indicated in the 'protocol' data node under 'l3' (Figure 25), but only TCP and UDP specific match criteria are elaborated in this version as these protocols are widely used in the context of VPN services. Augmentations can be considered in the future to add other Layer 4 specific data nodes, if needed.

TCP or UDP-related match criteria can be specified in the L3NM as shown in Figure 26.

As discussed in [I-D.ietf-opsawg-vpn-common], some transport protocols use existing protocols (e.g., TCP or UDP) as substrate. The match criteria for such protocols may rely upon the 'protocol' under 'l3', TCP/UDP match criteria shown in Figure 26, part of the TCP/UDP payload, or a combination thereof. This version of the module does not support such advanced match criteria. Future revisions of the VPN common module or augmentations to the L3NM may consider adding match criteria based on the transport protocol payload (e.g., by means of a bitmask match).

```
+--rw qos {vpn-common:qos}?
|   +--rw qos-classification-policy
|   |   +--rw rule* [id]
|   |   |   +--rw id string
|   |   |   +--rw (match-type)?
|   |   |   |   +--:(match-flow)
|   |   |   |   |   +--rw (l3)?
|   |   |   |   |   |   ...
|   |   |   |   +--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)
```

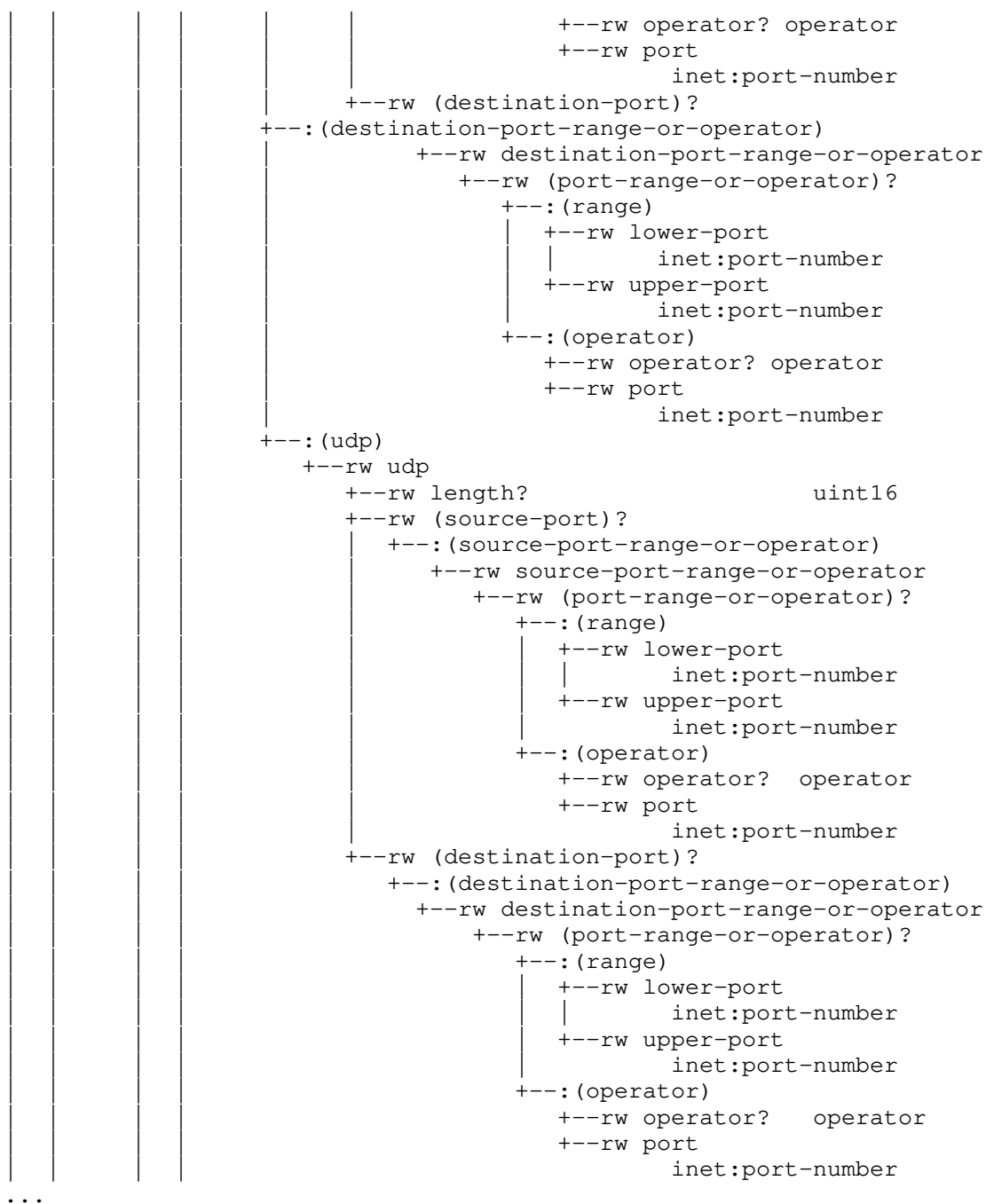


Figure 26: QoS Subtree Structure (L4)

Application match: Relies upon application-specific classification.

7.7. Multicast

Multicast may be enabled for a particular VPN at the VPN node and VPN network access levels (see Figure 27). Some data nodes (e.g., max-groups) can be controlled at various levels: VPN service, VPN node level, or VPN network access.

```

...
+--rw vpn-services
  +--rw vpn-service* [vpn-id]
    ...
    +--rw vpn-instance-profiles
      +--rw vpn-instance-profile* [profile-id]
        ....
        +--rw multicast {vpn-common:multicast}?
        ...
    +--rw vpn-nodes
      +--rw vpn-node* [vpn-node-id]
        ...
        +--rw active-vpn-instance-profiles
          +--rw vpn-instance-profile* [profile-id]
            ...
            +--rw multicast {vpn-common:multicast}?
            ...
        +--rw vpn-network-accesses
          +--rw vpn-network-access* [id]
            ...
            +--rw service
              ...
              +--rw multicast {vpn-common:multicast}?
              ...

```

Figure 27: Overall Multicast Subtree Structure

Multicast-related data nodes at the VPN instance profile level has the structure that is shown in Figure 30.

```

...
+--rw vpn-services
  +--rw vpn-service* [vpn-id]
    ...
    +--rw vpn-instance-profiles
      +--rw vpn-instance-profile* [profile-id]
        ....
        +--rw multicast {vpn-common:multicast}?
        +--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 igmp {vpn-common:igmp and vpn-common:ipv4}?
  +--rw static-group* [group-addr]
    +--rw group-addr
      | rt-types:ipv4-multicast-group-address
    +--rw source-addr?
      | rt-types:ipv4-multicast-source-address
    +--rw max-groups?                        uint32
    +--rw max-entries?                      uint32
    +--rw version?                          identityref
+--rw mld {vpn-common:mld and vpn-common:ipv6}?
  +--rw static-group* [group-addr]
    +--rw group-addr
      | rt-types:ipv6-multicast-group-address
    +--rw source-addr?
      | rt-types:ipv6-multicast-source-address
    +--rw max-groups?                      uint32
    +--rw max-entries?                    uint32
    +--rw version?                        identityref
+--rw pim {vpn-common:pim}?
  +--rw hello-interval?                    rt-types:timer-value-seconds16
  +--rw dr-priority?                       uint32

```

...

Figure 28: Multicast Subtree Structure (VPN Instance Profile Level)

The model supports a single type of tree per VPN access ('tree-flavor'): Any-Source Multicast (ASM), Source-Specific Multicast (SSM), or bidirectional.

When ASM is used, the model supports the configuration of Rendezvous Points (RPs). RP discovery may be 'static', 'bsr-rp', or 'auto-rp'. When set to 'static', RP to multicast grouping mappings 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.

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

When a particular VPN using ASM requires a more optimal traffic delivery (e.g., requested using [RFC8299]), '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. For example, anycast is one of the mechanisms to enhance RPs redundancy, resilience against failures, and to recover from failures quickly.

The same structure as the one depicted in Figure 30 is used when configuring multicast-related parameters at the VPN node level. When defined at the VPN node level (Figure 29), Internet Group Management Protocol (IGMP) [RFC1112][RFC2236][RFC3376], Multicast Listener Discovery (MLD) [RFC2710][RFC3810], and Protocol Independent Multicast (PIM) [RFC7761] parameters are applicable to all VPN network accesses of that VPN node unless corresponding nodes are overridden at the VPN network access level.

```

...
+--rw vpn-nodes
  +--rw vpn-node* [vpn-node-id]
    ...
    +--rw active-vpn-instance-profiles
      +--rw vpn-instance-profile* [profile-id]
        ...
        +--rw multicast {vpn-common:multicast}?
          +--rw tree-flavor* identityref
          +--rw rp
            ...
          +--rw igmp {vpn-common:igmp and vpn-common:ipv4}?
            ...
          +--rw mld {vpn-common:mld and vpn-common:ipv6}?
            ...
          +--rw pim {vpn-common:pim}?
            ...

```

Figure 29: Multicast Subtree Structure (VPN Node Level)

Multicast-related data nodes at the VPN network access level are shown in Figure 30. The values configured at the VPN network access level override the values configured for the corresponding data nodes in other levels.

```

...
+--rw vpn-network-accesses
  +--rw vpn-network-access* [id]
    ...
    +--rw service
      ...
      +--rw multicast {vpn-common:multicast}?
        +--rw access-type? enumeration
        +--rw address-family? identityref
        +--rw protocol-type? enumeration
        +--rw remote-source? boolean
        +--rw igmp {vpn-common:igmp}?
          +--rw static-group* [group-addr]
            +--rw group-addr
              rt-types:ipv4-multicast-group-address
            +--rw source-addr?
              rt-types:ipv4-multicast-source-address
          +--rw max-groups? uint32
          +--rw max-entries? uint32
          +--rw max-group-sources? uint32
          +--rw version? identityref
          +--rw status
            +--rw admin-status

```

```

|         |         +---rw status?          identityref
|         |         +---rw last-change?     yang:date-and-time
|         +---ro oper-status
|         |         +---ro status?          identityref
|         |         +---ro last-change?     yang:date-and-time
+---rw mld {vpn-common:mld}?
|   +---rw static-group* [group-addr]
|   |   +---rw group-addr
|   |   |       rt-types:ipv6-multicast-group-address
|   |   +---rw source-addr?
|   |   |       rt-types:ipv6-multicast-source-address
+---rw max-groups?          uint32
+---rw max-entries?         uint32
+---rw max-group-sources?   uint32
+---rw version?             identityref
+---rw status
|   +---rw admin-status
|   |   +---rw status?          identityref
|   |   +---rw last-change?     yang:date-and-time
|   +---ro oper-status
|   |   +---ro status?          identityref
|   |   +---ro last-change?     yang:date-and-time
+---rw pim {vpn-common:pim}?
|   +---rw hello-interval?   rt-types:timer-value-seconds16
|   +---rw dr-priority?      uint32
|   +---rw status
|   |   +---rw admin-status
|   |   |       +---rw status?          identityref
|   |   |       +---rw last-change?     yang:date-and-time
|   |   +---ro oper-status
|   |   |       +---ro status?          identityref
|   |   |       +---ro last-change?     yang:date-and-time

```

Figure 30: Multicast Subtree Structure (VPN Network Access Level)

8. L3NM YANG Module

This module uses types defined in [RFC6991] and [RFC8343]. It also uses groupings defined in [RFC8519], [RFC8177], and [RFC8294].

```

<CODE BEGINS> file "ietf-l3vpn-ntw@2021-09-28.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
      "RFC 6991: Common YANG Data Types, Section 4";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types, Section 3";
  }
  import ietf-key-chain {
    prefix key-chain;
    reference
      "RFC 8177: YANG Key Chain.";
  }
  import ietf-routing-types {
    prefix rt-types;
    reference
      "RFC 8294: Common YANG Data Types for the Routing Area";
  }
  import ietf-interfaces {
    prefix if;
    reference
      "RFC 8343: A YANG Data Model for Interface Management";
  }

  organization
    "IETF OPSAWG (Operations and Management Area Working Group)";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/opsawg/>
     WG List: <mailto:opsawg@ietf.org>

    Author:   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."
```

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

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

```
revision 2021-09-28 {
  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)";
}

/* Identities */

identity address-allocation-type {
  description
    "Base identity for address allocation type in the
    Provider Edge (PE)-Customer Edge (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 {
        if-feature "vpn-common:ipv6";
        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 4862: IPv6 Stateless Address Autoconfiguration";
    }

    identity static-address {
        base address-allocation-type;
        description
            "The Provider's network provides static IP addressing to the
            customer.";
    }

    identity slaac {
        if-feature "vpn-common:ipv6";
        base address-allocation-type;
        description
            "The Provider's network uses IPv6 SLAAC to provide addressing
            to the customer.";
        reference
            "RFC 4862: IPv6 Stateless Address Autoconfiguration";
    }

    identity local-defined-next-hop {
        description
            "Base identity of local defined next-hops.";
    }

    identity discard {
        base local-defined-next-hop;
        description
            "Indicates an action to discard traffic for the
            corresponding destination.
            For example, this can be used to blackhole traffic.";
    }

    identity local-link {
        base local-defined-next-hop;
        description
            "Treat traffic towards addresses within the specified next-hop
            prefix as though they are connected to a local link.";
```

```
}

identity l2-tunnel-type {
  description
    "Base identity for layer-2 tunnel selection under the VPN
    network access.";
}

identity pseudowire {
  base l2-tunnel-type;
  description
    "Pseudowire tunnel termination in the VPN network access.";
}

identity vpls {
  base l2-tunnel-type;
  description
    "Virtual Private LAN Service (VPLS) tunnel termination in
    the VPN network access.";
}

identity vxlan {
  base l2-tunnel-type;
  description
    "Virtual eXtensible Local Area Network (VXLAN) tunnel
    termination in the VPN network access.";
}

/* Typedefs */

typedef predefined-next-hop {
  type identityref {
    base local-defined-next-hop;
  }
  description
    "Pre-defined next-hop designation for locally generated routes.";
}

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.";
}

/* Groupings */
```

```
grouping vpn-instance-profile {
  description
    "Grouping for data nodes that may be factorized
    among many levels of the model. The grouping can
    be used to define generic profiles at the VPN service
    level and then referenced at the VPN node and VPN
    network access levels.";
  leaf local-as {
    if-feature "vpn-common:rtg-bgp";
    type inet:as-number;
    description
      "Provider's Autonomous System (AS) number. Used if the
      customer requests BGP routing.";
  }
  uses vpn-common:route-distinguisher;
  list address-family {
    key "address-family";
    description
      "Set of per-address family parameters.";
    leaf address-family {
      type identityref {
        base vpn-common:address-family;
      }
      description
        "Indicates the address family (IPv4 and/or IPv6).";
    }
    container vpn-targets {
      description
        "Set of route targets to match for import and export routes
        to/from VRF.";
      uses vpn-common:vpn-route-targets;
    }
    list maximum-routes {
      key "protocol";
      description
        "Defines the maximum number of routes for the VRF.";
      leaf protocol {
        type identityref {
          base vpn-common:routing-protocol-type;
        }
        description
          "Indicates the routing protocol. 'any' value can
          be used to identify a limit that will apply for
          each active routing protocol.";
      }
      leaf maximum-routes {
        type uint32;
        description

```

```
        "Indicates the maximum number of prefixes that the
        VRF can accept for this address family and protocol.";
    }
}
}
container multicast {
    if-feature "vpn-common:multicast";
    description
        "Global multicast parameters.";
    leaf tree-flavor {
        type identityref {
            base vpn-common:multicast-tree-type;
        }
        description
            "Type of the multicast tree to be used.";
    }
}
container rp {
    description
        "Rendezvous Point (RP) parameters.";
    container rp-group-mappings {
        description
            "RP-to-group mappings parameters.";
        list rp-group-mapping {
            key "id";
            description
                "List of RP-to-group mappings.";
            leaf id {
                type uint16;
                description
                    "Unique identifier for the mapping.";
            }
        }
    }
    container provider-managed {
        description
            "Parameters for a provider-managed RP.";
        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 set to true, it indicates that a redundancy
            mechanism for the RP is required.";
    }
}
```

```
}
leaf optimal-traffic-delivery {
  type boolean;
  default "false";
  description
    "If set to true, the service provider (SP) must
     ensure that the 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 both RP redundancy and
       delivery through optimal path are
       activated.";
  }
  description
    "PIM Anycast-RP parameters.";
  leaf local-address {
    type inet:ip-address;
    description
      "IP local address for PIM RP. Usually, it
       corresponds to the Router ID or the
       primary address.";
  }
  leaf-list rp-set-address {
    type inet:ip-address;
    description
      "Specifies the IP address of other RP routers
       that share the same RP IP address.";
  }
}
}
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 {
```

```

description
    "Multicast groups associated with the RP.";
list group {
    key "id";
    description
        "List of multicast groups.";
    leaf id {
        type uint16;
        description
            "Identifier for the group.";
    }
    choice group-format {
        mandatory true;
        description
            "Choice for multicast group format.";
        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.";
            }
        }
    }
}
}
}
}
container rp-discovery {
    description
        "RP discovery parameters.";
    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.";
    }
    description
      "Container for the customer Bootstrap Router (BSR)
        candidate's addresses.";
    leaf-list bsr-candidate-address {
      type inet:ip-address;
      description
        "Specifies the address of candidate BSR.";
    }
  }
}

container igmp {
  if-feature "vpn-common:igmp and vpn-common:ipv4";
  description
    "Includes IGMP-related parameters.";
  list static-group {
    key "group-addr";
    description
      "Multicast static source/group associated to the
        IGMP session.";
    leaf group-addr {
      type rt-types:ipv4-multicast-group-address;
      description
        "Multicast group IPv4 address.";
    }
    leaf source-addr {
      type rt-types:ipv4-multicast-source-address;
      description
        "Multicast source IPv4 address.";
    }
  }
  leaf max-groups {
    type uint32;
    description
      "Indicates the maximum number of groups.";
  }
  leaf max-entries {
    type uint32;
```

```
        description
            "Indicates the maximum number of IGMP entries.";
    }
    leaf version {
        type identityref {
            base vpn-common:igmp-version;
        }
        default "vpn-common:igmpv2";
        description
            "Indicates the IGMP version.";
        reference
            "RFC 1112: Host Extensions for IP Multicasting
             RFC 2236: Internet Group Management Protocol, Version 2
             RFC 3376: Internet Group Management Protocol, Version 3";
    }
}
container mld {
    if-feature "vpn-common:mld and vpn-common:ipv6";
    description
        "Includes MLD-related parameters.";
    list static-group {
        key "group-addr";
        description
            "Multicast static source/group associated with the
             MLD session.";
        leaf group-addr {
            type rt-types:ipv6-multicast-group-address;
            description
                "Multicast group IPv6 address.";
        }
        leaf source-addr {
            type rt-types:ipv6-multicast-source-address;
            description
                "Multicast source IPv6 address.";
        }
    }
}
leaf max-groups {
    type uint32;
    description
        "Indicates the maximum number of groups.";
}
leaf max-entries {
    type uint32;
    description
        "Indicates the maximum number of MLD entries.";
}
leaf version {
    type identityref {
```

```
        base vpn-common:mld-version;
    }
    default "vpn-common:mldv2";
    description
        "Indicates the MLD protocol version.";
    reference
        "RFC 2710: Multicast Listener Discovery (MLD) for IPv6
        RFC 3810: Multicast Listener Discovery Version 2 (MLDv2)
        for IPv6";
    }
}
container pim {
    if-feature "vpn-common:pim";
    description
        "Only applies when protocol type is PIM.";
    leaf hello-interval {
        type rt-types:timer-value-seconds16;
        default "30";
        description
            "PIM hello-messages interval. If set to
            'infinity' or 'not-set', no periodic
            Hello messages are sent.";
        reference
            "RFC 7761: Protocol Independent Multicast - Sparse
            Mode (PIM-SM): Protocol Specification (Revised),
            Section 4.11";
    }
    leaf dr-priority {
        type uint32;
        default "1";
        description
            "Indicates the preference in the Designated Router (DR)
            election process. A larger value has a higher
            priority over a smaller value.";
        reference
            "RFC 7761: Protocol Independent Multicast - Sparse
            Mode (PIM-SM): Protocol Specification (Revised),
            Section 4.3.2";
    }
}
}
}

/* Main Blocks */
/* Main l3vpn-ntw */

container l3vpn-ntw {
    description
```

```
    "Main container for L3VPN services management.";
  container vpn-profiles {
    description
      "Contains a set of valid VPN profiles to reference in the VPN
       service.";
    uses vpn-common:vpn-profile-cfg;
  }
  container vpn-services {
    description
      "Container for the VPN services.";
    list vpn-service {
      key "vpn-id";
      description
        "List of VPN services.";
      uses vpn-common:vpn-description;
      leaf parent-service-id {
        type vpn-common:vpn-id;
        description
          "Pointer to the parent service, if any.
           A parent service can be an L3SM, a slice request, a VPN+
           service, etc.";
      }
      leaf vpn-type {
        type identityref {
          base vpn-common:service-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.";
      }
    }
    uses vpn-common:service-status;
    container vpn-instance-profiles {
      description
        "Container for a list of VPN instance profiles.";
      list vpn-instance-profile {
        key "profile-id";
        description
          "List of VPN instance profiles.";
        leaf profile-id {
          type string;
          description

```

```
        "VPN instance profile identifier.";
    }
    leaf role {
        type identityref {
            base vpn-common:role;
        }
        default "vpn-common:any-to-any-role";
        description
            "Role of the VPN node in the VPN.";
    }
    uses vpn-instance-profile;
}
}
container underlay-transport {
    description
        "Container for underlay transport.";
    uses vpn-common:underlay-transport;
}
container external-connectivity {
    if-feature "vpn-common:external-connectivity";
    description
        "Container for external connectivity.";
    choice profile {
        description
            "Choice for the external connectivity profile.";
        case profile {
            leaf profile-name {
                type leafref {
                    path "/l3vpn-ntw/vpn-profiles"
                      + "/valid-provider-identifiers"
                      + "/external-connectivity-identifier/id";
                }
                description
                    "Name of the service provider's profile to be applied
                     at the VPN service level.";
            }
        }
    }
}
}
container vpn-nodes {
    description
        "Container for VPN nodes.";
    list vpn-node {
        key "vpn-node-id";
        description
            "Includes a list of VPN nodes.";
        leaf vpn-node-id {
            type vpn-common:vpn-id;
        }
    }
}
```

```
        description
            "An identifier of the VPN node.";
    }
    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 local-as {
        if-feature "vpn-common:rtg-bgp";
        type inet:as-number;
        description
            "Provider's AS number in case the customer requests BGP
            routing.";
    }
    leaf router-id {
        type rt-types:router-id;
        description
            "A 32-bit number in the dotted-quad format that is used
            to uniquely identify a node within an autonomous
            system. This identifier is used for both IPv4 and
            IPv6.";
    }
    container active-vpn-instance-profiles {
        description
            "Container for active VPN instance profiles.";
        list vpn-instance-profile {
            key "profile-id";
            description
                "Includes a list of active VPN instance profiles.";
            leaf profile-id {
                type leafref {
                    path "/l3vpn-ntw/vpn-services/vpn-service"
                        + "/vpn-instance-profiles/vpn-instance-profile"
                        + "/profile-id";
                }
            }
            description
                "Node's active VPN instance profile.";
        }
        list router-id {
            key "address-family";
            description
```

```
        "Router-id per address family.";
    leaf address-family {
        type identityref {
            base vpn-common:address-family;
        }
        description
            "Indicates the address family for which the
             Router-ID applies.";
    }
    leaf router-id {
        type inet:ip-address;
        description
            "The router-id information can be an IPv4 or IPv6
             address. This can be used, for example, to
             configure an IPv6 address as a router-id
             when such capability is supported by underlay
             routers. In such case, the configured value
             overrides the generic one defined at the VPN
             node level.";
    }
    }
    uses vpn-instance-profile;
}
container msdp {
    if-feature "msdp";
    description
        "Includes MSDP-related parameters.";
    leaf peer {
        type inet:ipv4-address;
        description
            "Indicates the IPv4 address of the MSDP peer.";
    }
    leaf local-address {
        type inet:ipv4-address;
        description
            "Indicates the IPv4 address of the local end.
             This local address must be configured on
             the node.";
    }
    uses vpn-common:service-status;
}
uses vpn-common:vpn-components-group;
uses vpn-common:service-status;
container vpn-network-accesses {
    description
        "List of network accesses.";
    list vpn-network-access {
```

```
key "id";
description
  "List of network accesses.";
leaf id {
  type vpn-common:vpn-id;
  description
    "Identifier for the network access.";
}
leaf interface-id {
  type string;
  description
    "Identifier for the physical or logical
    interface.
    The identification of the sub-interface
    is provided at the connection and/or IP
    connection levels.";
}
leaf description {
  type string;
  description
    "Textual description of the network access.";
}
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.";
}
leaf vpn-instance-profile {
  type leafref {
    path "/l3vpn-ntw/vpn-services/vpn-service/vpn-nodes"
      + "/vpn-node/active-vpn-instance-profiles"
      + "/vpn-instance-profile/profile-id";
  }
  description
    "An identifier of an active VPN instance profile.";
}
uses vpn-common:service-status;
container connection {
  description
    "Defines layer 2 protocols and parameters that are
    required to enable connectivity between the PE
    and the CE.";
  container encapsulation {
    description
```

```
    "Container for layer 2 encapsulation.";
  leaf type {
    type identityref {
      base vpn-common:encapsulation-type;
    }
    default "vpn-common:priority-tagged";
    description
      "Encapsulation type. By default, the type of
       the tagged interface is 'priority-tagged'.";
  }
  container dot1q {
    when "derived-from-or-self(..../type, "
      + "'vpn-common:dot1q')" {
      description
        "Only applies when the type of the
         tagged interface is 'dot1q'.";
    }
    description
      "Tagged interface.";
    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 {
        range "1..4094";
      }
      description
        "VLAN identifier.";
    }
  }
}
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'.";
  }
  description
    "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'.";
  }
}
container qinq {
  when "derived-from-or-self(..../type, "
    + "'vpn-common:qinq')" {
    description
      "Only applies when the type of the tagged
      interface is QinQ.";
  }
  description
    "Includes QinQ parameters.";
  leaf tag-type {
    type identityref {
      base vpn-common:tag-type;
    }
    default "vpn-common:s-c-vlan";
    description
      "Tag type. By default, the tag type is
      'c-s-vlan'.";
  }
  leaf svlan-id {
    type uint16;
    mandatory true;
    description
      "S-VLAN identifier.";
  }
  leaf cvlan-id {
    type uint16;
    mandatory true;
    description
      "C-VLAN identifier.";
  }
}
}
choice l2-service {
  description
    "The layer 2 connectivity service can be
    provided by indicating a pointer to an L2VPN or
    by specifying a layer 2 tunnel service.";
  container l2-tunnel-service {
    description
      "Defines a layer 2 tunnel termination.
      It is only applicable when a tunnel is
```

```
        required. The supported values are:
        pseudowire, VPLS, and VXLAN. Other
        values may be defined, if needed.";
leaf type {
  type identityref {
    base l2-tunnel-type;
  }
  description
    "Selects the tunnel termination option for
    each vpn-network-access.";
}
container pseudowire {
  when "derived-from-or-self(..../type, "
    + "'pseudowire')" {
    description
      "Only applies when the type of the layer 2
      service type is pseudowire .";
  }
  description
    "Includes pseudowire termination parameters.";
  leaf vcid {
    type uint32;
    description
      "Indicates a PW or VC identifier.";
  }
  leaf far-end {
    type union {
      type uint32;
      type inet:ip-address;
    }
    description
      "Neighbor reference.";
    reference
      "RFC 8077: Pseudowire Setup and Maintenance
      Using the Label Distribution
      Protocol (LDP), Section 6.1";
  }
}
container vpls {
  when "derived-from-or-self(..../type, "
    + "'vpls')" {
    description
      "Only applies when the type of the layer 2
      service type is VPLS.";
  }
  description
    "VPLS termination parameters.";
  leaf vcid {
```

```
        type uint32;
        description
            "VC Identifier.";
    }
    leaf-list far-end {
        type union {
            type uint32;
            type inet:ip-address;
        }
        description
            "Neighbor reference.";
    }
}
container vxlan {
    when "derived-from-or-self(..../type, "
        + "'vxlan')" {
        description
            "Only applies when the type of the layer 2
            service type is VXLAN.";
    }
    description
        "VXLAN termination parameters.";
    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'.";
    }
    leaf-list peer-ip-address {
        type inet:ip-address;
        description
            "List of peer's IP addresses.";
    }
}
}
case l2vpn {
    leaf l2vpn-id {
        type vpn-common:vpn-id;
    }
}
```

```
        description
          "Indicates the L2VPN service associated with
          an Integrated Routing and Bridging (IRB)
          interface.";
      }
    }
  leaf l2-termination-point {
    type string;
    description
      "Specifies a reference to a local layer 2
      termination point such as a layer 2
      sub-interface.";
  }
  leaf local-bridge-reference {
    type string;
    description
      "Specifies a local bridge reference to
      accommodate, for example, implementations
      that require internal bridging.
      A reference may be a local bridge domain.";
  }
  leaf bearer-reference {
    if-feature "vpn-common:bearer-reference";
    type string;
    description
      "This is an internal reference for the service
      provider to identify the bearer associated
      with this VPN.";
  }
  container lag-interface {
    if-feature "vpn-common:lag-interface";
    description
      "Container of LAG interface attributes
      configuration.";
    leaf lag-interface-id {
      type string;
      description
        "LAG interface identifier.";
    }
  }
  container member-link-list {
    description
      "Container of Member link list.";
    list member-link {
      key "name";
      description
        "Member link.";
      leaf name {
```

```
        type string;
        description
            "Member link name.";
    }
}
}
}
}
container ip-connection {
    description
        "Defines IP connection parameters.";
    leaf l3-termination-point {
        type string;
        description
            "Specifies a reference to a local layer 3
            termination point such as a bridge domain
            interface.";
    }
    container ipv4 {
        if-feature "vpn-common:ipv4";
        description
            "IPv4-specific parameters.";
        leaf local-address {
            type inet:ipv4-address;
            description
                "The IP address used at the provider's
                interface.";
        }
        leaf prefix-length {
            type uint8 {
                range "0..32";
            }
            description
                "Subnet prefix length expressed in bits.
                It is applied to both local and customer
                addresses.";
        }
        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 to the peer site.

If there is no value for the address allocation type, then IPv4 addressing is not enabled.";

```
}
choice allocation-type {
  description
    "Choice of the IPv4 address allocation.";
  case provider-dhcp {
    description
      "DHCP allocated addresses related
      parameters. IP addresses are allocated
      by DHCP that is operated by the provider";
    leaf dhcp-service-type {
      type enumeration {
        enum server {
          description
            "Local DHCP server.";
        }
        enum relay {
          description
            "Local DHCP relay. DHCP requests are
            relayed to a provider's server.";
        }
      }
    }
    description
      "Indicates the type of DHCP service to
      be enabled on this access.";
  }
  choice service-type {
    description
      "Choice based on the DHCP service type.";
    case relay {
      description
        "Container for list of provider's DHCP
        servers (i.e., dhcp-service-type is set
        to relay).";
      leaf-list server-ip-address {
        type inet:ipv4-address;
        description
          "IPv4 addresses of the provider's DHCP
          server to use by the local DHCP
          relay.";
      }
    }
    case server {
```

```
description
  "A choice about how addresses are assigned
  when a local DHCP server is enabled.";
choice address-assign {
  default "number";
  description
    "Choice for how IPv4 addresses are
    assigned.";
  case number {
    leaf number-of-dynamic-address {
      type uint16;
      default "1";
      description
        "Specifies the number of IP
        addresses to be assigned to the
        customer on this access.";
    }
  }
  case explicit {
    container customer-addresses {
      description
        "Container for customer
        addresses to be allocated
        using DHCP.";
      list address-pool {
        key "pool-id";
        description
          "Describes IP addresses to be
          allocated by DHCP.

          When only start-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.";
        leaf pool-id {
          type string;
          description
            "A pool identifier for the
            address range from start-
            address to end-address.";
        }
        leaf start-address {
          type inet:ipv4-address;
          mandatory true;
```

```
        description
            "Indicates the first address
            in the pool.";
    }
    leaf end-address {
        type inet:ipv4-address;
        description
            "Indicates the last address
            in the pool.";
    }
    }
    }
    }
    }
}
}
case dhcp-relay {
    description
        "DHCP relay is provided by the operator.";
    container customer-dhcp-servers {
        description
            "Container for a list of customer's DHCP
            servers.";
        leaf-list server-ip-address {
            type inet:ipv4-address;
            description
                "IPv4 addresses of the customer's DHCP
                server.";
        }
    }
}
case static-addresses {
    description
        "Lists the IPv4 addresses that are used.";
    leaf primary-address {
        type leafref {
            path "../address/address-id";
        }
        description
            "Primary address of the connection.";
    }
    list address {
        key "address-id";
        description
            "Lists the IPv4 addresses that are used.";
        leaf address-id {
            type string;
```

```
        description
            "An identifier of the static IPv4
            address.";
    }
    leaf customer-address {
        type inet:ipv4-address;
        description
            "IPv4 address at the customer side.";
    }
}
}
}
container ipv6 {
    if-feature "vpn-common:ipv6";
    description
        "IPv6-specific parameters.";
    leaf local-address {
        type inet:ipv6-address;
        description
            "IPv6 address of the provider side.";
    }
    leaf prefix-length {
        type uint8 {
            range "0..128";
        }
        description
            "Subnet prefix length expressed in bits.
            It is applied to both local and customer
            addresses.";
    }
    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 addressing is
            disabled.";
    }
    choice allocation-type {
        description
            "A choice based on the IPv6 allocation type.";
        container 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 DHCPv6 provided by the
    operator.";
}
description
    "DHCPv6 allocated addresses related
    parameters.";
leaf dhcp-service-type {
    type enumeration {
        enum server {
            description
                "Local DHCPv6 server.";
        }
        enum relay {
            description
                "DHCPv6 relay.";
        }
    }
}
description
    "Indicates the type of the DHCPv6 service to
    be enabled on this access.";
}
choice service-type {
    description
        "Choice based on the DHCPv6 service type.";
    case relay {
        leaf-list server-ip-address {
            type inet:ipv6-address;
            description
                "IPv6 addresses of the provider's
                DHCPv6 server.";
        }
    }
    case server {
        choice address-assign {
            default "number";
            description
                "Choice about how IPv6 prefixes are
                assigned by the DHCPv6 server.";
            case number {
                leaf number-of-dynamic-address {
                    type uint16;
                    default "1";
                    description
                        "Describes the number of IPv6
                        prefixes that are allocated to
```

```

        the customer on this access.";
    }
}
case explicit {
    container customer-addresses {
        description
            "Container for customer IPv6
            addresses allocated by DHCPv6.";
        list address-pool {
            key "pool-id";
            description
                "Describes IPv6 addresses
                allocated by DHCPv6.

                When only start-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.";
            leaf pool-id {
                type string;
                description
                    "Pool identifier for the address
                    range from identified by start-
                    address and end-address.";
            }
            leaf start-address {
                type inet:ipv6-address;
                mandatory true;
                description
                    "Indicates the first address.";
            }
            leaf end-address {
                type inet:ipv6-address;
                description
                    "Indicates the last address.";
            }
        }
    }
}
}
}
}
}
}
}
case dhcp-relay {

```

```

description
  "DHCPv6 relay provided by the operator.";
container customer-dhcp-servers {
  description
    "Container for a list of customer DHCP
    servers.";
  leaf-list server-ip-address {
    type inet:ipv6-address;
    description
      "Contains the IP addresses of the customer
      DHCPv6 server.";
  }
}
}
case static-addresses {
  description
    "IPv6-specific parameters for static
    allocation.";
  leaf primary-address {
    type leafref {
      path "../address/address-id";
    }
    description
      "Principal address of the connection";
  }
  list address {
    key "address-id";
    description
      "Describes IPv6 addresses that are used.";
    leaf address-id {
      type string;
      description
        "An identifier of an IPv6 address.";
    }
    leaf customer-address {
      type inet:ipv6-address;
      description
        "An IPv6 address of the customer side.";
    }
  }
}
}
}
}
container routing-protocols {
  description
    "Defines routing protocols.";
  list routing-protocol {

```

```
key "id";
description
  "List of routing protocols used on
   the CE/PE link. This list can be augmented.";
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";
  description
    "Routing profiles.";
  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.";
  }
}
container static {
  when "derived-from-or-self(..../type, "
    + "'vpn-common:static-routing') " {
    description
      "Only applies when protocol is static.";
  }
  description
    "Configuration specific to static routing.";
  container cascaded-lan-prefixes {
    description
      "LAN prefixes from the customer.";
  }
}
```

```
list ipv4-lan-prefixes {
  if-feature "vpn-common:ipv4";
  key "lan next-hop";
  description
    "List of LAN prefixes for the site.";
  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 union {
      type inet:ip-address;
      type predefined-next-hop;
    }
    description
      "The next-hop that is to be used
      for the static route. This may be
      specified as an IP address or a
      pre-defined next-hop type (e.g.,
      discard or local-link).";
  }
  leaf bfd-enable {
    if-feature "vpn-common:bfd";
    type boolean;
    description
      "Enables BFD.";
  }
  leaf metric {
    type uint32;
    description
      "Indicates the metric associated with
      the static route.";
  }
  leaf preference {
    type uint32;
    description
      "Indicates the preference of the static
      routes.";
  }
  uses vpn-common:service-status;
}
```

```
list ipv6-lan-prefixes {
  if-feature "vpn-common:ipv6";
  key "lan next-hop";
  description
    "List of LAN prefixes for the site.";
  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 union {
      type inet:ip-address;
      type predefined-next-hop;
    }
    description
      "The next-hop that is to be used for the
      static route. This may be specified as
      an IP address or a pre-defined next-hop
      type (e.g., discard or local-link).";
  }
  leaf bfd-enable {
    if-feature "vpn-common:bfd";
    type boolean;
    description
      "Enables BFD.";
  }
  leaf metric {
    type uint32;
    description
      "Indicates the metric associated with
      the static route.";
  }
  leaf preference {
    type uint32;
    description
      "Indicates the preference associated
      with the static route.";
  }
  uses vpn-common:service-status;
}
```

```
}
container bgp {
  when "derived-from-or-self(..../type, "
    + "'vpn-common:bgp-routing')" {
    description
      "Only applies when protocol is BGP.";
  }
  description
    "BGP-specific configuration.";
  leaf description {
    type string;
    description
      "Includes a description of the BGP session.

      This description is meant to be used for
      diagnosis purposes. The semantic of the
      description is local to an
      implementation.";
  }
  leaf local-as {
    type inet:as-number;
    description
      "Indicates a local AS Number (ASN) if a
      distinct ASN than the one configured at
      the VPN node level is needed.";
  }
  leaf peer-as {
    type inet:as-number;
    mandatory true;
    description
      "Indicates the customer's ASN when
      the customer requests BGP routing.";
  }
  leaf address-family {
    type identityref {
      base vpn-common:address-family;
    }
    description
      "This node contains the address families to be
      activated. Dual-stack means that both IPv4
      and IPv6 will be activated.";
  }
  leaf local-address {
    type union {
      type inet:ip-address;
      type if:interface-ref;
    }
    description
```

```
        "Set the local IP address to use for the BGP
        transport session. This may be expressed as
        either an IP address or a reference to an
        interface.";
    }
    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.";
    }
    leaf as-override {
        type boolean;
        default "false";
        description
            "Defines whether ASN override is enabled,
            i.e., replace the ASN of the customer
            specified in the AS_Path attribute with
            the local ASN.";
    }
    leaf allow-own-as {
        type uint8;
        default "0";
        description
            "Specifies the number of occurrences
            of the provider's ASN that can occur
            within the AS_PATH before it
            is rejected.";
    }
    leaf prepend-global-as {
        type boolean;
        default "false";
        description
            "In some situations, the ASN that is
            provided at the VPN node level may be
            distinct from the one configured at the
            VPN network access level. When such
            ASNs are provided, they are both
            prepended to the BGP route updates
            for this access. To disable that
```

```
        behavior, the prepend-global-as
        must be set to 'false'. In such a case,
        the ASN that is provided at
        the VPN node level is not prepended to
        the BGP route updates for this access.";
    }
    leaf send-default-route {
        type boolean;
        default "false";
        description
            "Defines whether default routes can be
            advertised to its peer. If set, the
            default routes are advertised to its
            peer.";
    }
    leaf site-of-origin {
        when "../address-family = 'vpn-common:ipv4' or "
            + "'vpn-common:dual-stack'" {
            description
                "Only applies if IPv4 is activated.";
        }
        type rt-types:route-origin;
        description
            "The Site of Origin attribute is encoded as
            a Route Origin Extended Community. It is
            meant to uniquely identify the set of routes
            learned from a site via a particular CE/PE
            connection and is used to prevent routing
            loops.";
        reference
            "RFC 4364: BGP/MPLS IP Virtual Private
            Networks (VPNs), Section 7";
    }
    leaf ipv6-site-of-origin {
        when "../address-family = 'vpn-common:ipv6' or "
            + "'vpn-common:dual-stack'" {
            description
                "Only applies if IPv6 is activated.";
        }
        type rt-types:ipv6-route-origin;
        description
            "IPv6 Route Origins are IPv6 Address Specific
            BGP Extended that are meant to the Site of
            Origin for VRF information.";
        reference
            "RFC 5701: IPv6 Address Specific BGP Extended
            Community Attribute";
    }
}
```

```
list redistribute-connected {
  key "address-family";
  description
    "Indicates the per-AF policy to follow
    for connected routes.";
  leaf address-family {
    type identityref {
      base vpn-common:address-family;
    }
    description
      "Indicates the address family.";
  }
  leaf enable {
    type boolean;
    description
      "Enables to redistribute connected
      routes.";
  }
}
container bgp-max-prefix {
  description
    "Controls the behavior when a prefix
    maximum is reached.";
  leaf max-prefix {
    type uint32;
    default "5000";
    description
      "Indicates the maximum number of BGP
      prefixes allowed in the BGP session.

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

      If the limit is exceeded, the action
      indicated in violate-action will be
      followed.";
    reference
      "RFC 4271: A Border Gateway Protocol 4
      (BGP-4), 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
                    "The BGP session restarts after
                    a time interval.";
            }
        }
        description
            "BGP neighbor max-prefix violate
            action.";
    }
    leaf restart-timer {
        type uint32;
        units "seconds";
        description
            "Time interval after which the BGP
            session will be reestablished.";
    }
}
container bgp-timers {
    description
        "Includes two BGP timers that can be
        customized when building a VPN service
        with BGP used as CE-PE routing
        protocol.";
    leaf keepalive {
        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 KEEPALIVE messages would be one third of the Hold Time interval.";

reference

"RFC 4271: A Border Gateway Protocol 4 (BGP-4), Section 4.4";

```
}
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
    "RFC 4271: A Border Gateway Protocol 4
     (BGP-4), Section 4.2";
}
}
container authentication {
  description
    "Container for BGP authentication
     parameters between a PE and a CE.";
  leaf enable {
    type boolean;
    default "false";
    description
      "Enables or disables authentication.";
  }
  container keying-material {
    when "../enable = 'true'";
    description
      "Container for describing how a BGP routing
       session is to be secured between a PE and
       a CE.";
```

```
choice option {
  description
    "Choice of authentication options.";
  case ao {
    description
      "Uses TCP-Authentication Option
      (TCP-AO).";
    reference
      "RFC 5925: The TCP Authentication
      Option.";
    leaf enable-ao {
      type boolean;
      description
        "Enables TCP-AO.";
    }
    leaf ao-keychain {
      type key-chain:key-chain-ref;
      description
        "Reference to the TCP-AO key chain.";
      reference
        "RFC 8177: YANG Key Chain.";
    }
  }
  case md5 {
    description
      "Uses MD5 to secure the session.";
    reference
      "RFC 4364: BGP/MPLS IP Virtual Private
      Networks (VPNs),
      Section 13.2";
    leaf md5-keychain {
      type key-chain:key-chain-ref;
      description
        "Reference to the MD5 key chain.";
      reference
        "RFC 8177: YANG Key Chain";
    }
  }
  case explicit {
    leaf key-id {
      type uint32;
      description
        "Key Identifier.";
    }
    leaf key {
      type string;
      description
        "BGP authentication key.
```

```

        This model only supports the subset
        of keys that are representable as
        ASCII strings.";
    }
    leaf crypto-algorithm {
        type identityref {
            base key-chain:crypto-algorithm;
        }
        description
            "Indicates the cryptographic algorithm
            associated with the key.";
    }
}
case ipsec {
    description
        "Specifies a reference to an IKE
        Security Association (SA).";
    leaf sa {
        type string;
        description
            "Indicates the administrator-assigned
            name of the SA.";
    }
}
}
}
}
uses vpn-common:service-status;
}
container ospf {
    when "derived-from-or-self(..type, "
        + "'vpn-common:ospf-routing')" {
        description
            "Only applies when protocol is OSPF.";
    }
    description
        "OSPF-specific configuration.";
    leaf address-family {
        type identityref {
            base vpn-common:address-family;
        }
        description
            "Indicates whether IPv4, IPv6, or
            both are to be activated.";
    }
    leaf area-id {
        type yang:dotted-quad;
        mandatory true;
    }
}
```

```
description
  "Area ID.";
reference
  "RFC 4577: OSPF as the Provider/Customer
    Edge Protocol for BGP/MPLS IP
    Virtual Private Networks
    (VPNs), Section 4.2.3
  RFC 6565: OSPFv3 as a Provider Edge to
    Customer Edge (PE-CE) Routing
    Protocol, Section 4.2";
}
leaf metric {
  type uint16;
  default "1";
  description
    "Metric of the PE-CE link. It is used
    in the routing state calculation and
    path selection.";
}
container sham-links {
  if-feature "vpn-common:rtg-ospf-sham-link";
  description
    "List of sham links.";
  reference
    "RFC 4577: OSPF as the Provider/Customer
      Edge Protocol for BGP/MPLS IP
      Virtual Private Networks
      (VPNs), Section 4.2.7
    RFC 6565: OSPFv3 as a Provider Edge to
      Customer Edge (PE-CE) Routing
      Protocol, Section 5";
  list sham-link {
    key "target-site";
    description
      "Creates a sham link with another site.";
    leaf target-site {
      type string;
      description
        "Target site for the sham link connection.
        The site is referred to by its
        identifier.";
    }
    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.";
    reference
        "RFC 4577: OSPF as the Provider/Customer
        Edge Protocol for BGP/MPLS IP
        Virtual Private Networks
        (VPNs), Section 4.2.7.3
        RFC 6565: OSPFv3 as a Provider Edge to
        Customer Edge (PE-CE) Routing
        Protocol, Section 5.2";
    }
}
leaf max-lsa {
    type uint32 {
        range "1..4294967294";
    }
    description
        "Maximum number of allowed LSAs OSPF.";
}
container authentication {
    description
        "Authentication configuration.";
    leaf enable {
        type boolean;
        default "false";
        description
            "Enables or disables authentication.";
    }
}
container keying-material {
    when "../enable = 'true'";
    description
        "Container for describing how an OSPF
        session is to be secured between a CE
        and a PE.";
    choice option {
        description
            "Options for OSPF authentication.";
        case auth-key-chain {
            leaf key-chain {
                type key-chain:key-chain-ref;
                description
                    "key-chain name.";
            }
        }
        case auth-key-explicit {
            leaf key-id {
                type uint32;
            }
        }
    }
}
```

```
        description
            "Key identifier.";
    }
    leaf key {
        type string;
        description
            "OSPF authentication key.
            This model only supports the subset
            of keys that are representable as
            ASCII strings.";
    }
    leaf crypto-algorithm {
        type identityref {
            base key-chain:crypto-algorithm;
        }
        description
            "Indicates the cryptographic algorithm
            associated with the key.";
    }
}
case ipsec {
    leaf sa {
        type string;
        description
            "Indicates the administrator-assigned
            name of the SA.";
        reference
            "RFC 4552: Authentication
            /Confidentiality for
            OSPFv3";
    }
}
}
}
}
uses vpn-common:service-status;
}
container isis {
    when "derived-from-or-self(..../type, "
        + "'vpn-common:isis-routing')" {
        description
            "Only applies when protocol is IS-IS.";
    }
    description
        "IS-IS specific configuration.";
    leaf address-family {
        type identityref {
            base vpn-common:address-family;
        }
    }
}
```

```
    }
    description
      "Indicates whether IPv4, IPv6, or both
       are to be activated.";
  }
  leaf area-address {
    type area-address;
    mandatory true;
    description
      "Area address.";
  }
  leaf level {
    type identityref {
      base vpn-common:isis-level;
    }
    description
      "Can be level-1, level-2, or level-1-2.";
  }
  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 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.";
  }
  container authentication {
    description
      "Authentication configuration.";
    leaf enable {
```

```
    type boolean;
    default "false";
    description
        "Enables or disables authentication.";
}
container keying-material {
    when "../enable = 'true'";
    description
        "Container for describing how an IS-IS
        session is to be secured between a CE
        and a PE.";
    choice option {
        description
            "Options for IS-IS authentication.";
        case auth-key-chain {
            leaf key-chain {
                type key-chain:key-chain-ref;
                description
                    "key-chain name.";
            }
        }
        case auth-key-explicit {
            leaf key-id {
                type uint32;
                description
                    "Key Identifier.";
            }
            leaf key {
                type string;
                description
                    "IS-IS authentication key.
                    This model only supports the subset
                    of keys that are representable as
                    ASCII strings.";
            }
            leaf crypto-algorithm {
                type identityref {
                    base key-chain:crypto-algorithm;
                }
                description
                    "Indicates the cryptographic algorithm
                    associated with the key.";
            }
        }
    }
}
}
}
uses vpn-common:service-status;
```

```
}
container rip {
  when "derived-from-or-self(..../type, "
    + "'vpn-common:rip-routing')" {
    description
      "Only applies when the protocol is RIP.
      For IPv4, the model assumes that RIP
      version 2 is used.";
  }
  description
    "Configuration specific to RIP routing.";
  leaf address-family {
    type identityref {
      base vpn-common:address-family;
    }
    description
      "Indicates whether IPv4, IPv6, or both
      address families are to be activated.";
  }
  container timers {
    description
      "Indicates the RIP timers.";
    reference
      "RFC 2453: RIP Version 2";
    leaf update-interval {
      type uint16 {
        range "1..32767";
      }
      units "seconds";
      default "30";
      description
        "Indicates the RIP update time.
        That is, the amount of time for which
        RIP updates are sent.";
    }
    leaf invalid-interval {
      type uint16 {
        range "1..32767";
      }
      units "seconds";
      default "180";
      description
        "Is the interval before a route is declared
        invalid after no updates are received.
        This value is at least three times
        the value for the update-interval
        argument.";
    }
  }
}
```

```
leaf holddown-interval {
  type uint16 {
    range "1..32767";
  }
  units "seconds";
  default "180";
  description
    "Specifies the interval before better routes
     are released.";
}
leaf flush-interval {
  type uint16 {
    range "1..32767";
  }
  units "seconds";
  default "240";
  description
    "Indicates the RIP flush timer. That is,
     the amount of time that must elapse before
     a route is removed from the routing
     table.";
}
}
leaf default-metric {
  type uint8 {
    range "0..16";
  }
  default "1";
  description
    "Sets the default metric.";
}
container authentication {
  description
    "Authentication configuration.";
  leaf enable {
    type boolean;
    default "false";
    description
      "Enables or disables authentication.";
  }
}
container keying-material {
  when "../enable = 'true'";
  description
    "Container for describing how a RIP
     session is to be secured between a CE
     and a PE.";
  choice option {
    description
```

```
        "Specifies the authentication scheme.";
    case auth-key-chain {
        leaf key-chain {
            type key-chain:key-chain-ref;
            description
                "key-chain name.";
        }
    }
    case auth-key-explicit {
        leaf key {
            type string;
            description
                "RIP authentication key.
                This model only supports the subset
                of keys that are representable as
                ASCII strings.";
        }
        leaf crypto-algorithm {
            type identityref {
                base key-chain:crypto-algorithm;
            }
            description
                "Indicates the cryptographic algorithm
                associated with the key.";
        }
    }
}
}
}
uses vpn-common:service-status;
}
container vrrp {
    when "derived-from-or-self(..../type, "
        + "'vpn-common:vrrp-routing')";
    description
        "Only applies when protocol is VRRP.";
}
description
    "Configuration specific to VRRP.";
reference
    "RFC 5798: Virtual Router Redundancy Protocol
    (VRRP) Version 3 for IPv4 and IPv6";
leaf address-family {
    type identityref {
        base vpn-common:address-family;
    }
    description
        "Indicates whether IPv4, IPv6, or both
```

```
        address families are to be enabled.";
    }
    leaf vrrp-group {
        type uint8 {
            range "1..255";
        }
        description
            "Includes the VRRP group identifier.";
    }
    leaf backup-peer {
        type inet:ip-address;
        description
            "Indicates the IP address of the peer.";
    }
    leaf-list virtual-ip-address {
        type inet:ip-address;
        description
            "Virtual IP addresses for a single VRRP
            group.";
        reference
            "RFC 5798: Virtual Router Redundancy Protocol
            (VRRP) Version 3 for IPv4 and
            IPv6, Sections 1.2 and 1.3";
    }
    leaf priority {
        type uint8 {
            range "1..254";
        }
        default "100";
        description
            "Sets the local priority of the VRRP
            speaker.";
    }
    leaf ping-reply {
        type boolean;
        default "false";
        description
            "Controls whether the VRRP speaker should
            answer to ping requests.";
    }
    uses vpn-common:service-status;
}
}
container oam {
    description
        "Defines the Operations, Administration,
        and Maintenance (OAM) mechanisms used."
```

```
BFD is set as a fault detection mechanism,
but other mechanisms can be defined in the
future.";
container bfd {
  if-feature "vpn-common:bfd";
  description
    "Container for BFD.";
  leaf session-type {
    type identityref {
      base vpn-common:bfd-session-type;
    }
    default "vpn-common:classic-bfd";
    description
      "Specifies the BFD session type.";
  }
  leaf desired-min-tx-interval {
    type uint32;
    units "microseconds";
    default "1000000";
    description
      "The minimum interval between transmission of
      BFD control packets that the operator
      desires.";
    reference
      "RFC 5880: Bidirectional Forwarding Detection
      (BFD), Section 6.8.7";
  }
  leaf required-min-rx-interval {
    type uint32;
    units "microseconds";
    default "1000000";
    description
      "The minimum interval between received BFD
      control packets that the PE should support.";
    reference
      "RFC 5880: Bidirectional Forwarding Detection
      (BFD), Section 6.8.7";
  }
  leaf local-multiplier {
    type uint8 {
      range "1..255";
    }
    default "3";
    description
      "Specifies the detection multiplier that is
      transmitted to a BFD peer.

      The detection interval for the receiving
```

```
        BFD peer is calculated by multiplying the value
        of the negotiated transmission interval by
        the received detection multiplier value.";
    reference
        "RFC 5880: Bidirectional Forwarding Detection
        (BFD), Section 6.8.7";
}
leaf holdtime {
    type uint32;
    units "milliseconds";
    description
        "Expected BFD holdtime.

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

        If the provider doesn't allow the
        customer to use this function,
        the fixed-value will not be set.";
    reference
        "RFC 5880: Bidirectional Forwarding Detection
        (BFD), Section 6.8.18";
}
leaf profile {
    type leafref {
        path "/l3vpn-ntw/vpn-profiles"
            + "/valid-provider-identifiers"
            + "/bfd-profile-identifier/id";
    }
    description
        "Well-known service provider profile name.

        The provider can propose some profiles
        to the customer, depending on the
        service level the customer wants to
        achieve.";
}
container authentication {
    presence "Enables BFD authentication";
    description
        "Parameters for BFD authentication.";
    leaf key-chain {
        type key-chain:key-chain-ref;
        description
            "Name of the key-chain.";
    }
}
```

```
    leaf meticulous {
      type boolean;
      description
        "Enables meticulous mode.";
      reference
        "RFC 5880: Bidirectional Forwarding
          Detection (BFD), Section 6.7";
    }
  }
  uses vpn-common:service-status;
}

container security {
  description
    "Site-specific security parameters.";
  container encryption {
    if-feature "vpn-common:encryption";
    description
      "Container for CE-PE security encryption.";
    leaf enabled {
      type boolean;
      default "false";
      description
        "If true, traffic encryption on the
          connection is required. Otherwise, it
          is disabled.";
    }
    leaf layer {
      when "../enabled = 'true'" {
        description
          "It is included only when encryption
            is enabled.";
      }
      type enumeration {
        enum layer2 {
          description
            "Encryption occurs at Layer 2.";
        }
        enum layer3 {
          description
            "Encryption occurs at Layer 3.
              For example, IPsec may be used when
              a customer requests Layer 3
              encryption.";
        }
      }
    }
    description
      "Indicates the layer on which encryption
```

```
        is applied.";
    }
}
container encryption-profile {
    when "../encryption/enabled = 'true'" {
        description
            "Indicates the layer on which encryption
            is enabled.";
    }
    description
        "Container for encryption profile.";
    choice profile {
        description
            "Choice for the encryption 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 service provider's profile
                    to be applied.";
            }
        }
        case customer-profile {
            leaf customer-key-chain {
                type key-chain:key-chain-ref;
                description
                    "Customer-supplied key chain.";
            }
        }
    }
}
container service {
    description
        "Service parameters of the attachment.";
    leaf inbound-bandwidth {
        if-feature "vpn-common:inbound-bw";
        type uint64;
        units "bps";
        description
            "From the customer site's perspective, the
            service inbound bandwidth of the connection
            or download bandwidth from the SP to
            the site. Note that the L3SM uses 'input-
```

```
        -bandwidth' to refer to the same concept.";
    }
    leaf outbound-bandwidth {
        if-feature "vpn-common:outbound-bw";
        type uint64;
        units "bps";
        description
            "From the customer site's perspective,
             the service outbound bandwidth of the
             connection or upload bandwidth from
             the site to the SP. Note that the L3SM uses
             'output-bandwidth' to refer to the same
             concept.";
    }
    leaf mtu {
        type uint32;
        units "bytes";
        description
            "MTU at service level. If the service is IP,
             it refers to the IP MTU. If Carriers'
             Carriers (CsC) is enabled, the requested MTU
             will refer to the MPLS maximum labeled packet
             size and not to the IP MTU.";
    }
    container qos {
        if-feature "vpn-common:qos";
        description
            "QoS configuration.";
        container qos-classification-policy {
            description
                "Configuration of the traffic classification
                 policy.";
            uses vpn-common:qos-classification-policy;
        }
        container qos-action {
            description
                "List of QoS action policies.";
            list rule {
                key "id";
                description
                    "List of QoS actions.";
                leaf id {
                    type string;
                    description
                        "An identifier of the QoS action rule.";
                }
                leaf target-class-id {
                    type string;
                }
            }
        }
    }
}
```

```
    description
      "Identification of the class of service.
       This identifier is internal to the
       administration.";
  }
  leaf inbound-rate-limit {
    type decimal64 {
      fraction-digits 5;
      range "0..100";
    }
    units "percent";
    description
      "Specifies whether/how to rate-limit the
       inbound traffic matching this QoS policy.
       It is expressed as a percent of the value
       that is indicated in 'input-bandwidth'.";
  }
  leaf outbound-rate-limit {
    type decimal64 {
      fraction-digits 5;
      range "0..100";
    }
    units "percent";
    description
      "Specifies whether/how to rate-limit the
       outbound traffic matching this QoS policy.
       It is expressed as a percent of the value
       that is indicated in 'output-bandwidth'.";
  }
}

container qos-profile {
  description
    "QoS profile configuration.";
  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.";
    }
  }
}
container carriers-carrier {
  if-feature "vpn-common:carriers-carrier";
  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 MPLS service using an
     IP VPN to carry its traffic).";
  leaf signaling-type {
    type enumeration {
      enum ldp {
        description
          "Use LDP as the signaling protocol
           between the PE and the CE. In this
           case, an IGP routing protocol must
           also be configured.";
      }
      enum bgp {
        description
          "Use BGP as the signaling 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 signaling type.";
  }
}
container ntp {
  description
```

```
    "Time synchronization may be needed in some
    VPNs such as infrastructure and Management
    VPNs. This container includes parameters to
    enable NTP service.";
  reference
    "RFC 5905: Network Time Protocol Version 4:
      Protocol and Algorithms
      Specification";
  leaf broadcast {
    type enumeration {
      enum client {
        description
          "The VPN node will listen to NTP broadcast
          messages on this VPN network access.";
      }
      enum server {
        description
          "The VPN node will behave as a broadcast
          server.";
      }
    }
    description
      "Indicates NTP broadcast mode to use for the
      VPN network access.";
  }
  container auth-profile {
    description
      "Pointer to a local profile.";
    leaf profile-id {
      type string;
      description
        "A pointer to a local authentication
        profile on the VPN node is provided.";
    }
  }
  uses vpn-common:service-status;
}
container multicast {
  if-feature "vpn-common:multicast";
  description
    "Multicast parameters for the network
    access.";
  leaf access-type {
    type enumeration {
      enum receiver-only {
        description
          "The peer site only has receivers.";
      }
    }
  }
}
```

```
enum source-only {
  description
    "The peer site only has sources.";
}
enum source-receiver {
  description
    "The peer site has both sources and
    receivers.";
}
}
default "source-receiver";
description
  "Type of multicast site.";
}
leaf address-family {
  type identityref {
    base vpn-common:address-family;
  }
  description
    "Indicates the 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
    "A remote multicast source is a source that is
    not on the same subnet as the
    vpn-network-access. When set to 'true', the
    multicast traffic from a remote source is
    accepted.";
}
container igmp {
  when "../protocol-type = 'host' and "
    + "../address-family = 'vpn-common:ipv4' or "
    + "'vpn-common:dual-stack'";
  if-feature "vpn-common:igmp";
  description
    "Includes IGMP-related parameters.";
  list static-group {
    key "group-addr";
    description
      "Multicast static source/group associated to
      IGMP session";
    leaf group-addr {
      type rt-types:ipv4-multicast-group-address;
      description
        "Multicast group IPv4 address.";
    }
    leaf source-addr {
      type rt-types:ipv4-multicast-source-address;
      description
        "Multicast source IPv4 address.";
    }
  }
}
leaf max-groups {
  type uint32;
  description
    "Indicates the maximum number of groups.";
}
leaf max-entries {
  type uint32;
  description
    "Indicates the maximum number of IGMP
    entries.";
}
```

```
leaf max-group-sources {
  type uint32;
  description
    "The maximum number of group sources.";
}
leaf version {
  type identityref {
    base vpn-common:igmp-version;
  }
  default "vpn-common:igmpv2";
  description
    "Version of the IGMP.";
}
uses vpn-common:service-status;
}
container mld {
  when "../protocol-type = 'host' and "
    + "../address-family = 'vpn-common:ipv6' or "
    + "'vpn-common:dual-stack'";
  if-feature "vpn-common:mld";
  description
    "Includes MLD-related parameters.";
  list static-group {
    key "group-addr";
    description
      "Multicast static source/group associated to
       the MLD session";
    leaf group-addr {
      type rt-types:ipv6-multicast-group-address;
      description
        "Multicast group IPv6 address.";
    }
    leaf source-addr {
      type rt-types:ipv6-multicast-source-address;
      description
        "Multicast source IPv6 address.";
    }
  }
}
leaf max-groups {
  type uint32;
  description
    "Indicates the maximum number of groups.";
}
leaf max-entries {
  type uint32;
  description
    "Indicates the maximum number of MLD
     entries.";
```

```
}
leaf max-group-sources {
  type uint32;
  description
    "The maximum number of group sources.";
}
leaf version {
  type identityref {
    base vpn-common:mld-version;
  }
  default "vpn-common:mldv2";
  description
    "Version of the MLD protocol.";
}
uses vpn-common:service-status;
}
container pim {
  when "../protocol-type = 'router'";
  if-feature "vpn-common:pim";
  description
    "Only applies when protocol type is PIM.";
  leaf hello-interval {
    type rt-types:timer-value-seconds16;
    default "30";
    description
      "PIM hello-messages interval. If set to
       'infinity' or 'not-set', no periodic
       Hello messages are sent.";
    reference
      "RFC 7761: Protocol Independent Multicast -
       Sparse Mode (PIM-SM): Protocol
       Specification (Revised),
       Section 4.11";
  }
  leaf dr-priority {
    type uint32;
    default "1";
    description
      "Indicates the preference in the DR election
       process. A larger value has a higher
       priority over a smaller value.";
    reference
      "RFC 7761: Protocol Independent Multicast -
       Sparse Mode (PIM-SM): Protocol
       Specification (Revised),
       Section 4.3.2";
  }
}
uses vpn-common:service-status;
```

```
<CODE ENDS>
```

9. 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 [RFC8446].

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.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) 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-profiles'**: This container includes a set of sensitive data that influence how the L3VPN service is delivered. For example, an attacker who has access to these data nodes may be able to manipulate routing policies, QoS policies, or encryption properties. These data nodes are defined with "nacm:default-deny-write" tagging [I-D.ietf-opsawg-vpn-common].
- * **'vpn-services'**: 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, keying material), leading to malfunctioning of the service and therefore to SLA violations. In addition, an attacker could attempt to create an L3VPN service or add a new network access. In addition to using NACM to prevent authorized access, such activity can be detected by adequately monitoring and tracking network configuration changes.

Some readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via `get`, `get-config`, or `notification`) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

- * `'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.
- * `'keying-material'`: An attacker can retrieve the cryptographic keys protecting the underlying VPN service (CE-PE routing, in particular). These keys could be used to inject spoofed routing advertisements.

Several data nodes (`'bgp'`, `'ospf'`, `'isis'`, `'rip'`, and `'bfd'`) rely upon [RFC8177] for authentication purposes. Therefore, this module inherits the security considerations discussed in Section 5 of [RFC8177]. Also, these data nodes support supplying explicit keys as strings in ASCII format. The use of keys in hexadecimal string format would afford greater key entropy with the same number of key-string octets. However, such format is not included in this version of the L3NM because it is not supported by the underlying device modules (e.g., [RFC8695]).

As discussed in Section 7.6.3, the module supports MD5 to basically accommodate the installed BGP base. MD5 suffers from the security weaknesses discussed in Section 2 of [RFC6151] or Section 2.1 of [RFC6952].

[RFC8633] describes best current practices to be considered in VPNs making use of NTP. Moreover, a mechanism to provide cryptographic security for NTP is specified in [RFC8915].

10. 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

11. References

11.1. Normative References

- [I-D.ietf-opsawg-vpn-common]
Barguil, S., Dios, O. G. D., Boucadair, M., and Q. Wu, "A Layer 2/3 VPN Common YANG Model", Work in Progress, Internet-Draft, draft-ietf-opsawg-vpn-common-11, 23 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-opsawg-vpn-common-11.txt>>.
- [ISO10589] ISO, "Intermediate System to Intermediate System intra-domain routing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473)", 2002, <International Standard 10589:2002, Second Edition>.
- [RFC1112] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, DOI 10.17487/RFC1112, August 1989, <<https://www.rfc-editor.org/info/rfc1112>>.
- [RFC1195] Callon, R., "Use of OSI IS-IS for routing in TCP/IP and dual environments", RFC 1195, DOI 10.17487/RFC1195, December 1990, <<https://www.rfc-editor.org/info/rfc1195>>.
- [RFC2080] Malkin, G. and R. Minnear, "RIPng for IPv6", RFC 2080, DOI 10.17487/RFC2080, January 1997, <<https://www.rfc-editor.org/info/rfc2080>>.
- [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>>.

- [RFC2236] Fenner, W., "Internet Group Management Protocol, Version 2", RFC 2236, DOI 10.17487/RFC2236, November 1997, <<https://www.rfc-editor.org/info/rfc2236>>.
- [RFC2453] Malkin, G., "RIP Version 2", STD 56, RFC 2453, DOI 10.17487/RFC2453, November 1998, <<https://www.rfc-editor.org/info/rfc2453>>.
- [RFC2710] Deering, S., Fenner, W., and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, DOI 10.17487/RFC2710, October 1999, <<https://www.rfc-editor.org/info/rfc2710>>.
- [RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, DOI 10.17487/RFC3376, October 2002, <<https://www.rfc-editor.org/info/rfc3376>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <<https://www.rfc-editor.org/info/rfc3810>>.
- [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>>.
- [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>>.
- [RFC4552] Gupta, M. and N. Melam, "Authentication/Confidentiality for OSPFv3", RFC 4552, DOI 10.17487/RFC4552, June 2006, <<https://www.rfc-editor.org/info/rfc4552>>.
- [RFC4577] Rosen, E., Psenak, P., and P. Pillay-Esnault, "OSPF as the Provider/Customer Edge Protocol for BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4577, DOI 10.17487/RFC4577, June 2006, <<https://www.rfc-editor.org/info/rfc4577>>.
- [RFC5308] Hopps, C., "Routing IPv6 with IS-IS", RFC 5308, DOI 10.17487/RFC5308, October 2008, <<https://www.rfc-editor.org/info/rfc5308>>.

- [RFC5701] Rekhter, Y., "IPv6 Address Specific BGP Extended Community Attribute", RFC 5701, DOI 10.17487/RFC5701, November 2009, <<https://www.rfc-editor.org/info/rfc5701>>.
- [RFC5709] Bhatia, M., Manral, V., Fanto, M., White, R., Barnes, M., Li, T., and R. Atkinson, "OSPFv2 HMAC-SHA Cryptographic Authentication", RFC 5709, DOI 10.17487/RFC5709, October 2009, <<https://www.rfc-editor.org/info/rfc5709>>.
- [RFC5798] Nadas, S., Ed., "Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6", RFC 5798, DOI 10.17487/RFC5798, March 2010, <<https://www.rfc-editor.org/info/rfc5798>>.
- [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>>.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <<https://www.rfc-editor.org/info/rfc5905>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [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>>.
- [RFC6565] Pillay-Esnault, P., Moyer, P., Doyle, J., Ertekin, E., and M. Lundberg, "OSPFv3 as a Provider Edge to Customer Edge (PE-CE) Routing Protocol", RFC 6565, DOI 10.17487/RFC6565, June 2012, <<https://www.rfc-editor.org/info/rfc6565>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7166] Bhatia, M., Manral, V., and A. Lindem, "Supporting Authentication Trailer for OSPFv3", RFC 7166, DOI 10.17487/RFC7166, March 2014, <<https://www.rfc-editor.org/info/rfc7166>>.
- [RFC7474] Bhatia, M., Hartman, S., Zhang, D., and A. Lindem, Ed., "Security Extension for OSPFv2 When Using Manual Key Management", RFC 7474, DOI 10.17487/RFC7474, April 2015, <<https://www.rfc-editor.org/info/rfc7474>>.
- [RFC7761] Fenner, B., Handley, M., Holbrook, H., Kouvelas, I., Parekh, R., Zhang, Z., and L. Zheng, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", STD 83, RFC 7761, DOI 10.17487/RFC7761, March 2016, <<https://www.rfc-editor.org/info/rfc7761>>.
- [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>>.
- [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>>.
- [RFC8177] Lindem, A., Ed., Qu, Y., Yeung, D., Chen, I., and J. Zhang, "YANG Data Model for Key Chains", RFC 8177, DOI 10.17487/RFC8177, June 2017, <<https://www.rfc-editor.org/info/rfc8177>>.

- [RFC8294] Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger, "Common YANG Data Types for the Routing Area", RFC 8294, DOI 10.17487/RFC8294, December 2017, <<https://www.rfc-editor.org/info/rfc8294>>.
- [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>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [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>>.

11.2. Informative References

- [I-D.evenwu-opsawg-yang-composed-vpn]
Even, R., Wu, B., Wu, Q., and YingCheng, "YANG Data Model for Composed VPN Service Delivery", Work in Progress, Internet-Draft, draft-evenwu-opsawg-yang-composed-vpn-03, 8 March 2019, <<https://www.ietf.org/archive/id/draft-evenwu-opsawg-yang-composed-vpn-03.txt>>.
- [I-D.ietf-bess-evpn-prefix-advertisement]
Rabadan, J., Henderickx, W., Drake, J. E., Lin, W., and A. Sajassi, "IP Prefix Advertisement in EVPN", Work in Progress, Internet-Draft, draft-ietf-bess-evpn-prefix-advertisement-11, 18 May 2018, <<https://www.ietf.org/archive/id/draft-ietf-bess-evpn-prefix-advertisement-11.txt>>.
- [I-D.ietf-idr-bgp-model]
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", Work in

Progress, Internet-Draft, draft-ietf-idr-bgp-model-11, 11 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-idr-bgp-model-11.txt>>.

[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)", Work in Progress, Internet-Draft, draft-ietf-pim-yang-17, 19 May 2018, <<https://www.ietf.org/archive/id/draft-ietf-pim-yang-17.txt>>.

[I-D.ietf-rtgwg-qos-model]

Choudhary, A., Jethanandani, M., Strahle, N., Aries, E., and I. Chen, "A YANG Data Model for Quality of Service (QoS)", Work in Progress, Internet-Draft, draft-ietf-rtgwg-qos-model-04, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-rtgwg-qos-model-04.txt>>.

[I-D.ietf-teas-enhanced-vpn]

Dong, J., Bryant, S., Li, Z., Miyasaka, T., and Y. Lee, "A Framework for Enhanced Virtual Private Network (VPN+) Services", Work in Progress, Internet-Draft, draft-ietf-teas-enhanced-vpn-08, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-teas-enhanced-vpn-08.txt>>.

[I-D.ietf-teas-ietf-network-slices]

Farrel, A., Gray, E., Drake, J., Rokui, R., Homma, S., Makhijani, K., Contreras, L. M., and J. Tantsura, "Framework for IETF Network Slices", Work in Progress, Internet-Draft, draft-ietf-teas-ietf-network-slices-04, 23 August 2021, <<https://www.ietf.org/archive/id/draft-ietf-teas-ietf-network-slices-04.txt>>.

[I-D.ogondio-opsawg-uni-topology]

Dios, O. G. D., Barguil, S., Wu, Q., and M. Boucadair, "A YANG Model for User-Network Interface (UNI) Topologies", Work in Progress, Internet-Draft, draft-ogondio-opsawg-uni-topology-01, 2 April 2020, <<https://www.ietf.org/archive/id/draft-ogondio-opsawg-uni-topology-01.txt>>.

[IEEE802.1AX]

"Link Aggregation", IEEE Std 802.1AX-2020, 2020.

- [PYANG] "pyang", November 2020,
<<https://github.com/mbj4668/pyang>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless
Address Autoconfiguration", RFC 4862,
DOI 10.17487/RFC4862, September 2007,
<<https://www.rfc-editor.org/info/rfc4862>>.
- [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>>.
- [RFC6151] Turner, S. and L. Chen, "Updated Security Considerations
for the MD5 Message-Digest and the HMAC-MD5 Algorithms",
RFC 6151, DOI 10.17487/RFC6151, March 2011,
<<https://www.rfc-editor.org/info/rfc6151>>.

- [RFC6952] Jethanandani, M., Patel, K., and L. Zheng, "Analysis of BGP, LDP, PCEP, and MSDP Issues According to the Keying and Authentication for Routing Protocols (KARP) Design Guide", RFC 6952, DOI 10.17487/RFC6952, May 2013, <<https://www.rfc-editor.org/info/rfc6952>>.
- [RFC7149] Boucadair, M. and C. Jacquenet, "Software-Defined Networking: A Perspective from within a Service Provider Environment", RFC 7149, DOI 10.17487/RFC7149, March 2014, <<https://www.rfc-editor.org/info/rfc7149>>.
- [RFC7297] Boucadair, M., Jacquenet, C., and N. Wang, "IP Connectivity Provisioning Profile (CPP)", RFC 7297, DOI 10.17487/RFC7297, July 2014, <<https://www.rfc-editor.org/info/rfc7297>>.
- [RFC7426] Haleplidis, E., Ed., Pentikousis, K., Ed., Denazis, S., Hadi Salim, J., Meyer, D., and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology", RFC 7426, DOI 10.17487/RFC7426, January 2015, <<https://www.rfc-editor.org/info/rfc7426>>.
- [RFC7880] Pignataro, C., Ward, D., Akiya, N., Bhatia, M., and S. Pallagatti, "Seamless Bidirectional Forwarding Detection (S-BFD)", RFC 7880, DOI 10.17487/RFC7880, July 2016, <<https://www.rfc-editor.org/info/rfc7880>>.
- [RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <<https://www.rfc-editor.org/info/rfc7942>>.
- [RFC8077] Martini, L., Ed. and G. Heron, Ed., "Pseudowire Setup and Maintenance Using the Label Distribution Protocol (LDP)", STD 84, RFC 8077, DOI 10.17487/RFC8077, February 2017, <<https://www.rfc-editor.org/info/rfc8077>>.
- [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>>.
- [RFC8633] Reilly, D., Stenn, H., and D. Sibold, "Network Time Protocol Best Current Practices", BCP 223, RFC 8633, DOI 10.17487/RFC8633, July 2019, <<https://www.rfc-editor.org/info/rfc8633>>.
- [RFC8695] Liu, X., Sarda, P., and V. Choudhary, "A YANG Data Model for the Routing Information Protocol (RIP)", RFC 8695, DOI 10.17487/RFC8695, February 2020, <<https://www.rfc-editor.org/info/rfc8695>>.
- [RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network Time Protocol", RFC 8915, DOI 10.17487/RFC8915, September 2020, <<https://www.rfc-editor.org/info/rfc8915>>.

[RFC8969] Wu, Q., Ed., Boucadair, M., Ed., Lopez, D., Xie, C., and L. Geng, "A Framework for Automating Service and Network Management with YANG", RFC 8969, DOI 10.17487/RFC8969, January 2021, <<https://www.rfc-editor.org/info/rfc8969>>.

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 31, 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'.

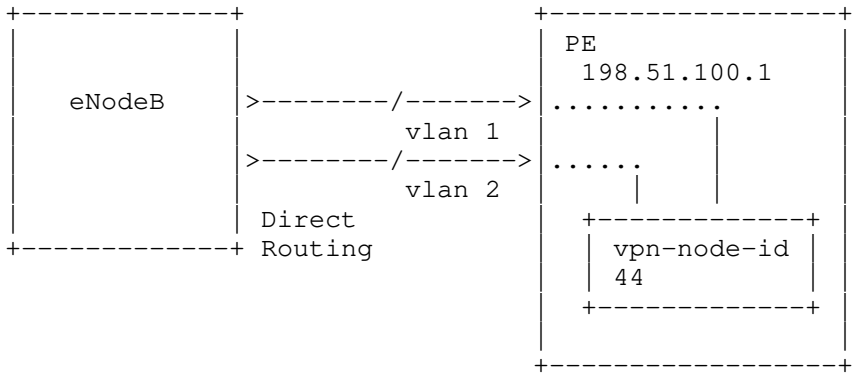


Figure 31: Mobile Backhaul Example

To create an L3VPN service using the L3NM, the following steps can be followed.

First: Create the 4G VPN service (Figure 32).

```

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",
        "vpn-description": "VPN to deploy 4G services",
        "vpn-instance-profiles": {
          "vpn-instance-profile": [
            {
              "profile-id": "simple-profile",
              "local-as": 65550,
              "rd": "0:65550:1",
              "address-family": [
                {
                  "address-family": "ietf-vpn-common:dual-stack",
                  "vpn-target": [
                    {
                      "id": 1,
                      "route-targets": [
                        {
                          "route-target": "0:65550:1"
                        }
                      ],
                      "route-target-type": "both"
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  }
}

```

Figure 32: Create VPN Service

Second: Create a VPN node as depicted in Figure 33. In this type of service, the VPN node is equivalent to the VRF configured in the physical device ('ne-id'=198.51.100.1).

===== NOTE: '\ ' line wrapping per RFC 8792 =====

```
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": "198.51.100.1",
        "active-vpn-instance-profiles": {
          "vpn-instance-profile": [
            {
              "profile-id": "simple-profile"
            }
          ]
        }
      }
    ]
  }
}
```

Figure 33: Create VPN Node

Finally, two VPN network accesses are created using the same physical port ('interface-id'=1/1/1). Each 'vpn-network-access' has a particular VLAN (1,2) to differentiate the traffic between: Sync and data (Figure 34).

===== NOTE: '\ ' line wrapping per RFC 8792 =====

```
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": [
      {
        "id": "1/1/1.1",
        "interface-id": "1/1/1",
        "description": "Interface SYNC to eNODE-B",
        "vpn-network-access-type": "ietf-vpn-common:point-to-point",
        "vpn-instance-profile": "simple-profile",
        "status": {
```

```
    "admin-status": {
      "status": "ietf-vpn-common:admin-up"
    }
  },
  "connection": {
    "encapsulation": {
      "type": "ietf-vpn-common:dot1q",
      "dot1q": {
        "cvlan-id": 1
      }
    }
  },
  "ip-connection": {
    "ipv4": {
      "local-address": "192.0.2.1",
      "prefix-length": 30,
      "address-allocation-type": "static-address",
      "static-addresses": {
        "primary-address": "1",
        "address": [
          {
            "address-id": "1",
            "customer-address": "192.0.2.2"
          }
        ]
      }
    },
    "ipv6": {
      "local-address": "2001:db8::1",
      "prefix-length": 64,
      "address-allocation-type": "static-address",
      "primary-address": "1",
      "address": [
        {
          "address-id": "1",
          "customer-address": "2001:db8::2"
        }
      ]
    }
  },
  "routing-protocols": {
    "routing-protocol": [
      {
        "id": "1",
        "type": "ietf-vpn-common:direct"
      }
    ]
  }
}
```

```
    },
    {
      "id": "1/1/1.2",
      "interface-id": "1/1/1",
      "description": "Interface DATA to eNODE-B",
      "vpn-network-access-type": "ietf-vpn-common:point-to-point",
      "vpn-instance-profile": "simple-profile",
      "status": {
        "admin-status": {
          "status": "ietf-vpn-common:admin-up"
        }
      },
      "connection": {
        "encapsulation": {
          "type": "ietf-vpn-common:dot1q",
          "dot1q": {
            "cvlan-id": 2
          }
        }
      },
      "ip-connection": {
        "ipv4": {
          "local-address": "192.0.2.1",
          "prefix-length": 30,
          "address-allocation-type": "static-address",
          "static-addresses": {
            "primary-address": "1",
            "address": [
              {
                "address-id": "1",
                "customer-address": "192.0.2.2"
              }
            ]
          }
        },
        "ipv6": {
          "local-address": "2001:db8::1",
          "prefix-length": 64,
          "address-allocation-type": "static-address",
          "primary-address": "1",
          "address": [
            {
              "address-id": "1",
              "customer-address": "2001:db8::2"
            }
          ]
        }
      }
    },
  ],
}
```

```

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

```

Figure 34: Create VPN Network Access

A.2. Loopback Interface

An example of loopback interface is depicted in Figure 35.

```

{
  "ietf-l3vpn-ntw:vpn-network-accesses": {
    "vpn-network-access": [
      {
        "id": "vpn-access-loopback",
        "interface-id": "Loopback1",
        "description": "An example of loopback interface.",
        "vpn-network-access-type": "ietf-vpn-common:loopback",
        "status": {
          "admin-status": {
            "status": "ietf-vpn-common:admin-up"
          }
        },
        "ip-connection": {
          "ipv6": {
            "local-address": "2001:db8::4",
            "prefix-length": 128
          }
        }
      }
    ]
  }
}

```

Figure 35: VPN Network Access with a Loopback Interface (Message Body)

A.3. Overriding VPN Instance Profile Parameters

Figure 36 shows a simplified example to illustrate how some information that is provided at the VPN service level (particularly as part of the 'vpn-instance-profiles') can be overridden by the one configured at the VPN node level. In this example, PE3 and PE4 inherit the 'vpn-instance-profiles' parameters that are specified at the VPN service level, but PE1 and PE2 are provided with "maximum-routes" values at the VPN node level that override the ones that are specified at the VPN service level.

```
{
  "ietf-l3vpn-ntw:vpn-services": {
    "vpn-service": [
      {
        "vpn-id": "override-example",
        "vpn-service-topology": "ietf-vpn-common:hub-spoke",
        "vpn-instance-profiles": {
          "vpn-instance-profile": [
            {
              "profile-id": "HUB",
              "role": "ietf-vpn-common:hub-role",
              "local-as": 64510,
              "rd-suffix": 1001,
              "address-family": [
                {
                  "address-family": "ietf-vpn-common:dual-stack",
                  "maximum-routes": [
                    {
                      "protocol": "ietf-vpn-common:any",
                      "maximum-routes": 100
                    }
                  ]
                }
              ]
            }
          ]
        },
        {
          "profile-id": "SPOKE",
          "role": "ietf-vpn-common:spoke-role",
          "local-as": 64510,
          "address-family": [
            {
              "address-family": "ietf-vpn-common:dual-stack",
              "maximum-routes": [
                {
                  "protocol": "ietf-vpn-common:any",
                  "maximum-routes": 1000
                }
              ]
            }
          ]
        }
      ]
    }
  }
}
```

```

    ]
  }
]
},
"vpn-nodes": {
  "vpn-node": [
    {
      "vpn-node-id": "PE1",
      "ne-id": "pe1",
      "router-id": "198.51.100.1",
      "active-vpn-instance-profiles": {
        "vpn-instance-profile": [
          {
            "profile-id": "HUB",
            "rd": "1:198.51.100.1:1001",
            "address-family": [
              {
                "address-family": "ietf-vpn-common:dual-stack",
                "maximum-routes": [
                  {
                    "protocol": "ietf-vpn-common:any",
                    "maximum-routes": 10
                  }
                ]
              }
            ]
          }
        ]
      }
    }
  ]
},
{
  "vpn-node-id": "PE2",
  "ne-id": "pe2",
  "router-id": "198.51.100.2",
  "active-vpn-instance-profiles": {
    "vpn-instance-profile": [
      {
        "profile-id": "SPOKE",
        "address-family": [
          {
            "address-family": "ietf-vpn-common:dual-stack",
            "maximum-routes": [
              {
                "protocol": "ietf-vpn-common:any",
                "maximum-routes": 100
              }
            ]
          }
        ]
      }
    ]
  }
}

```

```

    ]
  }
]
}
},
{
  "vpn-node-id": "PE3",
  "ne-id": "pe3",
  "router-id": "198.51.100.3",
  "active-vpn-instance-profiles": {
    "vpn-instance-profile": [
      {
        "profile-id": "SPOKE"
      }
    ]
  }
},
{
  "vpn-node-id": "PE4",
  "ne-id": "pe4",
  "router-id": "198.51.100.4",
  "active-vpn-instance-profiles": {
    "vpn-instance-profile": [
      {
        "profile-id": "SPOKE"
      }
    ]
  }
}
]
}
}
]
}
}
}

```

Figure 36: VPN Instance Profile Example (Message Body)

A.4. 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.

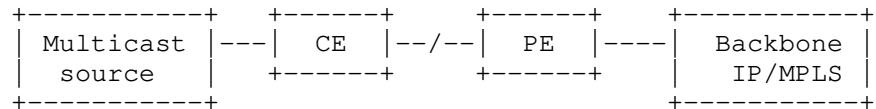


Figure 37: Multicast L3VPN Service Example

An example is provided below to illustrate how to configure a multicast L3VPN service using the L3NM.

First, the multicast service is created together with a generic VPN instance profile (see the excerpt of the request message body shown in Figure 38)

```

{
  "ietf-l3vpn-ntw:vpn-services": {
    "vpn-service": [
      {
        "vpn-id": "Multicast-IPTV",
        "vpn-description": "Multicast IPTV VPN service",
        "customer-name": "a-name",
        "vpn-service-topology": "ietf-vpn-common:hub-spoke",
        "vpn-instance-profiles": {
          "vpn-instance-profile": [
            {
              "profile-id": "multicast",
              "role": "ietf-vpn-common:hub-role",
              "local-as": 65536,
              "multicast": {
                "rp": {
                  "rp-group-mappings": {
                    "rp-group-mapping": [
                      {
                        "id": 1,
                        "rp-address": "203.0.113.17",
                        "groups": {
                          "group": [
                            {
                              "id": 1,
                              "group-address": "239.130.0.0/15"
                            }
                          ]
                        }
                      ]
                    }
                  },
                  "rp-discovery": {
                    "rp-discovery-type": "ietf-vpn-common:static-rp"
                  }
                }
              }
            ]
          }
        ]
      }
    ]
  }
}

```

Figure 38: 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 39). In this example, the VPN node will represent VRF configured in the physical device.

```
{
  "ietf-l3vpn-ntw:vpn-node": [
    {
      "vpn-node-id": "500003105",
      "description": "VRF-IPTV-MULTICAST",
      "ne-id": "198.51.100.10",
      "router-id": "198.51.100.10",
      "active-vpn-instance-profiles": {
        "vpn-instance-profile": [
          {
            "profile-id": "multicast",
            "rd": "65536:31050202"
          }
        ]
      }
    }
  ]
}
```

Figure 39: 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 40).

```
{
  "ietf-l3vpn-ntw:vpn-network-access": {
    "id": "1/1/1",
    "description": "Connected-to-source",
    "vpn-network-access-type": "ietf-vpn-common:point-to-point",
    "vpn-instance-profile": "multicast",
    "status": {
      "admin-status": {
        "status": "vpn-common:admin-up"
      },
      "ip-connection": {
        "ipv4": {
          "local-address": "203.0.113.1",
          "prefix-length": 30,
          "address-allocation-type": "static-address",
          "static-addresses": {
            "primary-address": "1",
            "address": [
              {

```

```

        "address-id": "1",
        "customer-address": "203.0.113.2"
    }
}
},
"routing-protocols": {
    "routing-protocol": [
        {
            "id": "1",
            "type": "ietf-vpn-common:bgp-routing",
            "bgp": {
                "description": "Connected to CE",
                "peer-as": "65537",
                "address-family": "ietf-vpn-common:ipv4",
                "neighbor": "203.0.113.2"
            }
        }
    ]
},
"service": {
    "inbound-bandwidth": "1000000000",
    "outbound-bandwidth": "1000000000",
    "mtu": 1500,
    "multicast": {
        "access-type": "source-only",
        "address-family": "ietf-vpn-common:ipv4",
        "protocol-type": "router",
        "pim": {
            "hello-interval": 30,
            "status": {
                "admin-status": {
                    "status": "ietf-vpn-common:admin-up"
                }
            }
        }
    }
}
}
}
}

```

Figure 40: 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 document and is based on a proposal described in [RFC7942]. 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 [RFC7942], "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 to the RFC Editor: As per [RFC7942] guidelines, please remove this Implementation Status appendix 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>

B.5. Juniper Implementation

<https://github.com/IETF-OPSAWG-WG/lxnm/blob/master/Implementattion/Juniper>

Acknowledgements

During the discussions of this work, helpful comments, suggestions, and reviews were received from (listed alphabetically): Raul Arco, Miguel Cros Cecilia, Joe Clarke, Dhruv Dhody, Adrian Farrel, Roque Gagliano, Christian Jacquenet, Kireeti Kompella, Julian Lucek, Greg Mirsky, and Tom Petch. Many thanks to them. Thanks to Philip Eardly for the review of an early version of the document.

Daniel King, Daniel Voyer, Luay Jalil, and Stephane Litkowski contributed to early version of the individual submission. Many thanks to Robert Wilton for the AD review. Thanks to Andrew Malis for the routing directorate review, Rifaat Shekh-Yusef for the security directorate review, Qin Wu for the opsdire review, and Pete Resnick for the genart directorate review. Thanks to Michael Scharf for the discussion on TCP-AO. Thanks to Martin Duke, Lars Eagert, Zaheduzzaman Sarker, Roman Danyliw, Erik Kline, Benjamin Kaduk, Francesca Palombini, and Eric Vyncke for the IESG review.

This work was supported in part by the European Commission funded H2020-ICT-2016-2 METRO-HAUL project (G.A. 761727) and Horizon 2020 Secured autonomic traffic management for a Tera of SDN flows (Teraflow) project (G.A. 101015857).

Contributors

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

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

Paul Sherratt
Gamma Telecom
Email: paul.sherratt@gamma.co.uk

Authors' Addresses

Samier Barguil
Telefonica
Madrid
Spain

Email: samier.barguilgiraldo.ext@telefonica.com

Oscar Gonzalez de Dios (editor)
Telefonica
Madrid
Spain

Email: oscar.gonzalezdedios@telefonica.com

Mohamed Boucadair (editor)
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Luis Angel Munoz
Vodafone
Spain

Email: luis-angel.munoz@vodafone.com

Alejandro Aguado
Nokia
Madrid
Spain

Email: alejandro.aguado_martin@nokia.com

opsawg
Internet-Draft
Intended status: Standards Track
Expires: 2 April 2022

S. Barguil
O. Gonzalez de Dios, Ed.
Telefonica
M. Boucadair, Ed.
Orange
Q. Wu
Huawei
29 September 2021

A Layer 2/3 VPN Common YANG Model
draft-ietf-opsawg-vpn-common-12

Abstract

This document defines a common YANG module that is meant to be reused by various VPN-related modules such as Layer 3 VPN and Layer 2 VPN network models.

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: A Layer 2/3 VPN Common YANG 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 2 April 2022.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Terminology	3
3. Description of the VPN Common YANG Module	3
4. Layer 2/3 VPN Common Module	13
5. Security Considerations	59
6. IANA Considerations	60
7. Acknowledgements	60
8. Contributors	61
9. References	61
9.1. Normative References	61
9.2. Informative References	62
Appendix A. Example of Common Data Nodes in Early L2NM/L3NM Designs	69
Authors' Addresses	69

1. Introduction

The IETF has specified YANG data modules for VPN services, e.g., Layer 3 VPN Service Model (L3SM) [RFC8299] or Layer 2 VPN Service Model (L2SM) [RFC8466]. Other relevant YANG models are the Layer 3 VPN Network Model (L3NM) [I-D.ietf-opsawg-l3sm-l3nm] and the Layer 2 VPN Network Model (L2NM) [I-D.ietf-opsawg-l2nm]. There are common data nodes and structures that are present in all of these models or at least a subset of them.

This document defines a common YANG module that is meant to be reused by various VPN-related modules such as L3NM [I-D.ietf-opsawg-l3sm-l3nm] and L2NM [I-D.ietf-opsawg-l2nm]: "ietf-vpn-common" (Section 4).

The "ietf-vpn-common" module includes a set of identities, types, and groupings that are meant to be reused by other VPN-related YANG modules independently of their layer (e.g., Layer 2, Layer 3) and the type of the module (e.g., network model, service model) including possible future revisions of existing models (e.g., L3SM [RFC8299] or L2SM [RFC8466]).

2. Terminology

The terminology for describing YANG modules is defined in [RFC7950].

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

The reader may refer to [RFC4026] and [RFC4176] for VPN-related terms.

The document inherits many terms from [RFC8299] and [RFC8466] (e.g., Enhanced Mobile Broadband (eMBB), Ultra-Reliable and Low Latency Communications (URLLC), Massive Machine Type Communications (mMTC)).

3. Description of the VPN Common YANG Module

The "ietf-vpn-common" module defines a set of common VPN-related features, including:

Encapsulation features such as:

- * Dot1q [IEEE802.1Q],
- * QinQ [IEEE802.1ad],
- * link aggregation [IEEE802.1AX], and
- * Virtual eXtensible Local Area Network (VXLAN) [RFC7348].

Multicast [RFC6513].

Routing features such as:

- * BGP [RFC4271],
- * OSPF [RFC4577][RFC6565],
- * IS-IS [ISO10589],
- * RIP [RFC2080][RFC2453],
- * Bidirectional Forwarding Detection (BFD) [RFC5880][RFC7880], and

- * Virtual Router Redundancy Protocol (VRRP) [RFC5798].

Also, the module defines a set of identities, including:

'service-type': Used to identify the VPN service type. Examples of supported service types are:

- * L3VPN,
- * Virtual Private LAN Service (VPLS) using BGP [RFC4761],
- * VPLS using Label Distribution Protocol (LDP) [RFC4762],
- * Virtual Private Wire Service (VPWS) [RFC8214],
- * BGP MPLS-Based Ethernet VPN [RFC7432],
- * Ethernet VPN (EVPN) [RFC8365], and
- * Provider Backbone Bridging Combined with Ethernet VPN (PBB-EVPN) [RFC7623].

'vpn-signaling-type': Used to identify the signaling mode used for a given service type. Examples of supported VPN signaling types are:

- * L2VPNs using BGP [RFC6624].
- * LDP [RFC5036], and
- * Layer Two Tunneling Protocol (L2TP) [RFC3931].

The module covers both IPv4 [RFC0791] and IPv6 [RFC8200] identities. It also includes multicast related identities such as Internet Group Management Protocol version 1 (IGMPv1) [RFC1112], IGMPv2 [RFC2236], IGMPv3 [RFC3376], Multicast Listener Discovery version 1 (MLDv1) [RFC2710], MLDv2 [RFC3810], and Protocol Independent Multicast (PIM) [RFC7761].

The reader should refer to Section 4 for the full list of supported identities (identities related to address families, VPN topologies, network access types, operational and administrative status, site or node roles, VPN service constraints, routing protocols, routes imports and exports, bandwidth and Quality of Service (QoS), etc.).

The "ietf-vpn-common" module also contains a set of reusable VPN-related groupings. The tree diagram of the "ietf-vpn-common" module that depicts the common groupings is provided in Figure 1.

```

module: ietf-vpn-common

  grouping vpn-description
    +-- vpn-id?          vpn-id
    +-- vpn-name?        string
    +-- vpn-description? string
    +-- customer-name?   string
  grouping vpn-profile-cfg
    +-- valid-provider-identifiers
      +-- external-connectivity-identifier* [id]
        | {external-connectivity}?
        | +-- id string
      +-- encryption-profile-identifier* [id]
        | +-- id string
      +-- qos-profile-identifier* [id]
        | +-- id string
      +-- bfd-profile-identifier* [id]
        | +-- id string
      +-- forwarding-profile-identifier* [id]
        | +-- id string
      +-- routing-profile-identifier* [id]
        | +-- id string
  grouping oper-status-timestamp
    +--ro status?      identityref
    +--ro last-change? yang:date-and-time
  grouping service-status
    +-- status
      +-- admin-status
        | +-- status?      identityref
        | +-- last-change? yang:date-and-time
      +-- oper-status
        +--ro status?      identityref
        +--ro last-change? yang:date-and-time
  grouping underlay-transport
    +-- (type)?
      +--:(abstract)
        | +-- transport-instance-id? string
      +--:(protocol)
        +-- protocol*          identityref
  grouping vpn-route-targets
    +-- vpn-target* [id]
      | +-- id                uint8
      | +-- route-targets* [route-target]
      | | +-- route-target    rt-types:route-target
      | +-- route-target-type rt-types:route-target-type
    +-- vpn-policies
      +-- import-policy? string
      +-- export-policy? string

```

```

grouping route-distinguisher
...
grouping vpn-components-group
  +-- groups
    +-- group* [group-id]
      +-- group-id    string
grouping placement-constraints
  +-- constraint* [constraint-type]
    +-- constraint-type?  identityref
    +-- target
      +-- (target-flavor)?
        +--:(id)
          |   +-- group* [group-id]
          |     +-- group-id    string
        +--:(all-accesses)
          |   +-- all-other-accesses?  empty
        +--:(all-groups)
          |   +-- all-other-groups?    empty
grouping ports
...
grouping qos-classification-policy
...

```

Figure 1: VPN Common Tree

The description of the common groupings is provided below:

'vpn-description':

A YANG grouping that provides common administrative VPN information such as an identifier, a name, a textual description, and a customer name.

'vpn-profile-cfg':

A YANG grouping that defines a set of valid profiles (encryption, routing, forwarding, etc.) that can be bound to a Layer 2/3 VPN. This document does not make any assumption about the structure of such profiles, but allows "gluing" a VPN service with other parameters that can be required locally to provide added value features to requesting customers.

For example, a service provider may provide an external connectivity to a VPN customer (e.g., to a private or public cloud, Internet). Such service may involve tweaking both filtering and NAT rules (e.g., bind a Virtual Routing and Forwarding (VRF) interface with a NAT instance as discussed in Section 2.10 of [RFC8512]). These added value features may be bound to all or a subset of network accesses. Some of these added value features may be implemented in nodes other than PEs (e.g., a P node or even a dedicated node that hosts the NAT function).

It is out of the scope of this document to elaborate the structure of these profiles.

'oper-status-timestamp':

A YANG grouping that defines the operational status updates of a VPN service or component.

'service-status':

A YANG grouping that defines the administrative and operational status of a component. The grouping can be applied to the whole service or an endpoint.

'underlay-transport':

A YANG grouping that defines the type of the underlay transport for a VPN service or how that underlay is set.

The underlay transport can be expressed as an abstract transport instance (e.g., an identifier of a VPN+ instance [I-D.ietf-teas-enhanced-vpn], a virtual network identifier [I-D.ietf-teas-actn-vn-yang][RFC8453], or a network slice name [I-D.ietf-teas-ietf-network-slices]) or as an ordered list of the actual protocols to be enabled in the network.

The module supports a rich set of protocol identifiers that can be used, e.g., to refer to an underlay transport. Examples of supported protocols are:

- IP-in-IP [RFC2003][RFC2473],
- GRE [RFC1701][RFC1702][RFC7676],
- MPLS-in-UDP [RFC7510],
- Generic Network Virtualization Encapsulation (GENEVE) [RFC8926],
- Segment Routing (SR) [RFC8660][RFC8663][RFC8754],

- Resource ReSerVation Protocol (RSVP) with traffic engineering extensions [RFC3209], and
- BGP with labeled prefixes [RFC8277].

'vpn-route-targets':

A YANG grouping that defines Route Target (RT) import/export rules used in a BGP-enabled VPN. This grouping can be used for both L3VPNs [RFC4364] and L2VPNs[RFC4664]. Note that this is modelled as a list to ease the reuse of this grouping in modules where an RT identifier is needed (e.g., associate an operator with RTs).

'route-distinguisher':

A YANG grouping that defines Route Distinguishers (RDs).

As depicted in Figure 2, the module supports these RD assignment modes: direct assignment, automatic assignment from a given pool, automatic assignment, and no assignment.

Also, the module accommodates deployments where only the Assigned Number subfield of RDs (Section 4.2 of [RFC4364]) is assigned from a pool while the Administrator subfield is set to, e.g., the router-id that is assigned to a VPN node. The module supports these modes for managing the Assigned Number subfield: explicit assignment, auto-assignment from a pool, and full auto-assignment.

```

grouping route-distinguisher
  +-- (rd-choice)?
    +--:(directly-assigned)
      | +-- rd?          rt-types:route-distinguisher
    +--:(directly-assigned-suffix)
      | +-- rd-suffix?   uint16
    +--:(auto-assigned)
      | +-- rd-auto
      |   +-- (auto-mode)?
      |     +--:(from-pool)
      |       | +-- rd-pool-name?   string
      |     +--:(full-auto)
      |       | +-- auto?           empty
      |       +--ro auto-assigned-rd? rt-types:route-distinguisher
    +--:(auto-assigned-suffix)
      | +-- rd-auto-suffix
      |   +-- (auto-mode)?
      |     +--:(from-pool)
      |       | +-- rd-pool-name?   string
      |     +--:(full-auto)
      |       | +-- auto?           empty
      |       +--ro auto-assigned-rd-suffix? uint16
    +--:(no-rd)
      | +-- no-rd?          empty

```

Figure 2: Route Distinguisher Grouping Subtree

'vpn-components-group':

A YANG grouping that is used to group VPN nodes, VPN network accesses, or sites. For example, diversity or redundancy constraints can be applied on a per-group basis.

'placement-constraints':

A YANG grouping that is used to define the placement constraints of a VPN node, VPN network access, or site.

'ports':

A YANG grouping that defines ranges of source and destination port numbers and operators. The subtree of this grouping is depicted in Figure 3.

```

grouping ports
+-- (source-port)?
|   +--:(source-port-range-or-operator)
|   |   +-- source-port-range-or-operator
|   |   |   +-- (port-range-or-operator)?
|   |   |   |   +--:(range)
|   |   |   |   |   +-- lower-port      inet:port-number
|   |   |   |   |   +-- upper-port     inet:port-number
|   |   |   |   +--:(operator)
|   |   |   |   |   +-- operator?      operator
|   |   |   |   |   +-- port           inet:port-number
|   +-- (destination-port)?
|   |   +--:(destination-port-range-or-operator)
|   |   |   +-- destination-port-range-or-operator
|   |   |   |   +-- (port-range-or-operator)?
|   |   |   |   |   +--:(range)
|   |   |   |   |   |   +-- lower-port      inet:port-number
|   |   |   |   |   |   +-- upper-port     inet:port-number
|   |   |   |   +--:(operator)
|   |   |   |   |   +-- operator?      operator
|   |   |   |   |   +-- port           inet:port-number

```

Figure 3: Port Numbers Grouping Subtree

'qos-classification-policy':

A YANG grouping that defines a set of QoS classification policies based on various match Layer 3/4 and application criteria. The subtree of this grouping is depicted in Figure 4.

The QoS match criteria reuse groupings that are defined in the packet fields module "ietf-packet-fields" (Section 4.2 of [RFC8519]).

Any layer 4 protocol can be indicated in the 'protocol' data node under 'l3', but only TCP and UDP specific match criteria are elaborated in this version as these protocols are widely used in the context of VPN services. Future revisions can be considered to add other Layer 4 specific parameters (e.g., Stream Control Transmission Protocol [RFC4960]), if needed.

Some transport protocols use existing protocols (e.g., TCP or UDP) as substrate. The match criteria for such protocols may rely upon the 'protocol' under 'l3', TCP/UDP match criteria shown in Figure 4, part of the TCP/UDP payload, or a combination thereof. This version of the module does not support such advanced match criteria. Future revisions of the module may consider adding match criteria based on the transport protocol payload (e.g., by means of a bitmask match).

```

grouping qos-classification-policy
  +-- rule* [id]
    +-- id                                     string
    +-- (match-type)?
      +--:(match-flow)
        +-- (l3)?
          +--:(ipv4)
            +-- ipv4
              +-- dscp?                       inet:dscp
              +-- ecn?                         uint8
              +-- length?                     uint16
              +-- ttl?                       uint8
              +-- protocol?                   uint8
              +-- ihl?                       uint8
              +-- flags?                     bits
              +-- offset?                    uint16
              +-- identification?            uint16
              +-- (destination-network)?
                +--:(destination-ipv4-network)
                  +-- destination-ipv4-network?
                    inet:ipv4-prefix
              +-- (source-network)?
                +--:(source-ipv4-network)
                  +-- source-ipv4-network?
                    inet:ipv4-prefix
          +--:(ipv6)
            +-- ipv6
              +-- dscp?                       inet:dscp
              +-- ecn?                         uint8
              +-- length?                     uint16
              +-- ttl?                       uint8
              +-- protocol?                   uint8
              +-- (destination-network)?
                +--:(destination-ipv6-network)
                  +-- destination-ipv6-network?
                    inet:ipv6-prefix
              +-- (source-network)?
                +--:(source-ipv6-network)
                  +-- source-ipv6-network?

```

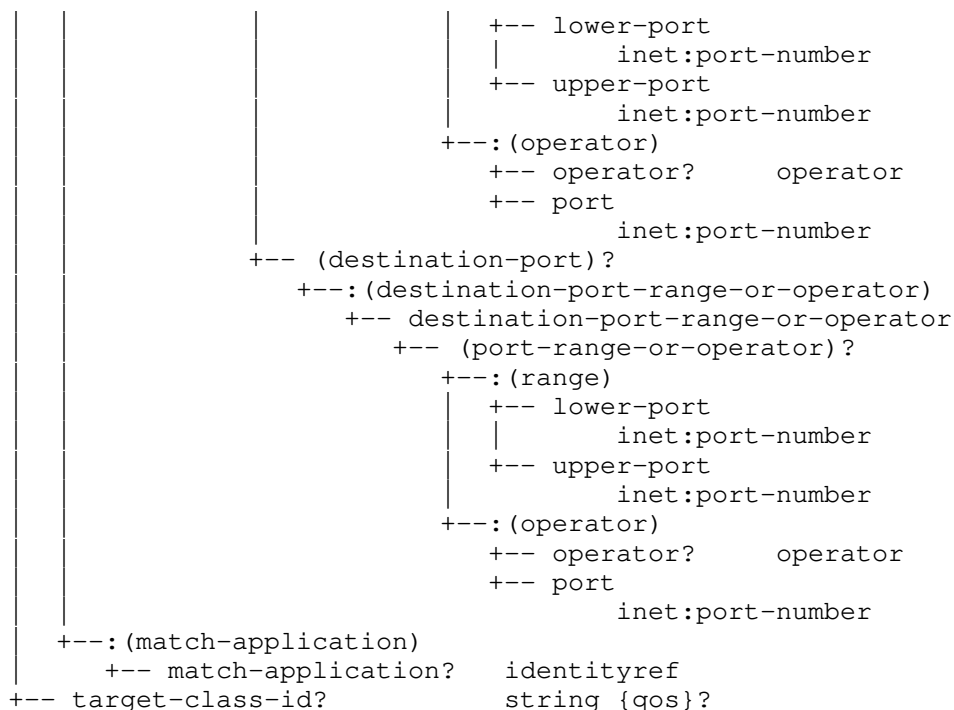



Figure 4: QoS Classification Subtree

4. Layer 2/3 VPN Common Module

This module uses types defined in [RFC6991], [RFC8294], and [RFC8519]. It also uses the extension defined in [RFC8341].

```
<CODE BEGINS> file "ietf-vpn-common@2021-09-10.yang"
module ietf-vpn-common {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-vpn-common";
  prefix vpn-common;

  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }
  import ietf-routing-types {
    prefix rt-types;
    reference
      "RFC 8294: Common YANG Data Types for the Routing Area";
```

```
}
import ietf-yang-types {
  prefix yang;
  reference
    "RFC 6991: Common YANG Data Types, Section 3";
}
import ietf-packet-fields {
  prefix packet-fields;
  reference
    "RFC 8519: YANG Data Model for Network Access
      Control Lists (ACLs)";
}

organization
  "IETF OPSAWG (Operations and Management Area Working Group)";
contact
  "WG Web:  <https://datatracker.ietf.org/wg/opsawg/>
  WG List:  <mailto:opsawg@ietf.org>

  Editor:   Mohamed Boucadair
            <mailto:mohamed.boucadair@orange.com>
  Author:   Samier Barguil
            <mailto:samier.barguilgiraldo.ext@telefonica.com>
  Author:   Oscar Gonzalez de Dios
            <mailto:oscar.gonzalezdedios@telefonica.com>
  Author:   Qin Wu
            <mailto:bill.wu@huawei.com>";

description
  "This YANG module defines a common module that is meant
  to be reused by various VPN-related modules (e.g.,
  Layer 3 VPN Service Model (L3SM), Layer 2 VPN Service
  Model (L2SM), Layer 3 VPN Network Model (L3NM), Layer 2
  VPN Network Model (L2NM)).

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

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Simplified BSD License
  set forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";

revision 2021-09-10 {
```

```
    description
      "Initial revision.";
    reference
      "RFC XXXX: A Layer 2/3 VPN Common YANG Model";
  }

  /***** Collection of VPN-related Features *****/
  /*
   * Features related to encapsulation schemes
   */

  feature dot1q {
    description
      "Indicates the support for the Dot1q encapsulation.";
    reference
      "IEEE Std 802.1Q: Bridges and Bridged Networks";
  }

  feature qinq {
    description
      "Indicates the support for the QinQ encapsulation.";
    reference
      "IEEE Std 802.1ad: Provider Bridges";
  }

  feature vxlan {
    description
      "Indicates the support for the Virtual eXtensible
       Local Area Network (VXLAN) encapsulation.";
    reference
      "RFC 7348: Virtual eXtensible Local Area Network (VXLAN):
       A Framework for Overlaying Virtualized Layer 2
       Networks over Layer 3 Networks";
  }

  feature qinany {
    description
      "Indicates the support for the QinAny encapsulation.
       The outer VLAN tag is set to a specific value but
       the inner VLAN tag is set to any.";
  }

  feature lag-interface {
    description
      "Indicates the support for Link Aggregation Group (LAG)
       between VPN network accesses.";
    reference
      "IEEE Std. 802.1AX: Link Aggregation";
  }
```

```
}

/*
 * Features related to multicast
 */

feature multicast {
  description
    "Indicates multicast capabilities support in a VPN.";
  reference
    "RFC 6513: Multicast in MPLS/BGP IP VPNs";
}

feature igmp {
  description
    "Indicates support for Internet Group Management Protocol
    (IGMP).";
  reference
    "RFC 1112: Host Extensions for IP Multicasting
    RFC 2236: Internet Group Management Protocol, Version 2
    RFC 3376: Internet Group Management Protocol, Version 3";
}

feature mld {
  description
    "Indicates support for Multicast Listener Discovery (MLD).";
  reference
    "RFC 2710: Multicast Listener Discovery (MLD) for IPv6
    RFC 3810: Multicast Listener Discovery Version 2 (MLDv2)
    for IPv6";
}

feature pim {
  description
    "Indicates support for Protocol Independent Multicast (PIM).";
  reference
    "RFC 7761: Protocol Independent Multicast - Sparse Mode
    (PIM-SM): Protocol Specification (Revised)";
}

/*
 * Features related to address family types
 */

feature ipv4 {
  description
    "Indicates IPv4 support in a VPN. That is, IPv4 traffic
    can be carried in the VPN, IPv4 addresses/prefixes can
```

```
        be assigned to a VPN network access, IPv4 routes can be
        installed for the CE/PE link, etc.";
    reference
        "RFC 791: Internet Protocol";
}

feature ipv6 {
    description
        "Indicates IPv6 support in a VPN. That is, IPv6 traffic
        can be carried in the VPN, IPv6 addresses/prefixes can
        be assigned to a VPN network access, IPv6 routes can be
        installed for the CE/PE link, etc.";
    reference
        "RFC 8200: Internet Protocol, Version 6 (IPv6)";
}

/*
 * Features related to routing protocols
 */

feature rtg-ospf {
    description
        "Indicates support for the OSPF as the Provider Edge (PE)/
        Customer Edge (CE) routing protocol.";
    reference
        "RFC 4577: OSPF as the Provider/Customer Edge Protocol
        for BGP/MPLS IP Virtual Private Networks (VPNs)
        RFC 6565: OSPFv3 as a Provider Edge to Customer Edge
        (PE-CE) Routing Protocol";
}

feature rtg-ospf-sham-link {
    description
        "Indicates support for OSPF sham links.";
    reference
        "RFC 4577: OSPF as the Provider/Customer Edge Protocol
        for BGP/MPLS IP Virtual Private Networks (VPNs),
        Section 4.2.7
        RFC 6565: OSPFv3 as a Provider Edge to Customer Edge
        (PE-CE) Routing Protocol, Section 5";
}

feature rtg-bgp {
    description
        "Indicates support for BGP as the PE/CE routing protocol.";
    reference
        "RFC 4271: A Border Gateway Protocol 4 (BGP-4)";
}
```

```
feature rtg-rip {
  description
    "Indicates support for RIP as the PE/CE routing protocol.";
  reference
    "RFC 2453: RIP Version 2
     RFC 2080: RIPng for IPv6";
}

feature rtg-isis {
  description
    "Indicates support for IS-IS as the PE/CE routing protocol.";
  reference
    "ISO10589: Intermediate System to Intermediate System intra-
     domain routeing information exchange protocol for
     use in conjunction with the protocol for providing
     the connectionless-mode network service
     (ISO 8473)";
}

feature rtg-vrrp {
  description
    "Indicates support for the Virtual Router Redundancy
     Protocol (VRRP) in CE/PE link.";
  reference
    "RFC 5798: Virtual Router Redundancy Protocol (VRRP) Version 3
     for IPv4 and IPv6";
}

feature bfd {
  description
    "Indicates support for Bidirectional Forwarding Detection (BFD)
     between the CE and the PE.";
  reference
    "RFC 5880: Bidirectional Forwarding Detection (BFD)";
}

/*
 * Features related to VPN service constraints
 */

feature bearer-reference {
  description
    "A bearer refers to properties of the CE-PE attachment that
     are below Layer 3.
     This feature indicates support for the bearer reference access
     constraint. That is, the reuse of a network connection that was
     already ordered to the service provider apart from the IP VPN
     site.";
```

```
}

feature placement-diversity {
  description
    "Indicates support for placement diversity constraints in the
    customer premises. An example of these constraints may be to
    avoid connecting a site network access to the same Provider
    Edge as a target site network access.";
}

/*
 * Features related to bandwidth and Quality of Service (QoS)
 */

feature qos {
  description
    "Indicates support for Classes of Service (CoSes) in the VPN.";
}

feature inbound-bw {
  description
    "Indicates support for the inbound bandwidth in a VPN. That is,
    support for specifying the download bandwidth from the service
    provider network to the VPN site. Note that the L3SM uses
    'input' to identify the same feature. That terminology should
    be deprecated in favor of the one defined in this module.";
}

feature outbound-bw {
  description
    "Indicates support for the outbound bandwidth in a VPN. That is,
    support for specifying the upload bandwidth from the VPN site
    to the service provider network. Note that the L3SM uses
    'output' to identify the same feature. That terminology should
    be deprecated in favor of the one defined in this module.";
}

/*
 * Features related to security and resilience
 */

feature encryption {
  description
    "Indicates support for encryption in the VPN.";
}

feature fast-reroute {
  description
```

```
        "Indicates support for Fast Reroute (FRR) capabilities for
        a VPN site.";
    }

    /*
     * Features related to advanced VPN options
     */

    feature external-connectivity {
        description
            "Indicates support for the VPN to provide external
            connectivity (e.g., Internet, private or public cloud).";
        reference
            "RFC 4364: BGP/MPLS IP Virtual Private Networks
            (VPNs), Section 11";
    }

    feature extranet-vpn {
        description
            "Indicates support for extranet VPNs. That is, the capability of
            a VPN to access a list of other VPNs.";
        reference
            "RFC 4364: BGP/MPLS IP Virtual Private Networks
            (VPNs), Section 1.1";
    }

    feature carriers-carrier {
        description
            "Indicates support for Carrier-of-Carrier VPNs.";
        reference
            "RFC 4364: BGP/MPLS IP Virtual Private Networks
            (VPNs), Section 9";
    }

    /*
     * Address family related identities
     */

    identity address-family {
        description
            "Defines a type for the address family.";
    }

    identity ipv4 {
        base address-family;
        description
            "Identity for IPv4 address family.";
    }
```

```
identity ipv6 {
    base address-family;
    description
        "Identity for IPv6 address family.";
}

identity dual-stack {
    base address-family;
    description
        "Identity for IPv4 and IPv6 address family.";
}

/*
 * Identities related to VPN topology
 */

identity vpn-topology {
    description
        "Base identity of the VPN topology.";
}

identity any-to-any {
    base vpn-topology;
    description
        "Identity for any-to-any VPN topology. All VPN sites
        can communicate with each other without any restrictions.";
}

identity hub-spoke {
    base vpn-topology;
    description
        "Identity for Hub-and-Spoke VPN topology. All Spokes can
        communicate only with Hubs but not with each other. Hubs
        can communicate with each other.";
}

identity hub-spoke-disjoint {
    base vpn-topology;
    description
        "Identity for Hub-and-Spoke VPN topology where Hubs cannot
        communicate with each other.";
}

identity custom {
    base vpn-topology;
    description
        "Identity for custom VPN topologies where the role of the nodes
        is not strictly Hub or Spoke. The VPN topology is controlled by
```

```
        the import/export policies. The custom topology reflects more
        complex VPN nodes such as VPN node that acts as Hub for certain
        nodes and Spoke to others.";
    }

    /*
     * Identities related to network access types
     */

    identity site-network-access-type {
        description
            "Base identity for site network access type.";
    }

    identity point-to-point {
        base site-network-access-type;
        description
            "Point-to-point access type.";
    }

    identity multipoint {
        base site-network-access-type;
        description
            "Multipoint access type.";
    }

    identity irb {
        base site-network-access-type;
        description
            "Integrated Routing Bridge (IRB).
             Identity for pseudowire connections.";
    }

    identity loopback {
        base site-network-access-type;
        description
            "Loopback access type.";
    }

    /*
     * Identities related to operational and administrative status
     */

    identity operational-status {
        description
            "Base identity for the operational status.";
    }
}
```

```
identity op-up {
  base operational-status;
  description
    "Operational status is Up/Enabled.";
}

identity op-down {
  base operational-status;
  description
    "Operational status is Down/Disabled.";
}

identity op-unknown {
  base operational-status;
  description
    "Operational status is Unknown.";
}

identity administrative-status {
  description
    "Base identity for administrative status.";
}

identity admin-up {
  base administrative-status;
  description
    "Administrative status is Up/Enabled.";
}

identity admin-down {
  base administrative-status;
  description
    "Administrative status is Down/Disabled.";
}

identity admin-testing {
  base administrative-status;
  description
    "Administrative status is up for testing purposes.";
}

identity admin-pre-deployment {
  base administrative-status;
  description
    "Administrative status is pre-deployment phase. That is,
    prior to the actual deployment of a service.";
}
```

```
/*
 * Identities related to site or node role
 */

identity role {
  description
    "Base identity of a site or a node role.";
}

identity any-to-any-role {
  base role;
  description
    "Any-to-any role.";
}

identity spoke-role {
  base role;
  description
    "A node or a site is acting as a Spoke.";
}

identity hub-role {
  base role;
  description
    "A node or a site is acting as a Hub.";
}

identity custom-role {
  base role;
  description
    "VPN node with custom or complex role in the VPN. For some
    sources/destinations it can behave as a Hub, but for others it
    can act as a Spoke depending on the configured policy.";
}

/*
 * Identities related to VPN service constraints
 */

identity placement-diversity {
  description
    "Base identity for access placement constraints.";
}

identity bearer-diverse {
  base placement-diversity;
  description
    "Bearer diversity.
```

```
        The bearers should not use common elements.";
    }

    identity pe-diverse {
        base placement-diversity;
        description
            "PE diversity.";
    }

    identity pop-diverse {
        base placement-diversity;
        description
            "Point Of Presence (POP) diversity.";
    }

    identity linecard-diverse {
        base placement-diversity;
        description
            "Linecard diversity.";
    }

    identity same-pe {
        base placement-diversity;
        description
            "Having sites connected on the same PE.";
    }

    identity same-bearer {
        base placement-diversity;
        description
            "Having sites connected using the same bearer.";
    }

    /*
     * Identities related to service types
     */

    identity service-type {
        description
            "Base identity for service type.";
    }

    identity l3vpn {
        base service-type;
        description
            "L3VPN service.";
        reference
            "RFC 4364: BGP/MPLS IP Virtual Private Networks (VPNs)";
    }
}
```

```
}

identity vpls {
  base service-type;
  description
    "VPLS service.";
  reference
    "RFC 4761: Virtual Private LAN Service (VPLS) Using BGP for
      Auto-Discovery and Signaling
     RFC 4762: Virtual Private LAN Service (VPLS) Using Label
      Distribution Protocol (LDP) Signaling";
}

identity vpws {
  base service-type;
  description
    "Virtual Private Wire Service (VPWS) service.";
  reference
    "RFC 4664: Framework for Layer 2 Virtual Private Networks
      (L2VPNs), Section 3.1.1";
}

identity vpws-evpn {
  base service-type;
  description
    "EVPN used to support VPWS service.";
  reference
    "RFC 8214: Virtual Private Wire Service Support in Ethernet VPN";
}

identity pbb-evpn {
  base service-type;
  description
    "Provider Backbone Bridging (PBB) EVPNs service.";
  reference
    "RFC 7623: Provider Backbone Bridging Combined with Ethernet VPN
      (PBB-EVPN)";
}

identity mpls-evpn {
  base service-type;
  description
    "MPLS-based EVPN service.";
  reference
    "RFC 7432: BGP MPLS-Based Ethernet VPN";
}

identity vxlan-evpn {
```

```
    base service-type;
    description
        "VXLAN-based EVPN service.";
    reference
        "RFC 8365: A Network Virtualization Overlay Solution Using
        Ethernet VPN (EVPN)";
}

/*
 * Identities related to VPN signaling type
 */

identity vpn-signaling-type {
    description
        "Base identity for VPN signaling types";
}

identity bgp-signaling {
    base vpn-signaling-type;
    description
        "Layer 2 VPNs using BGP signaling.";
    reference
        "RFC 6624: Layer 2 Virtual Private Networks Using BGP for
        Auto-Discovery and Signaling
        RFC 7432: BGP MPLS-Based Ethernet VPN";
}

identity ldp-signaling {
    base vpn-signaling-type;
    description
        "Targeted Label Distribution Protocol (LDP) signaling.";
    reference
        "RFC 5036: LDP Specification";
}

identity l2tp-signaling {
    base vpn-signaling-type;
    description
        "Layer Two Tunneling Protocol (L2TP) signaling.";
    reference
        "RFC 3931: Layer Two Tunneling Protocol - Version 3 (L2TPv3)";
}

/*
 * Identities related to routing protocols
 */

identity routing-protocol-type {
```

```
    description
      "Base identity for routing protocol type.";
  }

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

  identity bgp-routing {
    if-feature "rtg-bgp";
    base routing-protocol-type;
    description
      "BGP routing protocol.";
    reference
      "RFC 4271: A Border Gateway Protocol 4 (BGP-4)";
  }

  identity ospf-routing {
    if-feature "rtg-ospf";
    base routing-protocol-type;
    description
      "OSPF routing protocol.";
    reference
      "RFC 4577: OSPF as the Provider/Customer Edge Protocol
        for BGP/MPLS IP Virtual Private Networks (VPNs)
        RFC 6565: OSPFv3 as a Provider Edge to Customer Edge
        (PE-CE) Routing Protocol";
  }

  identity rip-routing {
    if-feature "rtg-rip";
    base routing-protocol-type;
    description
      "RIP routing protocol.";
    reference
      "RFC 2453: RIP Version 2
        RFC 2080: RIPng for IPv6";
  }

  identity isis-routing {
    if-feature "rtg-isis";
    base routing-protocol-type;
    description
      "IS-IS routing protocol.";
    reference
      "ISO10589: Intermediate System to Intermediate System intra-
```

```
        domain routeing information exchange protocol for
        use in conjunction with the protocol for providing
        the connectionless-mode network service
        (ISO 8473)";
    }

    identity vrrp-routing {
        if-feature "rtg-vrrp";
        base routing-protocol-type;
        description
            "VRRP protocol.

            This is to be used when LANs are directly connected to PEs.";
        reference
            "RFC 5798: Virtual Router Redundancy Protocol (VRRP) Version 3
            for IPv4 and IPv6";
    }

    identity direct-routing {
        base routing-protocol-type;
        description
            "Direct routing.

            This is to be used when LANs are directly connected to PEs
            and must be advertised in the VPN.";
    }

    identity any-routing {
        base routing-protocol-type;
        description
            "Any routing protocol.

            This can be, e.g., used to set policies that apply to any
            routing protocol in place.";
    }

    identity isis-level {
        if-feature "rtg-isis";
        description
            "Base identity for the IS-IS level.";
        reference
            "ISO10589: Intermediate System to Intermediate System intra-
            domain routeing information exchange protocol for
            use in conjunction with the protocol for providing
            the connectionless-mode network service
            (ISO 8473)";
    }
}
```

```
identity level-1 {
  base isis-level;
  description
    "IS-IS level 1.";
}

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

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

identity bfd-session-type {
  if-feature "bfd";
  description
    "Base identity for the BFD session type.";
}

identity classic-bfd {
  base bfd-session-type;
  description
    "Classic BFD.";
  reference
    "RFC 5880: Bidirectional Forwarding Detection (BFD)";
}

identity s-bfd {
  base bfd-session-type;
  description
    "Seamless BFD.";
  reference
    "RFC 7880: Seamless Bidirectional Forwarding Detection (S-BFD)";
}

/*
 * Identities related to Routes Import and Export
 */

identity ie-type {
  description
    "Base identity for 'import/export' routing profiles.
    These profiles can be reused between VPN nodes.";
```

```
    }

    identity import {
      base ie-type;
      description
        "'Import' routing profile.";
      reference
        "RFC 4364: BGP/MPLS IP Virtual Private Networks
        (VPNs), Section 4.3.1";
    }

    identity export {
      base ie-type;
      description
        "'Export' routing profile.";
      reference
        "RFC 4364: BGP/MPLS IP Virtual Private Networks
        (VPNs), Section 4.3.1";
    }

    identity import-export {
      base ie-type;
      description
        "'Import/export' routing profile.";
    }

    /*
     * Identities related to bandwidth and QoS
     */

    identity bw-direction {
      description
        "Base identity for the bandwidth direction.";
    }

    identity inbound-bw {
      if-feature "inbound-bw";
      base bw-direction;
      description
        "Inbound bandwidth.";
    }

    identity outbound-bw {
      if-feature "outbound-bw";
      base bw-direction;
      description
        "Outbound bandwidth.";
    }
  }
```

```
identity bw-type {
  description
    "Base identity for the bandwidth type.";
}

identity bw-per-cos {
  if-feature "qos";
  base bw-type;
  description
    "The bandwidth is per-CoS.";
}

identity bw-per-port {
  base bw-type;
  description
    "The bandwidth is per-site network access.";
}

identity bw-per-site {
  base bw-type;
  description
    "The bandwidth is per-site. It is applicable to all the site
    network accesses within a site.";
}

identity bw-per-service {
  base bw-type;
  description
    "The bandwidth is per-VPN service.";
}

identity qos-profile-direction {
  if-feature "qos";
  description
    "Base identity for the QoS profile direction.";
}

identity site-to-wan {
  base qos-profile-direction;
  description
    "Customer site to provider's network direction.
    This is typically the CE-to-PE direction.";
}

identity wan-to-site {
  base qos-profile-direction;
  description
    "Provider's network to customer site direction.
```

```
        This is typically the PE-to-CE direction.";
    }

    identity both {
        base qos-profile-direction;
        description
            "Both WAN-to-Site and Site-to-WAN directions.";
    }

    /*
     * Identities related to underlay transport instances
     */

    identity transport-instance-type {
        description
            "Base identity for underlay transport instance type.";
    }

    identity virtual-network {
        base transport-instance-type;
        description
            "Virtual network.";
        reference
            "RFC 8453: Framework for Abstraction and Control of TE
             Networks (ACTN)";
    }

    identity enhanced-vpn {
        base transport-instance-type;
        description
            "Enhanced VPN (VPN+). VPN+ is an approach that is
             based on existing VPN and Traffic Engineering (TE)
             technologies but adds characteristics that specific
             services require over and above classical VPNs.";
        reference
            "I-D.ietf-teas-enhanced-vpn:
             A Framework for Enhanced Virtual Private Network
             (VPN+) Services";
    }

    identity ietf-network-slice {
        base transport-instance-type;
        description
            "IETF network slice. An IETF network slice
             is a logical network topology connecting a number of
             endpoints using a set of shared or dedicated network
             resources that are used to satisfy specific service
             objectives.";
```

```
    reference
      "I-D.ietf-teas-ietf-network-slices:
       Framework for IETF Network Slices";
  }

  /*
   * Identities related to protocol types. These types are typically
   * used to identify the underlay transport.
   */

  identity protocol-type {
    description
      "Base identity for Protocol Type.";
  }

  identity ip-in-ip {
    base protocol-type;
    description
      "Transport is based on IP-in-IP.";
    reference
      "RFC 2003: IP Encapsulation within IP
       RFC 2473: Generic Packet Tunneling in IPv6 Specification";
  }

  identity ip-in-ipv4 {
    base ip-in-ip;
    description
      "Transport is based on IP over IPv4.";
    reference
      "RFC 2003: IP Encapsulation within IP";
  }

  identity ip-in-ipv6 {
    base ip-in-ip;
    description
      "Transport is based on IP over IPv6.";
    reference
      "RFC 2473: Generic Packet Tunneling in IPv6 Specification";
  }

  identity gre {
    base protocol-type;
    description
      "Transport is based on Generic Routing Encapsulation (GRE).";
    reference
      "RFC 1701: Generic Routing Encapsulation (GRE)
       RFC 1702: Generic Routing Encapsulation over IPv4 networks
       RFC 7676: IPv6 Support for Generic Routing Encapsulation (GRE)";
  }
```

```
}

identity gre-v4 {
  base gre;
  description
    "Transport is based on GRE over IPv4.";
  reference
    "RFC 1702: Generic Routing Encapsulation over IPv4 networks";
}

identity gre-v6 {
  base gre;
  description
    "Transport is based on GRE over IPv6.";
  reference
    "RFC 7676: IPv6 Support for Generic Routing Encapsulation (GRE)";
}

identity vxlan-trans {
  base protocol-type;
  description
    "Transport is based on VXLAN.";
  reference
    "RFC 7348: Virtual eXtensible Local Area Network (VXLAN):
      A Framework for Overlaying Virtualized Layer 2
      Networks over Layer 3 Networks";
}

identity geneve {
  base protocol-type;
  description
    "Transport is based on Generic Network Virtualization
      Encapsulation (GENEVE).";
  reference
    "RFC 8926: Geneve: Generic Network Virtualization Encapsulation";
}

identity ldp {
  base protocol-type;
  description
    "Transport is based on LDP.";
  reference
    "RFC 5036: LDP Specification";
}

identity mpls-in-udp {
  base protocol-type;
  description
```

```
        "Transport is MPLS in UDP.";
    reference
        "RFC 7510: Encapsulating MPLS in UDP";
}

identity sr {
    base protocol-type;
    description
        "Transport is based on Segment Routing (SR).";
    reference
        "RFC 8660: Segment Routing with the MPLS Data Plane
        RFC 8663: MPLS Segment Routing over IP
        RFC 8754: IPv6 Segment Routing Header (SRH)";
}

identity sr-mpls {
    base sr;
    description
        "Transport is based on SR with MPLS.";
    reference
        "RFC 8660: Segment Routing with the MPLS Data Plane";
}

identity srv6 {
    base sr;
    description
        "Transport is based on SR over IPv6.";
    reference
        "RFC 8754: IPv6 Segment Routing Header (SRH)";
}

identity sr-mpls-over-ip {
    base sr;
    description
        "Transport is based on SR over MPLS over IP.";
    reference
        "RFC 8663: MPLS Segment Routing over IP";
}

identity rsvp-te {
    base protocol-type;
    description
        "Transport setup relies upon RSVP-TE.";
    reference
        "RFC 3209: RSVP-TE: Extensions to RSVP for LSP Tunnels";
}

identity bgp-lu {
```

```
    base protocol-type;
    description
        "Transport setup relies upon BGP-LU.";
    reference
        "RFC 8277: Using BGP to Bind MPLS Labels to Address Prefixes";
}

identity unknown {
    base protocol-type;
    description
        "Not known protocol type.";
}

/*
 * Identities related to encapsulations
 */

identity encapsulation-type {
    description
        "Base identity for the encapsulation type.";
}

identity priority-tagged {
    base encapsulation-type;
    description
        "Priority-tagged interface.";
}

identity dot1q {
    if-feature "dot1q";
    base encapsulation-type;
    description
        "Dot1q encapsulation.";
}

identity qinq {
    if-feature "qinq";
    base encapsulation-type;
    description
        "QinQ encapsulation.";
}

identity qinany {
    if-feature "qinany";
    base encapsulation-type;
    description
        "QinAny encapsulation.";
}
```

```
identity vxlan {
  if-feature "vxlan";
  base encapsulation-type;
  description
    "VxLAN encapsulation.";
}

identity ethernet-type {
  base encapsulation-type;
  description
    "Ethernet encapsulation type.";
}

identity vlan-type {
  base encapsulation-type;
  description
    "VLAN encapsulation type.";
}

identity untagged-int {
  base encapsulation-type;
  description
    "Untagged interface type.";
}

identity tagged-int {
  base encapsulation-type;
  description
    "Tagged interface type.";
}

identity lag-int {
  if-feature "lag-interface";
  base encapsulation-type;
  description
    "LAG interface type.";
}

/*
 * Identities related to VLAN Tag
 */

identity tag-type {
  description
    "Base identity for the tag types.";
}

identity c-vlan {
```

```
    base tag-type;
    description
        "Indicates Customer VLAN (C-VLAN) tag, normally using
        the 0x8100 Ethertype.";
}

identity s-vlan {
    base tag-type;
    description
        "Indicates Service VLAN (S-VLAN) tag.";
}

identity s-c-vlan {
    base tag-type;
    description
        "Uses both an S-VLAN tag and a C-VLAN tag.";
}

/*
 * Identities related to VXLAN
 */

identity vxlan-peer-mode {
    if-feature "vxlan";
    description
        "Base identity for the VXLAN peer mode.";
}

identity static-mode {
    base vxlan-peer-mode;
    description
        "VXLAN access in the static mode.";
}

identity bgp-mode {
    base vxlan-peer-mode;
    description
        "VXLAN access by BGP EVPN learning.";
}

/*
 * Identities related to multicast
 */

identity multicast-gp-address-mapping {
    if-feature "multicast";
    description
        "Base identity for multicast group mapping type.";
```

```
}

identity static-mapping {
  base multicast-gp-address-mapping;
  description
    "Static mapping, i.e., attach the interface to the
    multicast group as a static member.";
}

identity dynamic-mapping {
  base multicast-gp-address-mapping;
  description
    "Dynamic mapping, i.e., an interface is added to the
    multicast group as a result of snooping.";
}

identity multicast-tree-type {
  if-feature "multicast";
  description
    "Base identity for multicast tree type.";
}

identity ssm-tree-type {
  base multicast-tree-type;
  description
    "Source-Specific Multicast (SSM) tree type.";
}

identity asm-tree-type {
  base multicast-tree-type;
  description
    "Any-Source Multicast (ASM) tree type.";
}

identity bidir-tree-type {
  base multicast-tree-type;
  description
    "Bidirectional tree type.";
}

identity multicast-rp-discovery-type {
  if-feature "multicast";
  description
    "Base identity for Rendezvous Point (RP) discovery type.";
}

identity auto-rp {
  base multicast-rp-discovery-type;
```

```
    description
      "Auto-RP discovery type.";
  }

  identity static-rp {
    base multicast-rp-discovery-type;
    description
      "Static type.";
  }

  identity bsr-rp {
    base multicast-rp-discovery-type;
    description
      "Bootstrap Router (BSR) discovery type.";
  }

  identity group-management-protocol {
    if-feature "multicast";
    description
      "Base identity for multicast group management protocol.";
  }

  identity igmp-proto {
    base group-management-protocol;
    description
      "IGMP.";
    reference
      "RFC 1112: Host Extensions for IP Multicasting
       RFC 2236: Internet Group Management Protocol, Version 2
       RFC 3376: Internet Group Management Protocol, Version 3";
  }

  identity mld-proto {
    base group-management-protocol;
    description
      "MLD.";
    reference
      "RFC 2710: Multicast Listener Discovery (MLD) for IPv6
       RFC 3810: Multicast Listener Discovery Version 2 (MLDv2)
        for IPv6";
  }

  identity pim-proto {
    if-feature "pim";
    base routing-protocol-type;
    description
      "PIM.";
    reference
```

```
        "RFC 7761: Protocol Independent Multicast - Sparse Mode
          (PIM-SM): Protocol Specification (Revised)";
    }

    identity igmp-version {
        if-feature "igmp";
        description
            "Base identity for IGMP version.";
    }

    identity igmpv1 {
        base igmp-version;
        description
            "IGMPv1.";
        reference
            "RFC 1112: Host Extensions for IP Multicasting";
    }

    identity igmpv2 {
        base igmp-version;
        description
            "IGMPv2.";
        reference
            "RFC 2236: Internet Group Management Protocol, Version 2";
    }

    identity igmpv3 {
        base igmp-version;
        description
            "IGMPv3.";
        reference
            "RFC 3376: Internet Group Management Protocol, Version 3";
    }

    identity mld-version {
        if-feature "mld";
        description
            "Base identity for MLD version.";
    }

    identity mldv1 {
        base mld-version;
        description
            "MLDv1.";
        reference
            "RFC 2710: Multicast Listener Discovery (MLD) for IPv6";
    }
```

```
identity mldv2 {
  base mld-version;
  description
    "MLDv2.";
  reference
    "RFC 3810: Multicast Listener Discovery Version 2 (MLDv2)
      for IPv6";
}

/*
 * Identities related to traffic types
 */

identity tf-type {
  description
    "Base identity for the traffic type.";
}

identity multicast-traffic {
  base tf-type;
  description
    "Multicast traffic.";
}

identity broadcast-traffic {
  base tf-type;
  description
    "Broadcast traffic.";
}

identity unknown-unicast-traffic {
  base tf-type;
  description
    "Unknown unicast traffic.";
}

/*
 * Identities related to customer applications
 */

identity customer-application {
  description
    "Base identity for customer applications.";
}

identity web {
  base customer-application;
  description
```

```
        "Web applications (e.g., HTTP, HTTPS).";
    }

    identity mail {
        base customer-application;
        description
            "Mail application.";
    }

    identity file-transfer {
        base customer-application;
        description
            "File transfer application (e.g., FTP, SFTP).";
    }

    identity database {
        base customer-application;
        description
            "Database application.";
    }

    identity social {
        base customer-application;
        description
            "Social-network application.";
    }

    identity games {
        base customer-application;
        description
            "Gaming application.";
    }

    identity p2p {
        base customer-application;
        description
            "Peer-to-peer application.";
    }

    identity network-management {
        base customer-application;
        description
            "Management application (e.g., Telnet, syslog,
            SNMP).";
    }

    identity voice {
        base customer-application;
```

```
    description
      "Voice application.";
  }

  identity video {
    base customer-application;
    description
      "Video conference application.";
  }

  identity embb {
    base customer-application;
    description
      "Enhanced Mobile Broadband (eMBB) application.
       Note that an eMBB application demands network performance with a
       wide variety of characteristics, such as data rate, latency,
       loss rate, reliability, and many other parameters.";
  }

  identity urlhc {
    base customer-application;
    description
      "Ultra-Reliable and Low Latency Communications
       (URLLC) application. Note that an URLLC application demands
       network performance with a wide variety of characteristics, such
       as latency, reliability, and many other parameters.";
  }

  identity mmhc {
    base customer-application;
    description
      "Massive Machine Type Communications (mMTC) application.
       Note that an mMTC application demands network performance with
       a wide variety of characteristics, such as data rate, latency,
       loss rate, reliability, and many other parameters.";
  }

  /*
   * Identities related to service bundling
   */

  identity bundling-type {
    description
      "The base identity for the bundling type. It supports a subset or
       all CE-VLANs associated with an L2VPN service.";
  }

  identity multi-svc-bundling {
```

```
    base bundling-type;
    description
        "Multi-service bundling, i.e., multiple C-VLAN IDs
        can be associated with an L2VPN service at a site.";
}

identity one2one-bundling {
    base bundling-type;
    description
        "One-to-one service bundling, i.e., each L2VPN can
        be associated with only one C-VLAN ID at a site.";
}

identity all2one-bundling {
    base bundling-type;
    description
        "All-to-one bundling, i.e., all C-VLAN IDs are mapped
        to one L2VPN service.";
}

/*
 * Identities related to Ethernet Services
 */

identity control-mode {
    description
        "Base Identity for the type of control mode on Layer 2
        Control Protocol (L2CP).";
}

identity peer {
    base control-mode;
    description
        "'peer' mode, i.e., participate in the protocol towards the CE.
        Peering is common for Link Aggregation Control Protocol (LACP)
        and the Ethernet Local Management Interface (E-LMI) and,
        occasionally, for Link Layer Discovery Protocol (LLDP).
        For VPLSs and VPWSs, the subscriber can also request that the
        peer service provider enables spanning tree.";
}

identity tunnel {
    base control-mode;
    description
        "'tunnel' mode, i.e., pass to the egress or destination site. For
        Ethernet Private Lines (EPLs), the expectation is that L2CP
        frames are tunnelled.";
}
```

```
identity discard {
  base control-mode;
  description
    "'Discard' mode, i.e., discard the frame.";
}

identity neg-mode {
  description
    "Base identity for the negotiation mode.";
}

identity full-duplex {
  base neg-mode;
  description
    "Full-duplex negotiation mode.";
}

identity auto-neg {
  base neg-mode;
  description
    "Auto-negotiation mode.";
}

/***** Collection of VPN-related Types *****/

typedef vpn-id {
  type string;
  description
    "Defines an identifier that is used with a VPN module.
     This can be, for example, a service identifier, a node
     identifier, etc.";
}

/***** VPN-related reusable groupings *****/

grouping vpn-description {
  description
    "Provides common VPN information.";
  leaf vpn-id {
    type vpn-common:vpn-id;
    description
      "A VPN identifier that uniquely identifies a VPN.
       This identifier has a local meaning, e.g., within
       a service provider network.";
  }
  leaf vpn-name {
    type string;
    description

```

```
        "Used to associate a name with the service
        in order to facilitate the identification of
        the service.";
    }
    leaf vpn-description {
        type string;
        description
            "Textual description of a VPN.";
    }
    leaf customer-name {
        type string;
        description
            "Name of the customer that actually uses the VPN.";
    }
}

grouping vpn-profile-cfg {
    description
        "Grouping for VPN Profile configuration.";
    container valid-provider-identifiers {
        description
            "Container for valid provider profile identifiers.";
        list external-connectivity-identifier {
            if-feature "external-connectivity";
            key "id";
            description
                "List for profile identifiers that uniquely identify profiles
                governing how external connectivity is provided to a VPN.
                A profile indicates the type of external connectivity
                (Internet, cloud, etc.), the sites/nodes that are associated
                with a connectivity profile, etc. A profile can also indicate
                filtering rules and/or address translation rules. Such
                features may involve PE, P, or dedicated nodes as a function
                of the deployment.";
            leaf id {
                type string;
                description
                    "Identification of an external connectivity profile. The
                    profile only has significance within the service provider's
                    administrative domain.";
            }
        }
    }
    list encryption-profile-identifier {
        key "id";
        description
            "List for encryption profile identifiers.";
        leaf id {
            type string;
        }
    }
}
```

```
        description
            "Identification of the encryption profile to be used. The
             profile only has significance within the service provider's
             administrative domain.";
    }
}
list qos-profile-identifier {
    key "id";
    description
        "List for QoS Profile Identifiers.";
    leaf id {
        type string;
        description
            "Identification of the QoS profile to be used. The
             profile only has significance within the service provider's
             administrative domain.";
    }
}
list bfd-profile-identifier {
    key "id";
    description
        "List for BFD profile identifiers.";
    leaf id {
        type string;
        description
            "Identification of the BFD profile to be used. The
             profile only has significance within the service provider's
             administrative domain.";
    }
}
list forwarding-profile-identifier {
    key "id";
    description
        "List for forwarding profile identifiers.";
    leaf id {
        type string;
        description
            "Identification of the forwarding profile to be used.
             The profile only has significance within the service
             provider's administrative domain.";
    }
}
list routing-profile-identifier {
    key "id";
    description
        "List for Routing Profile Identifiers.";
    leaf id {
        type string;
```

```
        description
            "Identification of the routing profile to be used by the
            routing protocols within sites, vpn-network-accesses, or
            vpn-nodes for referring VRF's import/export policies.

            The profile only has significance within the service
            provider's administrative domain.";
    }
}
nacm:default-deny-write;
}
}

grouping oper-status-timestamp {
    description
        "This grouping defines some operational parameters for the
        service.";
    leaf status {
        type identityref {
            base operational-status;
        }
        config false;
        description
            "Operations status.";
    }
    leaf last-change {
        type yang:date-and-time;
        config false;
        description
            "Indicates the actual date and time of the service status
            change.";
    }
}

grouping service-status {
    description
        "Service status grouping.";
    container status {
        description
            "Service status.";
        container admin-status {
            description
                "Administrative service status.";
            leaf status {
                type identityref {
                    base administrative-status;
                }
                description

```

```
        "Administrative service status.";
    }
    leaf last-change {
        type yang:date-and-time;
        description
            "Indicates the actual date and time of the service status
            change.";
    }
}
container oper-status {
    description
        "Operational service status.";
    uses oper-status-timestamp;
}
}

grouping underlay-transport {
    description
        "This grouping defines the type of underlay transport for the
        VPN service or how that underlay is set. It can include an
        identifier to an abstract transport instance to which the VPN
        is grafted or indicate a technical implementation that is
        expressed as an ordered list of protocols.";
    choice type {
        description
            "A choice based on the type of underlay transport
            constraints.";
        case abstract {
            description
                "Indicates that the transport constraint is an abstract
                concept.";
            leaf transport-instance-id {
                type string;
                description
                    "An optional identifier of the abstract transport instance.";
            }
            leaf instance-type {
                type identityref {
                    base transport-instance-type;
                }
                description
                    "Indicates a transport instance type. For example, it can
                    be a VPN+, an IETF network slice, a virtual network, etc.";
            }
        }
        case protocol {
            description
```

```
        "Indicates a list of protocols.";
    leaf-list protocol {
        type identityref {
            base protocol-type;
        }
        ordered-by user;
        description
            "A client ordered list of transport protocols.";
    }
}
}
}

grouping vpn-route-targets {
    description
        "A grouping that specifies Route Target (RT) import-export rules
        used in a BGP-enabled VPN.";
    reference
        "RFC 4364: BGP/MPLS IP Virtual Private Networks (VPNs)
        RFC 4664: Framework for Layer 2 Virtual Private Networks
        (L2VPNs)";
    list vpn-target {
        key "id";
        description
            "Route targets. AND/OR operations may be defined
            based on the RTs assignment.";
        leaf id {
            type uint8;
            description
                "Identifies each VPN Target.";
        }
        list route-targets {
            key "route-target";
            description
                "List of RTs.";
            leaf route-target {
                type rt-types:route-target;
                description
                    "Conveys an RT value.";
            }
        }
    }
    leaf route-target-type {
        type rt-types:route-target-type;
        mandatory true;
        description
            "Import/export type of the RT.";
    }
}
```

```
container vpn-policies {
  description
    "VPN service policies. It contains references to the
    import and export policies to be associated with the
    VPN service.";
  leaf import-policy {
    type string;
    description
      "Identifies the 'import' policy.";
  }
  leaf export-policy {
    type string;
    description
      "Identifies the 'export' policy.";
  }
}

grouping route-distinguisher {
  description
    "Grouping for route distinguisher (RD).";
  choice rd-choice {
    description
      "Route distinguisher choice between several options
      on providing the route distinguisher value.";
    case directly-assigned {
      description
        "Explicitly assign an RD value.";
      leaf rd {
        type rt-types:route-distinguisher;
        description
          "Indicates an RD value that is explicitly
          assigned.";
      }
    }
    case directly-assigned-suffix {
      description
        "The value of the Assigned Number subfield of the RD.
        The Administrator subfield of the RD will be
        based on other configuration information such as
        router-id or ASN.";
      leaf rd-suffix {
        type uint16;
        description
          "Indicates the value of the Assigned Number
          subfield that is explicitly assigned.";
      }
    }
  }
}
```

```
case auto-assigned {
  description
    "The RD is auto-assigned.";
  container rd-auto {
    description
      "The RD is auto-assigned.";
    choice auto-mode {
      description
        "Indicates the auto-assignment mode. RD can be
        automatically assigned with or without
        indicating a pool from which the RD should be
        taken.

        For both cases, the server will auto-assign an RD
        value 'auto-assigned-rd' and use that value
        operationally.";
      case from-pool {
        leaf rd-pool-name {
          type string;
          description
            "The auto-assignment will be made from the pool
            identified by the rd-pool-name.";
        }
      }
      case full-auto {
        leaf auto {
          type empty;
          description
            "Indicates an RD is fully auto-assigned.";
        }
      }
    }
  }
  leaf auto-assigned-rd {
    type rt-types:route-distinguisher;
    config false;
    description
      "The value of the auto-assigned RD.";
  }
}

case auto-assigned-suffix {
  description
    "The value of the Assigned Number subfield will
    be auto-assigned. The Administrator subfield
    will be based on other configuration information such as
    router-id or ASN.";
  container rd-auto-suffix {
    description
```

```
    "The Assigned Number subfield is auto-assigned.";
  choice auto-mode {
    description
      "Indicates the auto-assignment mode of the Assigned Number
      subfield. This number can be automatically assigned
      with or without indicating a pool from which the value
      should be taken.

      For both cases, the server will auto-assign
      'auto-assigned-rd-suffix' and use that value to build
      the RD that will be used operationally.";
    case from-pool {
      leaf rd-pool-name {
        type string;
        description
          "The assignment will be made from the pool identified
          by the rd-pool-name.";
      }
    }
    case full-auto {
      leaf auto {
        type empty;
        description
          "Indicates that the Assigned Number is fully auto
          assigned.";
      }
    }
  }
  leaf auto-assigned-rd-suffix {
    type uint16;
    config false;
    description
      "Includes the value of the Assigned Number subfield that
      is auto-assigned .";
  }
}

case no-rd {
  description
    "Use the empty type to indicate RD has no value and is not to
    be auto-assigned.";
  leaf no-rd {
    type empty;
    description
      "No RD is assigned.";
  }
}
}
```

```
    }

    grouping vpn-components-group {
      description
        "Grouping definition to assign group-ids to associate VPN nodes,
        sites, or network accesses.";
      container groups {
        description
          "Lists the groups to which a VPN node, a site, or a network
          access belongs to.";
        list group {
          key "group-id";
          description
            "List of group-ids.";
          leaf group-id {
            type string;
            description
              "Is the group-id to which a VPN node, a site, or a network
              access belongs to.";
          }
        }
      }
    }

    grouping placement-constraints {
      description
        "Constraints for placing a network access.";
      list constraint {
        key "constraint-type";
        description
          "List of constraints.";
        leaf constraint-type {
          type identityref {
            base placement-diversity;
          }
          description
            "Diversity constraint type.";
        }
      }
      container target {
        description
          "The constraint will apply against this list of groups.";
        choice target-flavor {
          description
            "Choice for the group definition.";
          case id {
            list group {
              key "group-id";
              description

```

```
        "List of groups.";
        leaf group-id {
            type string;
            description
                "The constraint will apply against this particular
                group-id.";
        }
    }
}
case all-accesses {
    leaf all-other-accesses {
        type empty;
        description
            "The constraint will apply against all other network
            accesses of a site.";
    }
}
case all-groups {
    leaf all-other-groups {
        type empty;
        description
            "The constraint will apply against all other groups that
            the customer is managing.";
    }
}
}
}
}

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

```
        description
            "Destination port definition.";
        uses packet-fields:port-range-or-operator;
    }
}

grouping qos-classification-policy {
    description
        "Configuration of the traffic classification policy.";
    list rule {
        key "id";
        ordered-by user;
        description
            "List of marking rules.";
        leaf id {
            type string;
            description
                "An identifier of the QoS classification policy rule.";
        }
        choice match-type {
            default "match-flow";
            description
                "Choice for classification.";
            case match-flow {
                choice l3 {
                    description
                        "Either IPv4 or IPv6.";
                    container ipv4 {
                        description
                            "Rule set that matches IPv4 header.";
                        uses packet-fields:acl-ip-header-fields;
                        uses packet-fields:acl-ipv4-header-fields;
                    }
                    container ipv6 {
                        description
                            "Rule set that matches IPv6 header.";
                        uses packet-fields:acl-ip-header-fields;
                        uses packet-fields:acl-ipv6-header-fields;
                    }
                }
            }
            choice l4 {
                description
                    "Includes Layer 4 specific information.
                     This version focuses on TCP and UDP.";
                container tcp {
                    description
                        "Rule set that matches TCP header.";
```

```

        uses packet-fields:acl-tcp-header-fields;
        uses ports;
    }
    container udp {
        description
            "Rule set that matches UDP header.";
        uses packet-fields:acl-udp-header-fields;
        uses ports;
    }
}

case match-application {
    leaf match-application {
        type identityref {
            base customer-application;
        }
        description
            "Defines the application to match.";
    }
}

leaf target-class-id {
    if-feature "qos";
    type string;
    description
        "Identification of the class of service. This identifier is
        internal to the administration.";
}
}

}

}
}

<CODE ENDS>

```

5. Security Considerations

The YANG modules specified in this document define schemas for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The 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-vpn-common" module defines a set of identities, types, and groupings. These nodes are intended to be reused by other YANG modules. The module does not expose by itself any data nodes which are writable, contain read-only state, or RPCs. As such, there are no additional security issues to be considered relating to the "ietf-vpn-common" module.

Modules that use the groupings that are defined in this document should identify the corresponding security considerations. For example, reusing some of these groupings will expose privacy-related information (e.g., customer-name). Disclosing such information may be considered as a violation of the customer-provider trust relationship.

6. 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-vpn-common
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-vpn-common
namespace: urn:ietf:params:xml:ns:yang:ietf-vpn-common
maintained by IANA: N
prefix: vpn-common
reference: RFC XXXX
```

7. Acknowledgements

During the discussions of this work, helpful comments and reviews were received from (listed alphabetically): Alejandro Aguado, Raul Arco, Miguel Cros Cecilia, Joe Clarke, Dhruv Dhody, Adrian Farrel, Roque Gagliano, Christian Jacquenet, Kireeti Kompella, Julian Lucek, Tom Petch, Erez Segev, and Paul Sherratt. Many thanks to them.

This work is partially supported by the European Commission under Horizon 2020 grant agreement number 101015857 Secured autonomic traffic management for a Tera of SDN flows (Teraflow).

Many thanks to Radek Krejci for the yangdoctors review, Wesley Eddy for the tsvalt review, Ron Bonica and Victoria Pritchard for the Rtmdir review, Joel Halpern for the genart review, Tim Wicinski for the opmdir review, and Suresh Krishnan for the intdir review.

Special thanks to Robert Wilton for the AD review.

Thanks to Roman Danyliw, Lars Eagert, Warren Kumari, Erik Kline, Zaheduzzaman Sarker, Benjamin Kaduk, and Eric Vyncke for the IESG review.

8. Contributors

Italo Busi
Huawei Technologies
Email: Italo.Busi@huawei.com

Luis Angel Munoz
Vodafone
Email: luis-angel.munoz@vodafone.com

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

9. References

9.1. Normative References

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8294] Liu, X., Qu, Y., Lindem, A., Hopps, C., and L. Berger, "Common YANG Data Types for the Routing Area", RFC 8294, DOI 10.17487/RFC8294, December 2017, <<https://www.rfc-editor.org/info/rfc8294>>.
- [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>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [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>>.

9.2. Informative References

- [I-D.ietf-opsawg-l2nm]
Barguil, S., Dios, O. G. D., Boucadair, M., and L. A. Munoz, "A Layer 2 VPN Network YANG Model", Work in Progress, Internet-Draft, draft-ietf-opsawg-l2nm-06, 12 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-opsawg-l2nm-06.txt>>.

[I-D.ietf-opsawg-l3sm-l3nm]

Barguil, S., Dios, O. G. D., Boucadair, M., Munoz, L. A., and A. Aguado, "A Layer 3 VPN Network YANG Model", Work in Progress, Internet-Draft, draft-ietf-opsawg-l3sm-l3nm-15, 28 September 2021, <<https://www.ietf.org/archive/id/draft-ietf-opsawg-l3sm-l3nm-15.txt>>.

[I-D.ietf-teas-actn-vn-yang]

Lee, Y., Dhody, D., Ceccarelli, D., Bryskin, I., and B. Y. Yoon, "A YANG Data Model for VN Operation", Work in Progress, Internet-Draft, draft-ietf-teas-actn-vn-yang-12, 25 August 2021, <<https://www.ietf.org/archive/id/draft-ietf-teas-actn-vn-yang-12.txt>>.

[I-D.ietf-teas-enhanced-vpn]

Dong, J., Bryant, S., Li, Z., Miyasaka, T., and Y. Lee, "A Framework for Enhanced Virtual Private Network (VPN+) Services", Work in Progress, Internet-Draft, draft-ietf-teas-enhanced-vpn-08, 12 July 2021, <<https://www.ietf.org/archive/id/draft-ietf-teas-enhanced-vpn-08.txt>>.

[I-D.ietf-teas-ietf-network-slices]

Farrel, A., Gray, E., Drake, J., Rokui, R., Homma, S., Makhijani, K., Contreras, L. M., and J. Tantsura, "Framework for IETF Network Slices", Work in Progress, Internet-Draft, draft-ietf-teas-ietf-network-slices-04, 23 August 2021, <<https://www.ietf.org/archive/id/draft-ietf-teas-ietf-network-slices-04.txt>>.

[IEEE802.1ad]

"Virtual Bridged Local Area Networks Amendment 4: Provider Bridges", IEEE Std 802.1ad-2005, 2006.

[IEEE802.1AX]

"Link Aggregation", IEEE Std 802.1AX-2020, 2020.

[IEEE802.1Q]

"Bridges and Bridged Networks", IEEE Std 802.1Q-2018, 6 July 2018.

[ISO10589]

ISO, "Intermediate System to Intermediate System intra-domain routing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473)", 2002, <International Standard 10589:2002, Second Edition>.

- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://www.rfc-editor.org/info/rfc791>>.
- [RFC1112] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, DOI 10.17487/RFC1112, August 1989, <<https://www.rfc-editor.org/info/rfc1112>>.
- [RFC1701] Hanks, S., Li, T., Farinacci, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 1701, DOI 10.17487/RFC1701, October 1994, <<https://www.rfc-editor.org/info/rfc1701>>.
- [RFC1702] Hanks, S., Li, T., Farinacci, D., and P. Traina, "Generic Routing Encapsulation over IPv4 networks", RFC 1702, DOI 10.17487/RFC1702, October 1994, <<https://www.rfc-editor.org/info/rfc1702>>.
- [RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, DOI 10.17487/RFC2003, October 1996, <<https://www.rfc-editor.org/info/rfc2003>>.
- [RFC2080] Malkin, G. and R. Minnear, "RIPng for IPv6", RFC 2080, DOI 10.17487/RFC2080, January 1997, <<https://www.rfc-editor.org/info/rfc2080>>.
- [RFC2236] Fenner, W., "Internet Group Management Protocol, Version 2", RFC 2236, DOI 10.17487/RFC2236, November 1997, <<https://www.rfc-editor.org/info/rfc2236>>.
- [RFC2453] Malkin, G., "RIP Version 2", STD 56, RFC 2453, DOI 10.17487/RFC2453, November 1998, <<https://www.rfc-editor.org/info/rfc2453>>.
- [RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, DOI 10.17487/RFC2473, December 1998, <<https://www.rfc-editor.org/info/rfc2473>>.
- [RFC2710] Deering, S., Fenner, W., and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, DOI 10.17487/RFC2710, October 1999, <<https://www.rfc-editor.org/info/rfc2710>>.
- [RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, DOI 10.17487/RFC3209, December 2001, <<https://www.rfc-editor.org/info/rfc3209>>.

- [RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, DOI 10.17487/RFC3376, October 2002, <<https://www.rfc-editor.org/info/rfc3376>>.
- [RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <<https://www.rfc-editor.org/info/rfc3810>>.
- [RFC3931] Lau, J., Ed., Townsley, M., Ed., and I. Goyret, Ed., "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC 3931, DOI 10.17487/RFC3931, March 2005, <<https://www.rfc-editor.org/info/rfc3931>>.
- [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>>.
- [RFC4577] Rosen, E., Psenak, P., and P. Pillay-Esnault, "OSPF as the Provider/Customer Edge Protocol for BGP/MPLS IP Virtual Private Networks (VPNs)", RFC 4577, DOI 10.17487/RFC4577, June 2006, <<https://www.rfc-editor.org/info/rfc4577>>.
- [RFC4664] Andersson, L., Ed. and E. Rosen, Ed., "Framework for Layer 2 Virtual Private Networks (L2VPNs)", RFC 4664, DOI 10.17487/RFC4664, September 2006, <<https://www.rfc-editor.org/info/rfc4664>>.
- [RFC4761] Kompella, K., Ed. and Y. Rekhter, Ed., "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling", RFC 4761, DOI 10.17487/RFC4761, January 2007, <<https://www.rfc-editor.org/info/rfc4761>>.

- [RFC4762] Lasserre, M., Ed. and V. Kompella, Ed., "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling", RFC 4762, DOI 10.17487/RFC4762, January 2007, <<https://www.rfc-editor.org/info/rfc4762>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC5036] Andersson, L., Ed., Minei, I., Ed., and B. Thomas, Ed., "LDP Specification", RFC 5036, DOI 10.17487/RFC5036, October 2007, <<https://www.rfc-editor.org/info/rfc5036>>.
- [RFC5798] Nadas, S., Ed., "Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6", RFC 5798, DOI 10.17487/RFC5798, March 2010, <<https://www.rfc-editor.org/info/rfc5798>>.
- [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>>.
- [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>>.
- [RFC6565] Pillay-Esnault, P., Moyer, P., Doyle, J., Ertekin, E., and M. Lundberg, "OSPFv3 as a Provider Edge to Customer Edge (PE-CE) Routing Protocol", RFC 6565, DOI 10.17487/RFC6565, June 2012, <<https://www.rfc-editor.org/info/rfc6565>>.
- [RFC6624] Kompella, K., Kothari, B., and R. Cherukuri, "Layer 2 Virtual Private Networks Using BGP for Auto-Discovery and Signaling", RFC 6624, DOI 10.17487/RFC6624, May 2012, <<https://www.rfc-editor.org/info/rfc6624>>.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, DOI 10.17487/RFC7348, August 2014, <<https://www.rfc-editor.org/info/rfc7348>>.
- [RFC7432] Sajassi, A., Ed., Aggarwal, R., Bitar, N., Isaac, A., Uttaro, J., Drake, J., and W. Henderickx, "BGP MPLS-Based Ethernet VPN", RFC 7432, DOI 10.17487/RFC7432, February 2015, <<https://www.rfc-editor.org/info/rfc7432>>.

- [RFC7510] Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black, "Encapsulating MPLS in UDP", RFC 7510, DOI 10.17487/RFC7510, April 2015, <<https://www.rfc-editor.org/info/rfc7510>>.
- [RFC7623] Sajassi, A., Ed., Salam, S., Bitar, N., Isaac, A., and W. Henderickx, "Provider Backbone Bridging Combined with Ethernet VPN (PBB-EVPN)", RFC 7623, DOI 10.17487/RFC7623, September 2015, <<https://www.rfc-editor.org/info/rfc7623>>.
- [RFC7676] Pignataro, C., Bonica, R., and S. Krishnan, "IPv6 Support for Generic Routing Encapsulation (GRE)", RFC 7676, DOI 10.17487/RFC7676, October 2015, <<https://www.rfc-editor.org/info/rfc7676>>.
- [RFC7761] Fenner, B., Handley, M., Holbrook, H., Kouvelas, I., Parekh, R., Zhang, Z., and L. Zheng, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", STD 83, RFC 7761, DOI 10.17487/RFC7761, March 2016, <<https://www.rfc-editor.org/info/rfc7761>>.
- [RFC7880] Pignataro, C., Ward, D., Akiya, N., Bhatia, M., and S. Pallagatti, "Seamless Bidirectional Forwarding Detection (S-BFD)", RFC 7880, DOI 10.17487/RFC7880, July 2016, <<https://www.rfc-editor.org/info/rfc7880>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8214] Boutros, S., Sajassi, A., Salam, S., Drake, J., and J. Rabadan, "Virtual Private Wire Service Support in Ethernet VPN", RFC 8214, DOI 10.17487/RFC8214, August 2017, <<https://www.rfc-editor.org/info/rfc8214>>.
- [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>>.
- [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>>.

- [RFC8365] Sajassi, A., Ed., Drake, J., Ed., Bitar, N., Shekhar, R., Uttaro, J., and W. Henderickx, "A Network Virtualization Overlay Solution Using Ethernet VPN (EVPN)", RFC 8365, DOI 10.17487/RFC8365, March 2018, <<https://www.rfc-editor.org/info/rfc8365>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8660] Bashandy, A., Ed., Filsfils, C., Ed., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing with the MPLS Data Plane", RFC 8660, DOI 10.17487/RFC8660, December 2019, <<https://www.rfc-editor.org/info/rfc8660>>.
- [RFC8663] Xu, X., Bryant, S., Farrel, A., Hassan, S., Henderickx, W., and Z. Li, "MPLS Segment Routing over IP", RFC 8663, DOI 10.17487/RFC8663, December 2019, <<https://www.rfc-editor.org/info/rfc8663>>.
- [RFC8754] Filsfils, C., Ed., Dukes, D., Ed., Previdi, S., Leddy, J., Matsushima, S., and D. Voyer, "IPv6 Segment Routing Header (SRH)", RFC 8754, DOI 10.17487/RFC8754, March 2020, <<https://www.rfc-editor.org/info/rfc8754>>.
- [RFC8926] Gross, J., Ed., Ganga, I., Ed., and T. Sridhar, Ed., "Geneve: Generic Network Virtualization Encapsulation", RFC 8926, DOI 10.17487/RFC8926, November 2020, <<https://www.rfc-editor.org/info/rfc8926>>.

Appendix A. Example of Common Data Nodes in Early L2NM/L3NM Designs

In order to avoid data nodes duplication and to ease passing data among layers (i.e., from the service layer to the network layer and vice versa), early versions of the L3NM reused many of the data nodes that are defined in the L3SM. Nevertheless, that approach was abandoned because that design was interpreted as if the deployment of L3NM depends on L3SM, while this is not required. For example, a service provider may decide to use the L3NM to build its L3VPN services without exposing the L3SM to customers.

Likewise, early versions of the L2NM reused many of the data nodes that are defined in both L2SM and L3NM. An example of L3NM groupings reused in L2NM is shown in Figure 5. Such data nodes reuse was interpreted as if the deployment of the L2NM requires the support of the L3NM; which is not required.

```
module ietf-l2vpn-ntw {
  ...
  import ietf-l3vpn-ntw {
    prefix l3vpn-ntw;
    reference
      "RFC NNNN: A Layer 3 VPN Network YANG Model";
  }
  ...
  container l2vpn-ntw {
    ...
    container vpn-services {
      list vpn-service {
        ...
        uses l3vpn-ntw:service-status;
        uses l3vpn-ntw:svc-transport-encapsulation;
        ...
      }
    }
    ...
  }
}
```

Figure 5: Excerpt from the L2NM YANG Module

Authors' Addresses

Samier Barguil
Telefonica
Madrid
Spain

Email: samier.barguilgiraldo.ext@telefonica.com

Oscar Gonzalez de Dios (editor)
Telefonica
Madrid
Spain

Email: oscar.gonzalezdedios@telefonica.com

Mohamed Boucadair (editor)
Orange
France

Email: mohamed.boucadair@orange.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China

Email: bill.wu@huawei.com

OPSAWG Working Group
Internet-Draft
Intended status: Standards Track
Expires: 6 November 2022

B. Wu, Ed.
Q. Wu, Ed.
Huawei
M. Boucadair, Ed.
Orange
O. Gonzalez de Dios
Telefonica
B. Wen
Comcast
5 May 2022

A YANG Model for Network and VPN Service Performance Monitoring
draft-ietf-opsawg-yang-vpn-service-pm-08

Abstract

The data model for network topologies defined in RFC 8345 introduces vertical layering relationships between networks that can be augmented to cover network and service topologies. This document defines a YANG module for performance monitoring (PM) of both networks and VPN services that can be used to monitor and manage network performance on the topology at higher layer or the service topology between VPN sites.

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 6 November 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Terminology	3
2.1. Acronyms	3
3. Network and VPN Service Performance Monitoring Model Usage .	4
3.1. Collecting Data via Pub/Sub Mechanism	5
3.2. Collecting Data On-demand	6
4. Description of The Data Model	6
4.1. Layering Relationship between Multiple Layers of Topology	6
4.2. Network Level	9
4.3. Node Level	9
4.4. Link and Termination Point Level	10
5. Network and VPN Service Performance Monitoring YANG Module .	14
6. Security Considerations	29
7. IANA Considerations	30
8. Acknowledgements	31
9. Contributors	31
10. References	31
10.1. Normative References	31
10.2. Informative References	34
Appendix A. Illustrative Examples	35
A.1. VPN Performance Subscription Example	35
A.2. Example of VPN Performance Snapshot	37
A.3. Example of Percentile Monitoring	39
Authors' Addresses	39

1. Introduction

[RFC8969] describes a framework for automating service and network management with YANG [RFC6020] models. It defines that the performance measurement telemetry model should be tied to the services (such as a Layer 3 VPN or Layer 2 VPN) or to the network models to monitor the overall network performance and the Service Level Agreements (SLAs).

The performance of VPN services is associated with the performance changes of the underlay network that carries VPN services, such as the delay of the underlay tunnels and the packet loss status of the device interfaces. Additionally, the integration of Layer 2/Layer 3 VPN performance and network performance data enables the orchestrator to subscribe to VPN service performance in a unified manner. Therefore, this document defines a YANG module for both network and VPN service performance monitoring (PM). The module can be used to monitor and manage network performance on the topology level or the service topology between VPN sites, in particular.

This document does not introduce new metrics for network performance or mechanisms for measuring network performance, but uses the existing mechanisms and statistics to display the performance monitoring statistics at the network and service layers. All these metrics are defined as unidirectional metrics.

The YANG module defined in this document is designed as an augmentation to the network topology YANG model defined in [RFC8345] and draws on relevant YANG types defined in [RFC6991], [RFC8345], [RFC8532], and [RFC9181].

Appendix A provides a set of examples to illustrate the use of the module.

2. Terminology

The following terms are defined in [RFC7950] and are used in this specification:

- * augment
- * data model
- * data node

The terminology for describing YANG data models is found in [RFC7950].

The tree diagrams used in this document follow the notation defined in [RFC8340].

2.1. Acronyms

The following acronyms are used in the document:

L2VPN	Layer 2 Virtual Private Network
L3VPN	Layer 3 Virtual Private Network

L2NM	L2VPN Network Model
L3NM	L3VPN Network Model
MPLS	Multiprotocol Label Switching
OAM	Operations, Administration, and Maintenance
OWAMP	One-Way Active Measurement Protocol
PE	Provider Edge
PM	Performance Monitoring
SLA	Service Level Agreement
TP	Termination Point, as defined in [RFC8345] section 4.2
TWAMP	Two-Way Active Measurement Protocol
VPLS	Virtual Private LAN Service
VPN	Virtual Private Network

3. Network and VPN Service Performance Monitoring Model Usage

Models are key for automating network management operations. According to [RFC8969], together with service and network models, performance measurement telemetry models are needed to monitor network performance to meet specific service requirements (typically captured in an SLA).

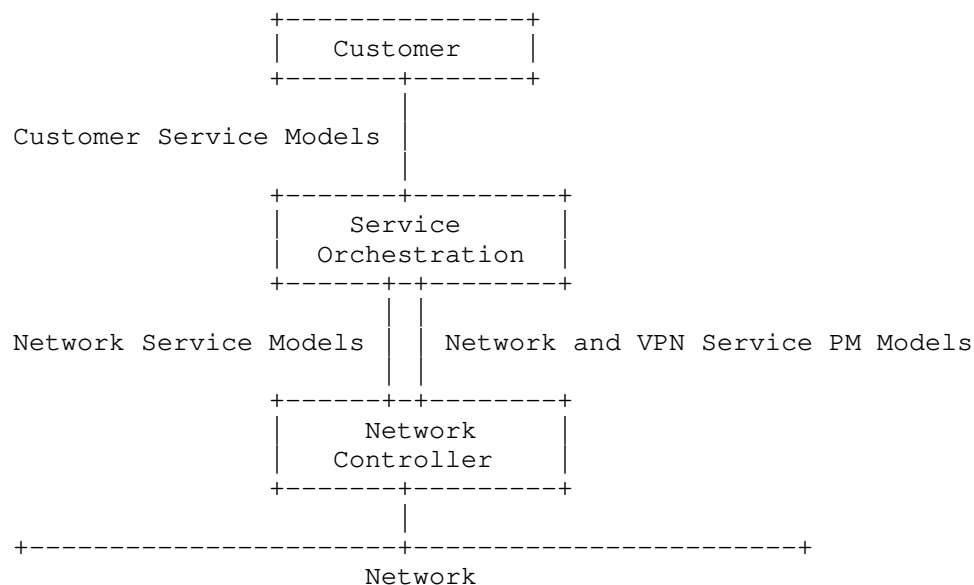


Figure 1: Reference Architecture

As shown in Figure 1, in the context of the layering model architecture described in [RFC8309], the network and VPN service performance monitoring (PM) model can be used to expose a set of

performance information to the above layer. Such information can be used by an orchestrator to subscribe to performance data. The network controller will then notify the orchestrator about corresponding parameter changes.

Before using the model, the controller needs to establish complete topology visibility of the network and VPN. For example, the controller can use network information from [RFC8345], [I-D.ietf-opsawg-sap] or VPN information from [RFC9182], [I-D.ietf-opsawg-l2nm]. Then the controller derives network or VPN level performance data by aggregating (and filtering) lower-level data collected via monitoring counters of the involved devices.

The network or VPN performance data can be based on different sources. For example, the performance monitoring data per link in the underlying network can be collected using a network performance measurement method such as One-Way Active Measurement Protocol (OWAMP) [RFC4656], Two-Way Active Measurement Protocol (TWAMP) [RFC5357], Simple Two-way Active Measurement Protocol (STAMP) [RFC8762], and Multiprotocol Label Switching (MPLS) Loss and Delay Measurement [RFC6374]. The performance monitoring information reflecting the quality of the network or VPN service (e.g., end-to-end network performance data between source node and destination node in the network or between VPN sites) can be computed and aggregated, for example, using the information from the Traffic Engineering Database (TED), [RFC7471] [RFC8570] [RFC8571] or LMAP [RFC8194].

The measurement and report intervals that are associated with these performance data usually depend on the configuration of the specific measurement method or collection method or various combinations. This document defines a network-wide measurement interval to align measurement requirements for networks or VPN services.

In addition, the amount of performance data collected from the devices can be huge. To avoid receiving a large amount of operational data of VPN instances, VPN interfaces, or tunnels, the network controller can specifically subscribe to metric-specific data using the tagging methods defined in [I-D.ietf-netmod-node-tags].

3.1. Collecting Data via Pub/Sub Mechanism

Some applications such as service-assurance applications, which must maintain a continuous view of operational data and state, can use the subscription model specified in [RFC8641] to subscribe to the specific network performance data or VPN service performance data they are interested in, at the data source. For example, network or VPN topology updates may be obtained through on-change notifications [RFC8641]. For dynamic PM data, various notifications can be

specified to obtain more complete data. A periodic notification [RFC8641] can be specified to obtain real-time performance data, a replay notification defined in [RFC5277] or [RFC8639] can be specified to obtain historical data, or alarm notifications [RFC8632] can be specified to get alarms for the metrics which exceed or fall below the performance threshold.

The data source can, then, use the network and VPN service assurance model defined in this document and the YANG Push model [RFC8641] to distribute specific telemetry data to target recipients.

3.2. Collecting Data On-demand

To obtain a snapshot of a large amount of performance data from a network topology or VPN network, service-assurance applications may retrieve information using the network and VPN service PM model through a NETCONF [RFC6241] or a RESTCONF [RFC8040] interface. For example, a specified "link-id" of a VPN can be used as a filter in a RESTCONF GET request to retrieve per-link VPN PM data.

4. Description of The Data Model

This document defines the YANG module, "ietf-network-vpn-pm", which is an augmentation to the "ietf-network" and "ietf-network-topology" modules.

The performance monitoring data augments the service topology as shown in Figure 2.

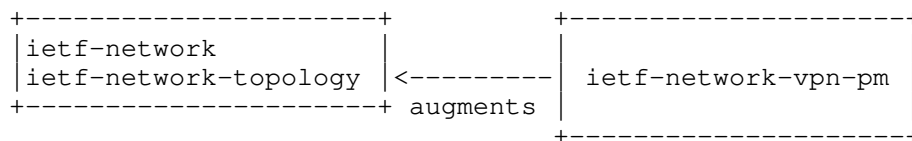


Figure 2: Module Augmentation

4.1. Layering Relationship between Multiple Layers of Topology

[RFC8345] defines a YANG data model for network/service topologies and inventories. The service topology described in [RFC8345] includes the virtual topology for a service layer above Layer 1 (L1), Layer 2 (L2), and Layer 3 (L3). This service topology has the generic topology elements of node, link, and terminating point. One typical example of a service topology is described in Figure 3 of [RFC8345]: two VPN service topologies instantiated over a common L3 topology. Each VPN service topology is mapped onto a subset of nodes from the common L3 topology.

Figure 3 illustrates an example of a topology that maps between the VPN service topology and an underlying network:

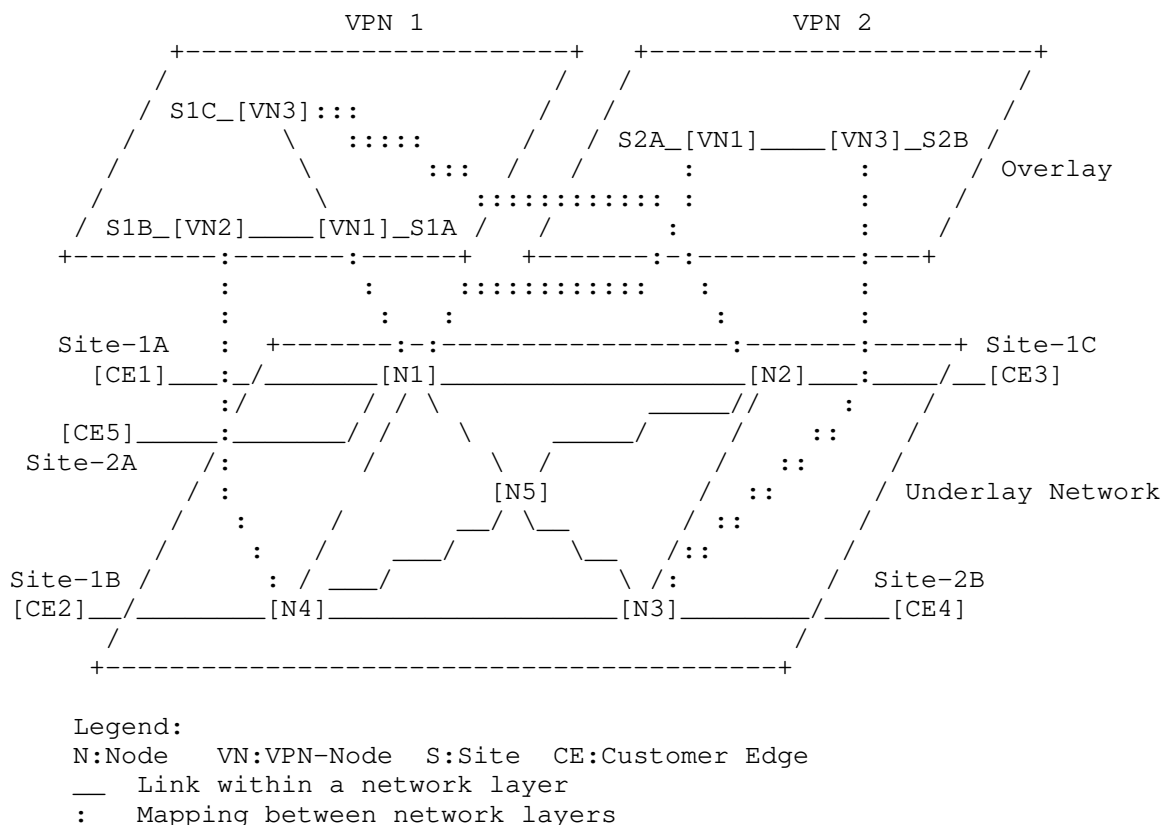


Figure 3: Example of Topology Mapping Between VPN Service Topology and Underlying Network

As shown in Figure 3, two VPN services topologies are built on top of one common underlying physical network:

VPN 1: This service topology supports hub-spoke communications for 'customer 1' connecting the customer's access at three sites: 'Site-1A', 'Site-1B', and 'Site-1C'. These sites are connected to nodes that are mapped to node 1 (N1), node 2 (N2), and node 4 (N4) in the underlying physical network. 'Site-1A' plays the role of hub while 'Site-1B' and 'Site-1C' are configured as spoke.

VPN 2: This service supports any-to-any communications for 'customer

2' connecting the customer's access at two sites: 'Site-2A' and 'Site-2B'. These sites are connected to nodes that are mapped to nodes 1 (N1) and node 3 (N3)5 in the underlying physical network. 'Site-2A' and 'Site-2B' have 'any-to-any' role.

Apart from the association between the VPN topology and the underlay topology, VPN Network PM can also provide the performance status of the underlay network and VPN services. For example, network PM can provide link PM statistics and port statistics. VPN PM can provide statistics on VPN access interfaces, the number of current VRF routes or L2VPN MAC entry of VPN nodes, and performance statistics on the logical point-to-point link between source and destination VPN nodes or between source and destination VPN access interfaces. Figure 4 illustrates an example of VPN PM and the difference between two VPN PM measurement methods. One is the VPN tunnel PM and the other is inter-VPN-access interface PM.

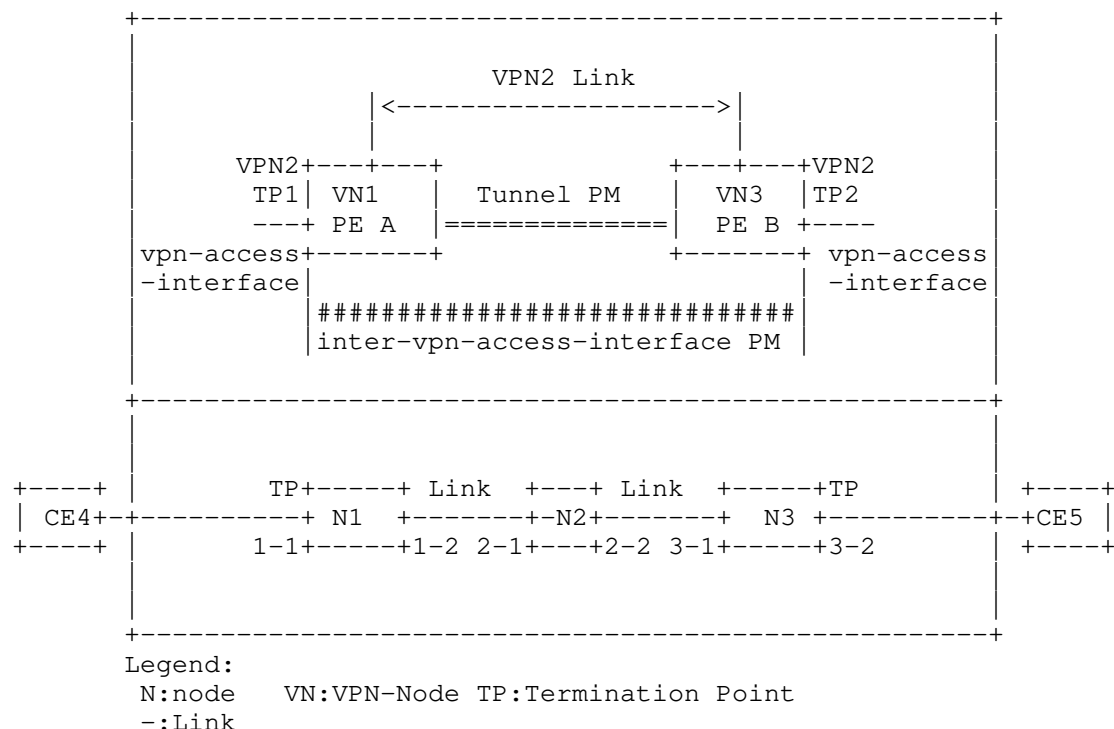


Figure 4: An Example of VPN PM

4.2. Network Level

For network performance monitoring, the container of "networks" in [RFC8345] does not need to be extended.

For VPN service performance monitoring, the container "service-type" is defined to indicate the VPN type, e.g., L3VPN or Virtual Private LAN Service (VPLS). The values are taken from [RFC9181]. When a network topology instance contains the L3VPN or other L2VPN network type, it represents a VPN instance that can perform performance monitoring.

The tree in Figure 5 is a part of ietf-network-vpn-pm tree. It defines the following set of network level attributes:

"vpn-id": Refers to an identifier of VPN service defined in [RFC9181]. This identifier is used to correlate the performance status with the network service configuration.

"vpn-service-topology": Indicates the type of the VPN topology. This model supports "any-to-any", "Hub and Spoke" (where Hubs can exchange traffic), and "Hub and Spoke disjoint" (where Hubs cannot exchange traffic) that are taken from [RFC9181]. These VPN topology types can be used to describe how VPN sites communicate with each other.

```
module: ietf-network-vpn-pm
  augment /nw:networks/nw:network/nw:network-types:
    +--rw service-type!
      +--rw service-type? identityref
  augment /nw:networks/nw:network:
    +--rw vpn-pm-attributes
      +--rw vpn-id?          vpn-common:vpn-id
      +--rw vpn-service-topology? identityref
```

Figure 5: Network Level YANG Tree of the Hierarchies

4.3. Node Level

The tree in Figure 6 is the node part of ietf-network-vpn-pm tree.

For network performance monitoring, a container of "pm-attributes" is augmented to the list of "node" that are defined in [RFC8345]. The container includes the following attributes:

"node-type": Indicates the device type of Provider Edge (PE),

Provider (P) device, or Autonomous System Border Router (ASBR) as defined in [RFC4026] and [RFC4364], so that the performance metric between any two nodes each with specific node type can be reported.

"entry-summary": Lists a set of IPv4 statistics, IPv6 statistics, and MAC statistics. The detailed statistics are specified separately.

For VPN service performance monitoring, the model defines one attribute:

"role": Defines the role in a particular VPN service topology. The roles are taken from [RFC9181] (e.g., any-to-any-role, spoke-role, hub-role).

```
augment /nw:networks/nw:network/nw:node:
  +--rw pm-attributes
    +--rw node-type?          identityref
    +--ro entry-summary
      +--ro ipv4-num
        +--ro maximum-routes?    uint32
        +--ro total-active-routes? uint32
      +--ro ipv6-num
        +--ro maximum-routes?    uint32
        +--ro total-active-routes? uint32
      +--ro mac-num
        +--ro mac-num-limit?      uint32
        +--ro total-active-mac-num? uint32
    +--rw role?                identityref
```

Figure 6: Node Level YANG Tree of the Hierarchies

4.4. Link and Termination Point Level

The tree in Figure 7 is the link and termination point (TP) part of ietf-network-vpn-pm tree.

The 'links' are classified into two types: topology link defined in [RFC8345] and abstract link of a VPN between PEs defined in this module.

The performance data of a link is a collection of counters and gauges that report the performance status.

```

augment /nw:networks/nw:network/nt:link:
  +---rw pm-attributes
    +---rw low-percentile?                percentile
    +---rw intermediate-percentile?       percentile
    +---rw high-percentile?               percentile
    +---rw measurement-interval?          uint32
    +---ro start-time?                    yang:date-and-time
    +---ro end-time?                      yang:date-and-time
    +---ro pm-source?                     identityref
    +---ro one-way-pm-statistics
      +---ro loss-statistics
        +---ro packet-loss-count?         yang:counter64
        +---ro loss-ratio?                 percentage
      +---ro delay-statistics
        +---ro unit-value?                 identityref
        +---ro min-delay-value?            yang:gauge64
        +---ro max-delay-value?            yang:gauge64
        +---ro low-delay-percentile?       yang:gauge64
        +---ro intermediate-delay-percentile? yang:gauge64
        +---ro high-delay-percentile?      yang:gauge64
      +---ro jitter-statistics
        +---ro unit-value?                 identityref
        +---ro min-jitter-value?           yang:gauge64
        +---ro max-jitter-value?           yang:gauge64
        +---ro low-jitter-percentile?      yang:gauge64
        +---ro intermediate-jitter-percentile? yang:gauge64
        +---ro high-jitter-percentile?     yang:gauge64
    +---ro one-way-pm-statistics-per-class* [class-id]
      +---ro class-id                     string
      +---ro loss-statistics
        +---ro packet-loss-count?         yang:counter64
        +---ro loss-ratio?                 percentage
      +---ro delay-statistics
        +---ro unit-value?                 identityref
        +---ro min-delay-value?            yang:gauge64
        +---ro max-delay-value?            yang:gauge64
        +---ro low-delay-percentile?       yang:gauge64
        +---ro intermediate-delay-percentile? yang:gauge64
        +---ro high-delay-percentile?      yang:gauge64
      +---ro jitter-statistics
        +---ro unit-value?                 identityref
        +---ro min-jitter-value?           yang:gauge64
        +---ro max-jitter-value?           yang:gauge64
        +---ro low-jitter-percentile?      yang:gauge64
        +---ro intermediate-jitter-percentile? yang:gauge64
        +---ro high-jitter-percentile?     yang:gauge64
  +---rw vpn-pm-type
    +---rw inter-vpn-access-interface

```

```

    |   +--rw inter-vpn-access-interface?   empty
    +--rw underlay-tunnel!
        +--ro vpn-underlay-transport-type?   identityref
augment /nw:networks/nw:network/nw:node/nt:termination-point:
+--ro pm-statistics
+--ro reference-time?           yang:date-and-time
+--ro inbound-octets?           yang:counter64
+--ro inbound-unicast?          yang:counter64
+--ro inbound-nunicast?         yang:counter64
+--ro inbound-discards?         yang:counter64
+--ro inbound-errors?           yang:counter64
+--ro inbound-unknown-protocol? yang:counter64
+--ro outbound-octets?          yang:counter64
+--ro outbound-unicast?         yang:counter64
+--ro outbound-nunicast?        yang:counter64
+--ro outbound-discards?        yang:counter64
+--ro outbound-errors?          yang:counter64
+--ro vpn-network-access* [network-access-id]
+--ro network-access-id         vpn-common:vpn-id
+--ro reference-time?           yang:date-and-time
+--ro inbound-octets?           yang:counter64
+--ro inbound-unicast?          yang:counter64
+--ro inbound-nunicast?         yang:counter64
+--ro inbound-discards?         yang:counter64
+--ro inbound-errors?           yang:counter64
+--ro inbound-unknown-protocol? yang:counter64
+--ro outbound-octets?          yang:counter64
+--ro outbound-unicast?         yang:counter64
+--ro outbound-nunicast?        yang:counter64
+--ro outbound-discards?        yang:counter64
+--ro outbound-errors?          yang:counter64

```

Figure 7: Link and Termination point Level YANG Tree of the hierarchies

For the data nodes of 'link' depicted in Figure 7, the YANG module defines the following minimal set of link-level performance attributes:

Percentile parameters: The module supports reporting delay and jitter metric by percentile values. By default, low percentile (10th percentile), intermediate percentile (50th percentile), high percentile (90th percentile) are used. Setting a percentile to 0.00 indicates the client is not interested in receiving particular percentile. If all percentile nodes are set to 0.00, this represents that no percentile related nodes will be reported for a given performance metric (e.g., one-way delay, one-way delay variation) and only peak/min values will be reported. For

example, a client can inform the server that it is interested in receiving only high percentiles. Then for a given link, at a given "start-time", "end-time" and "measurement-interval", the 'high-delay-percentile' and 'high-jitter-percentile' will be reported. An example to illustrate the use of percentiles is provided in Appendix A.3.

PM source ("pm-source"): Indicates the performance monitoring source. The data for the topology link can be based, e.g., on BGP-LS [RFC8571]. The statistics of the VPN abstract links can be collected based upon VPN OAM mechanisms, e.g., OAM mechanisms referenced in [RFC9182], or Ethernet service OAM [ITU-T-Y-1731] referenced in [I-D.ietf-opsawg-l2nm]. Alternatively, the data can be based upon the underlay technology OAM mechanisms, for example, Generic Routing Encapsulation (GRE) tunnel OAM.

Measurement interval ("measurement-interval"): Specifies the performance measurement interval, in seconds.

Start time ("start-time"): Indicates the start time of the performance measurement for link statistics.

End time ("end-time"): Indicates the end time of the performance measurement for link statistics.

Reference time ("reference-time"): Indicates the timestamp when the counters are obtained.

Loss statistics: A set of one-way loss statistics attributes that are used to measure end to end loss between VPN sites or between any two network nodes. The exact loss value or the loss percentage can be reported.

Delay statistics: A set of one-way delay statistics attributes that are used to measure end to end latency between VPN sites or between any two network nodes. The peak/min values or percentile values can be reported.

Jitter statistics: A set of one-way IP Packet Delay Variation [RFC3393] statistics attributes that are used to measure end to end jitter between VPN sites or between any two network nodes. The peak/min values or percentile values can be reported.

PM statistics per class ("one-way-pm-statistics-per-class"): Lists p

performance measurement statistics for the topology link or the abstract underlay link between VPN PEs with given "class-id" names. The list is defined separately from "one-way-pm-statistics", which is used to collect generic metrics for unspecified "class-id" names.

VPN PM type ("vpn-pm-type"): Indicates the VPN performance type, which can be inter-vpn-access-interface PM or VPN underlay-tunnel PM. These two methods are common VPN measurement methods. The inter-VPN-access-interface PM is to monitor the performance of logical point-to-point VPN connections between a source and a destination VPN access interfaces. And the underlay-tunnel PM is to monitor the performance of underlay tunnels of VPNs. The inter-VPN-access-interface PM includes PE-PE monitoring. Therefore, usually only one of the two methods is used. The inter-VPN-access-interface PM is defined as an empty leaf, which is not bound to a specific VPN access interface. The source or destination VPN access interface of the measurement can be augmented as needed.

VPN underlay transport type ("vpn-underlay-transport-type"): Indicates the abstract link protocol-type of a VPN, such as GRE or IP-in-IP. The leaf refers to an identifier of the "underlay-transport" defined in [RFC9181], which describes the transport technology to carry the traffic of the VPN service.

For the data nodes of 'termination-point' depicted in Figure 7, the module defines the following minimal set of statistics:

Inbound statistics: A set of inbound statistics attributes that are used to measure the inbound statistics of the termination point, such as received packets, received packets with errors, etc.

Outbound statistics: A set of outbound statistics attributes that are used to measure the outbound statistics of the termination point, such as sent packets, packets that could not be sent due to errors, etc.

VPN network access ("vpn-network-access"): Lists counters of the VPN network access defined in [RFC9182] or [I-D.ietf-opsawg-l2nm]. When multiple VPN network accesses are created using the same physical port, finer-grained metrics can be monitored. If a TP is associated with only a single VPN, this list is not required.

5. Network and VPN Service Performance Monitoring YANG Module

The "ietf-network-vpn-pm" module uses types defined in [RFC8345], [RFC6991], [RFC8532], and [RFC9181].

```
<CODE BEGINS> file "ietf-network-vpn-pm@2022-05-05.yang"
module ietf-network-vpn-pm {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-vpn-pm";
  prefix nvp;

  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Types";
  }
  import ietf-vpn-common {
    prefix vpn-common;
    reference
      "RFC 9181: A Common YANG Data Model for Layer 2 and
      Layer 3 VPNs.";
  }
  import ietf-network {
    prefix nw;
    reference
      "RFC 8345: A YANG Data Model for Network
      Topologies, Section 6.1";
  }
  import ietf-network-topology {
    prefix nt;
    reference
      "RFC 8345: A YANG Data Model for Network
      Topologies, Section 6.2";
  }
  import ietf-lime-time-types {
    prefix lime;
    reference
      "RFC 8532: Generic YANG Data Model for the Management of
      Operations, Administration, and Maintenance (OAM) Protocols
      That Use Connectionless Communications";
  }

  organization
    "IETF OPSAWG (Operations and Management Area Working Group)";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/opsawg/>
    WG List:  <mailto:opsawg@ietf.org>

    Editor: Bo Wu
           <lane.wubo@huawei.com>
    Editor: Mohamed Boucadair
           <mohamed.boucadair@orange.com>
    Editor: Qin Wu
```

```
<bill.wu@huawei.com>
Author: Oscar Gonzalez de Dios
       <oscar.gonzalezdedios@telefonica.com>
Author: Bin Wen
       <bin_wen@comcast.com>;
description
  "This module defines a model for Network and VPN Service
  Performance monitoring.

  Copyright (c) 2022 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject
  to the license terms contained in, the Revised BSD License
  set forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (https://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX
  (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
  for full legal notices.";

// RFC Ed.: update the date below with the date of RFC
// publication and remove this note.
// RFC Ed.: replace XXXX with actual RFC number and remove
// this note.

revision 2022-05-05 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Model for Network and VPN Service
    Performance Monitoring";
}

identity node-type {
  description
    "Base identity for node type";
}

identity pe {
  base node-type;
  description
    "Provider Edge (PE) node type. A PE is the name of the device
    or set of devices at the edge of the provider network with the
    functionality that is needed to interface with the customer.";
}
```

```
identity p {
  base node-type;
  description
    "Provider router node type. That is, a router
    in the core network that does not have interfaces
    directly toward a customer.";
}

identity asbr {
  base node-type;
  description
    "Autonomous System Border Router (ASBR) node type.";
  reference
    "RFC 4364: BGP/MPLS IP Virtual Private Networks (VPNs)";
}

identity pm-source-type {
  description
    "Base identity from which specific performance monitoring
    mechanism types are derived.";
}

identity pm-source-bgppls {
  base pm-source-type;
  description
    "Indicates BGP-LS as the performance monitoring metric source";
  reference
    "RFC 8571: BGP - Link State (BGP-LS) Advertisement of
    IGP Traffic Engineering Performance Metric Extensions";
}

identity pm-source-owamp {
  base pm-source-type;
  description
    "Indicates One-Way Active Measurement Protocol (OWAMP)
    as the performance monitoring metric source.";
  reference
    "RFC 4656: A One-Way Active Measurement Protocol (OWAMP)";
}

identity pm-source-twamp {
  base pm-source-type;
  description
    "Indicates Two-Way Active Measurement Protocol (TWAMP)
    as the performance monitoring metric source.";
  reference
    "RFC 5357: A Two-Way Active Measurement Protocol (TWAMP)";
}
```

```
identity pm-source-stamp {
  base pm-source-type;
  description
    "Indicates Simple Two-way Active Measurement Protocol (STAMP)
    as the performance monitoring metric source.";
  reference
    "RFC 8762: Simple Two-Way Active Measurement Protocol";
}

identity pm-source-y-1731 {
  base pm-source-type;
  description
    "Indicates Ethernet OAM Y.1731 as the performance monitoring
    metric source.";
  reference
    "ITU-T Y.1731: Operations, administration and
    maintenance (OAM) functions and mechanisms
    for Ethernet-based networks";
}

typedef percentage {
  type decimal64 {
    fraction-digits 5;
    range "0..100";
  }
  description
    "Percentage.";
}

typedef percentile {
  type decimal64 {
    fraction-digits 2;
    range "0..100";
  }
  description
    "The percentile is a value between 0 and 100,
    e.g. 10.00, 99.90 ,99.99 etc..
    For example, for a given one-way delay measurement,
    if the percentile is set to 95.00 and
    the 95th percentile one-way delay is 2 milliseconds,
    then the 95 percent of the sample value
    is less than or equal to 2 milliseconds.";
}

grouping entry-summary {
  description
    "Entry summary grouping used for network topology
    augmentation.";
```

```
container entry-summary {
  config false;
  description
    "Container for VPN or network entry summary.";
  container ipv4-num {
    leaf maximum-routes {
      type uint32;
      description
        "Indicates the maximum number of IPv4 routes
        for the VPN.";
    }
    leaf total-active-routes {
      type uint32;
      description
        "Indicates total active IPv4 routes for the VPN.";
    }
    description
      "IPv4-specific parameters.";
  }
  container ipv6-num {
    leaf maximum-routes {
      type uint32;
      description
        "Indicates the maximum number of IPv6 routes
        for the VPN.";
    }
    leaf total-active-routes {
      type uint32;
      description
        "Indicates total active IPv6 routes for the VPN.";
    }
    description
      "IPv6-specific parameters.";
  }
  container mac-num {
    leaf mac-num-limit {
      type uint32;
      description
        "Maximum number of MAC addresses.";
    }
    leaf total-active-mac-num {
      type uint32;
      description
        "Total active MAC entries for the VPN.";
    }
    description
      "MAC statistics.";
  }
}
```

```
    }  
  }  
  
  grouping link-loss-statistics {  
    description  
      "Grouping for per link error statistics.";  
    container loss-statistics {  
      description  
        "One-way link loss summarized information.";  
      reference  
        "RFC 4656: A One-way Active Measurement Protocol (OWAMP)  
        ITU-T Y.1731: Operations, administration and  
        maintenance (OAM) functions and mechanisms  
        for Ethernet-based networks";  
      leaf packet-loss-count {  
        type yang:counter64;  
        description  
          "Total received packet drops count.";  
      }  
      leaf loss-ratio {  
        type percentage;  
        description  
          "Loss ratio of the packets. Express as percentage  
          of packets lost with respect to packets sent.";  
      }  
    }  
  }  
}  
  
grouping link-delay-statistics {  
  description  
    "Grouping for per link delay statistics.";  
  container delay-statistics {  
    description  
      "One-way link delay summarized information.";  
    reference  
      "RFC 4656: A One-way Active Measurement Protocol (OWAMP)  
      ITU-T Y.1731: Operations, administration and  
      maintenance (OAM) functions and mechanisms  
      for Ethernet-based networks";  
    leaf unit-value {  
      type identityref {  
        base lime:time-unit-type;  
      }  
      default "lime:milliseconds";  
      description  
        "Time units, where the options are s, ms, ns, etc.";  
    }  
    leaf min-delay-value {
```

```
        type yang:gauge64;
        description
            "Minimum observed one-way delay.";
    }
    leaf max-delay-value {
        type yang:gauge64;
        description
            "Maximum observed one-way delay.";
    }
    leaf low-delay-percentile {
        type yang:gauge64;
        description
            "Low percentile of observed one-way delay with
             specific measurement method.";
    }
    leaf intermediate-delay-percentile {
        type yang:gauge64;
        description
            "Intermediate percentile of observed one-way delay with
             specific measurement method.";
    }
    leaf high-delay-percentile {
        type yang:gauge64;
        description
            "High percentile of observed one-way delay with
             specific measurement method.";
    }
}

}

grouping link-jitter-statistics {
    description
        "Grouping for per link jitter statistics.";
    container jitter-statistics {
        description
            "One-way link jitter summarized information.";
        reference
            "RFC 3393: IP Packet Delay Variation Metric
             for IP Performance Metrics (IPPM)
             RFC 4656: A One-way Active Measurement Protocol (OWAMP)
             ITU-T Y.1731: Operations, administration and
             maintenance (OAM) functions and mechanisms
             for Ethernet-based networks";
        leaf unit-value {
            type identityref {
                base lime:time-unit-type;
            }
            default "lime:milliseconds";
        }
    }
}
```

```
        description
            "Time units, where the options are s, ms, ns, etc.";
    }
    leaf min-jitter-value {
        type yang:gauge64;
        description
            "Minimum observed one-way jitter.";
    }
    leaf max-jitter-value {
        type yang:gauge64;
        description
            "Maximum observed one-way jitter.";
    }
    leaf low-jitter-percentile {
        type yang:gauge64;
        description
            "Low percentile of observed one-way jitter.";
    }
    leaf intermediate-jitter-percentile {
        type yang:gauge64;
        description
            "Intermediate percentile of observed one-way jitter.";
    }
    leaf high-jitter-percentile {
        type yang:gauge64;
        description
            "High percentile of observed one-way jitter.";
    }
}

grouping tp-svc-telemetry {
    leaf reference-time {
        type yang:date-and-time;
        config false;
        description
            "Indicates the time when the statistics are collected.";
    }
    leaf inbound-octets {
        type yang:counter64;
        description
            "The total number of octets received on the
            interface, including framing characters.";
    }
    leaf inbound-unicast {
        type yang:counter64;
        description
            "The total number of inbound unicast packets.";
```

```
}
leaf inbound-nunicast {
  type yang:counter64;
  description
    "The total number of inbound non-unicast
    (i.e., broadcast or multicast) packets.";
}
leaf inbound-discards {
  type yang:counter64;
  description
    "The number of inbound packets that were chosen to be
    discarded even though no errors had been detected.
    Possible reasons for discarding such a packet could
    be to free up buffer space, not enough buffer for
    too much data, etc.";
}
leaf inbound-errors {
  type yang:counter64;
  description
    "The number of inbound packets that contained errors.";
}
leaf inbound-unknown-protocol {
  type yang:counter64;
  description
    "The number of packets received via the interface
    which were discarded because of an unknown or
    unsupported protocol.";
}
leaf outbound-octets {
  type yang:counter64;
  description
    "The total number of octets transmitted out of the
    interface, including framing characters.";
}
leaf outbound-unicast {
  type yang:counter64;
  description
    "The total number of outbound unicast packets.";
}
leaf outbound-nunicast {
  type yang:counter64;
  description
    "The total number of outbound non unicast
    (i.e., broadcast or multicast) packets.";
}
leaf outbound-discards {
  type yang:counter64;
  description
```

```
        "The number of outbound packets which were chosen
        to be discarded even though no errors had been
        detected to prevent their being transmitted.
        Possible reasons for discarding such a packet could
        be to free up buffer space, not enough buffer for
        too much data, etc.";
    }
    leaf outbound-errors {
        type yang:counter64;
        description
            "The number of outbound packets that contained
            errors.";
    }
    description
        "Grouping for interface service telemetry.";
}

augment "/nw:networks/nw:network/nw:network-types" {
    description
        "Defines the service topologies types.";
    container service-type {
        presence "Indicates network service topology.";
        leaf service-type {
            type identityref {
                base vpn-common:service-type;
            }
            description
                "The presence identifies the network service type,
                e.g., L3VPN, VPLS, etc.";
        }
        description
            "Container for VPN service type.";
    }
}

augment "/nw:networks/nw:network" {
    when 'nw:network-types/nvp:service-type' {
        description
            "Augments only for VPN Network topology.";
    }
    description
        "Augments the network with service topology attributes";
    container vpn-pm-attributes {
        leaf vpn-id {
            type vpn-common:vpn-id;
            description
                "VPN identifier.";
        }
    }
}
```

```
    leaf vpn-service-topology {
      type identityref {
        base vpn-common:vpn-topology;
      }
      description
        "VPN service topology, e.g., hub-spoke, any-to-any,
        hub-spoke-disjoint.";
    }
    description
      "Container for VPN topology attributes.";
  }
}

augment "/nw:networks/nw:network/nw:node" {
  description
    "Augments the network node with other general attributes.";
  container pm-attributes {
    leaf node-type {
      type identityref {
        base node-type;
      }
      description
        "Node type, e.g., PE, P, ASBR.";
    }
    description
      "Container for node attributes.";
    uses entry-summary;
  }
}

augment "/nw:networks/nw:network/nw:node/pm-attributes" {
  when '../nw:network-types/nvp:service-type' {
    description
      "Augments only for VPN node attributes.";
  }
  description
    "Augments the network node with VPN specific attributes.";
  leaf role {
    type identityref {
      base vpn-common:role;
    }
    default "vpn-common:any-to-any-role";
    description
      "Role of the node in the VPN.";
  }
}

augment "/nw:networks/nw:network/nt:link" {
```

```
description
  "Augments the network topology link with performance
   monitoring attributes.";
container pm-attributes {
  description
    "Container for PM attributes.";
  leaf low-percentile {
    type percentile;
    default "10.00";
    description
      "Low percentile to report. Setting low-percentile
       into 0.00 indicates the client is not interested
       in receiving low percentile.";
  }
  leaf intermediate-percentile {
    type percentile;
    default "50.00";
    description
      "Intermediate percentile to report. Setting
       intermediate-percentile into 0.00 indicates the client
       is not interested in receiving intermediate percentile.";
  }
  leaf high-percentile {
    type percentile;
    default "95.00";
    description
      "High percentile to report. Setting high-percentile
       into 0.00 indicates the client is not interested in
       receiving high percentile.";
  }
  leaf measurement-interval {
    type uint32 {
      range "1..max";
    }
    units "seconds";
    default "60";
    description
      "Indicates the time interval to perform PM measurement.";
  }
  leaf start-time {
    type yang:date-and-time;
    config false;
    description
      "The time that the current measurement started.";
  }
  leaf end-time {
    type yang:date-and-time;
    config false;
```

```
        description
            "The time that the current measurement ended.";
    }
    leaf pm-source {
        type identityref {
            base pm-source-type;
        }
        config false;
        description
            "The OAM tool used to collect the PM data.";
    }
    container one-way-pm-statistics {
        config false;
        description
            "Container for link telemetry attributes.";
        uses link-loss-statistics;
        uses link-delay-statistics;
        uses link-jitter-statistics;
    }
    list one-way-pm-statistics-per-class {
        key "class-id";
        config false;
        description
            "The list of PM data based on class of service.";
        leaf class-id {
            type string;
            description
                "The class-id is used to identify the class of service.
                This identifier is internal to the administration.";
        }
        uses link-loss-statistics;
        uses link-delay-statistics;
        uses link-jitter-statistics;
    }
}

augment "/nw:networks/nw:network/nt:link/pm-attributes" {
    when '../..//nw:network-types/nvp:service-type' {
        description
            "Augments only for VPN Network topology.";
    }
    description
        "Augments the network topology link with VPN service
        performance monitoring attributes.";
    container vpn-pm-type {
        description
            "The VPN PM type of this logical point-to-point
```

```
        unidirectional VPN link.";
    container inter-vpn-access-interface {
        description
            "Indicates inter-vpn-access-interface PM, which is to
            monitor the performance of logical point-to-point VPN
            connections between a source and a destination
            VPN access interfaces.";
        leaf inter-vpn-access-interface {
            type empty;
            description
                "This is a placeholder for inter-vpn-access-interface PM,
                which is not bound to a specific VPN access interface.
                The source or destination VPN access interface
                of the measurement can be augmented as needed.";
        }
    }
}
container underlay-tunnel {
    presence "Enables VPN underlay tunnel PM";
    description
        "Indicates underlay-tunnel PM, which is to monitor
        the performance of underlay tunnels of VPNs.";
    leaf vpn-underlay-transport-type {
        type identityref {
            base vpn-common:protocol-type;
        }
        config false;
        description
            "The leaf indicates the underlay transport type of
            a VPN service, e.g., GRE, LDP, etc.";
    }
}
}
}

augment "/nw:networks/nw:network/nw:node/nt:termination-point" {
    description
        "Augments the network topology termination point with
        performance monitoring attributes.";
    container pm-statistics {
        config false;
        description
            "Container for termination point PM attributes.";
        uses tp-svc-telemetry;
    }
}

augment "/nw:networks/nw:network/nw:node"
+ "/nt:termination-point/pm-statistics" {
```

```
when '../.../nw:network-types/nvp:service-type' {
  description
    "Augments only for VPN Network topology.";
}
description
  "Augments the network topology termination-point with
  VPN service performance monitoring attributes";
list vpn-network-access {
  key "network-access-id";
  description
    "The list of PM based on VPN network accesses.";
  leaf network-access-id {
    type vpn-common:vpn-id;
    description
      "References to an identifier for the VPN network
      access, e.g. L3VPN or VPLS.";
  }
  uses tp-svc-telemetry;
}
}
}
}
<CODE ENDS>
```

6. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The 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.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

- * `"/nw:networks/nw:network/nw:network-types"`: This subtree specifies the VPN service type. Unauthorized access to this subtree may render the VPN service type invalid.
- * `"/nw:networks/nw:network/nvp:vpn-pm-attributes"`: This subtree specifies the VPN service identifier and VPN service topology. Unauthorized access to this subtree may disable the the VPN PM or render the VPN service topology invalid.
- * `"/nw:networks/nw:network/nw:node/nvp:pm-attributes"`: This subtree specifies the network node type and VPN service topology role. Unauthorized access to this subtree may render the node type or VPN service topology invalid.
- * `"/nw:networks/nw:network/nt:link/nvp:pm-attributes"`: This subtree specifies the PM of the network link and VPN link. Unauthorized access to this subtree can impact the network and VPN monitoring.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via `get`, `get-config`, or `notification`) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

- * `"/nw:networks/nw:network/nw:node/nvp:pm-attributes/nvp:vpn-summary-statistics"`: Unauthorized access to this subtree can disclose the operational state information of VPN instances.
- * `"/nw:networks/nw:network/nt:link/nvp:pm-attributes/nvp:one-way-pm-statistics"`: Unauthorized access to this subtree can disclose the operational state information of network links or VPN abstract links.
- * `"/nw:networks/nw:network/nw:node/nt:termination-point/nvp:pm-statistics"`: Unauthorized access to this subtree can disclose the operational state information of network termination points or VPN network accesses.

7. 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-network-vpn-pm`
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-network-vpn-pm
Namespace: urn:ietf:params:xml:ns:yang:ietf-network-vpn-pm
Maintained by IANA: N
Prefix: nvp
Reference: RFC XXXX (RFC Ed.: replace XXXX with actual RFC number and remove this note.)

8. Acknowledgements

Thanks to Joe Clarke, Adrian Farrel, Tom Petch, Greg Mirsky, Roque Gagliano, Erez Segev, and Dhruv Dhody for reviewing and providing important input to this document.

9. Contributors

The following authors contributed significantly to this document:

Michale Wang
Huawei
Email: wangzitao@huawei.com

Roni Even
Huawei
Email: ron.even.tlv@gmail.com

Change Liu
China Unicom
Email: liuc131@chinaunicom.cn

Honglei Xu
China Telecom
Email: xuhl6@chinatelecom.cn

10. References

10.1. Normative References

[ITU-T-Y-1731]
ITU-T, "Operator Ethernet Service Definition", August 2015, <<https://www.itu.int/rec/T-REC-Y.1731/en>>.

- [RFC3393] Demichelis, C. and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)", RFC 3393, DOI 10.17487/RFC3393, November 2002, <<https://www.rfc-editor.org/info/rfc3393>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [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>>.
- [RFC4656] Shalunov, S., Teitelbaum, B., Karp, A., Boote, J., and M. Zekauskas, "A One-way Active Measurement Protocol (OWAMP)", RFC 4656, DOI 10.17487/RFC4656, September 2006, <<https://www.rfc-editor.org/info/rfc4656>>.
- [RFC5357] Hedayat, K., Krzanowski, R., Morton, A., Yum, K., and J. Babiarz, "A Two-Way Active Measurement Protocol (TWAMP)", RFC 5357, DOI 10.17487/RFC5357, October 2008, <<https://www.rfc-editor.org/info/rfc5357>>.
- [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>>.
- [RFC6374] Frost, D. and S. Bryant, "Packet Loss and Delay Measurement for MPLS Networks", RFC 6374, DOI 10.17487/RFC6374, September 2011, <<https://www.rfc-editor.org/info/rfc6374>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8532] Kumar, D., Wang, Z., Wu, Q., Ed., Rahman, R., and S. Raghavan, "Generic YANG Data Model for the Management of Operations, Administration, and Maintenance (OAM) Protocols That Use Connectionless Communications", RFC 8532, DOI 10.17487/RFC8532, April 2019, <<https://www.rfc-editor.org/info/rfc8532>>.
- [RFC8571] Ginsberg, L., Ed., Previdi, S., Wu, Q., Tantsura, J., and C. Filsfils, "BGP - Link State (BGP-LS) Advertisement of IGP Traffic Engineering Performance Metric Extensions", RFC 8571, DOI 10.17487/RFC8571, March 2019, <<https://www.rfc-editor.org/info/rfc8571>>.
- [RFC8641] Clemm, A. and E. Voit, "Subscription to YANG Notifications for Datastore Updates", RFC 8641, DOI 10.17487/RFC8641, September 2019, <<https://www.rfc-editor.org/info/rfc8641>>.
- [RFC8762] Mirsky, G., Jun, G., Nydell, H., and R. Foote, "Simple Two-Way Active Measurement Protocol", RFC 8762, DOI 10.17487/RFC8762, March 2020, <<https://www.rfc-editor.org/info/rfc8762>>.

- [RFC9181] Barguil, S., Gonzalez de Dios, O., Ed., Boucadair, M., Ed., and Q. Wu, "A Common YANG Data Model for Layer 2 and Layer 3 VPNs", RFC 9181, DOI 10.17487/RFC9181, February 2022, <<https://www.rfc-editor.org/info/rfc9181>>.

10.2. Informative References

- [I-D.ietf-netmod-node-tags]
Wu, Q., Claise, B., Liu, P., Du, Z., and M. Boucadair, "Data Node Tags in YANG Modules", Work in Progress, Internet-Draft, draft-ietf-netmod-node-tags-07, 29 April 2022, <<https://www.ietf.org/archive/id/draft-ietf-netmod-node-tags-07.txt>>.
- [I-D.ietf-opsawg-l2nm]
Boucadair, M., Dios, O. G. D., Barguil, S., and L. A. Munoz, "A YANG Network Data Model for Layer 2 VPNs", Work in Progress, Internet-Draft, draft-ietf-opsawg-l2nm-15, 29 April 2022, <<https://www.ietf.org/archive/id/draft-ietf-opsawg-l2nm-15.txt>>.
- [I-D.ietf-opsawg-sap]
Boucadair, M., Dios, O. G. D., Barguil, S., Wu, Q., and V. Lopez, "A Network YANG Model for Service Attachment Points (SAPs)", Work in Progress, Internet-Draft, draft-ietf-opsawg-sap-04, 11 April 2022, <<https://www.ietf.org/archive/id/draft-ietf-opsawg-sap-04.txt>>.
- [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>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<https://www.rfc-editor.org/info/rfc5277>>.
- [RFC7471] Giacalone, S., Ward, D., Drake, J., Atlas, A., and S. Previdi, "OSPF Traffic Engineering (TE) Metric Extensions", RFC 7471, DOI 10.17487/RFC7471, March 2015, <<https://www.rfc-editor.org/info/rfc7471>>.
- [RFC8194] Schoenwaelder, J. and V. Bajpai, "A YANG Data Model for LMAP Measurement Agents", RFC 8194, DOI 10.17487/RFC8194, August 2017, <<https://www.rfc-editor.org/info/rfc8194>>.

- [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>>.
- [RFC8570] Ginsberg, L., Ed., Previdi, S., Ed., Giacalone, S., Ward, D., Drake, J., and Q. Wu, "IS-IS Traffic Engineering (TE) Metric Extensions", RFC 8570, DOI 10.17487/RFC8570, March 2019, <<https://www.rfc-editor.org/info/rfc8570>>.
- [RFC8632] Vallin, S. and M. Bjorklund, "A YANG Data Model for Alarm Management", RFC 8632, DOI 10.17487/RFC8632, September 2019, <<https://www.rfc-editor.org/info/rfc8632>>.
- [RFC8639] Voit, E., Clemm, A., Gonzalez Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Subscription to YANG Notifications", RFC 8639, DOI 10.17487/RFC8639, September 2019, <<https://www.rfc-editor.org/info/rfc8639>>.
- [RFC8969] Wu, Q., Ed., Boucadair, M., Ed., Lopez, D., Xie, C., and L. Geng, "A Framework for Automating Service and Network Management with YANG", RFC 8969, DOI 10.17487/RFC8969, January 2021, <<https://www.rfc-editor.org/info/rfc8969>>.
- [RFC9182] Barguil, S., Gonzalez de Dios, O., Ed., Boucadair, M., Ed., Munoz, L., and A. Aguado, "A YANG Network Data Model for Layer 3 VPNs", RFC 9182, DOI 10.17487/RFC9182, February 2022, <<https://www.rfc-editor.org/info/rfc9182>>.

Appendix A. Illustrative Examples

A.1. VPN Performance Subscription Example

The example shown in Figure 8 illustrates how a client subscribes to the performance monitoring information between nodes ('node-id') A and B in the L3 network topology. The performance monitoring parameter that the client is interested in is end-to-end loss.

```
POST /restconf/operations
    /ietf-subscribed-notifications:establish-subscription
{
  "ietf-subscribed-notifications:input":{
    "stream-subtree-filter":{
      "ietf-network:networks":{
        "network":{
          "network-id":"foo:l3-network",
          "ietf-network-vpn-pm:service-type":{
            "ietf-vpn-common:l3vpn":{}
          }
        }
      }
    }
  }
```

```

    },
    "node":[
      {
        "node-id":"A",
        "ietf-network-vpn-pm:pm-attributes":{
          "node-type":"PE"
        },
        "termination-point":{
          "tp-id":"1-0-1"
        }
      },
      {
        "node-id":"B",
        "ietf-network-vpn-pm:pm-attributes":{
          "node-type":"PE"
        },
        "termination-point":{
          "tp-id":"2-0-1"
        }
      }
    ],
    "ietf-network-topology:link":{
      "link-id":"A-B",
      "source":{
        "source-node":"A"
      },
      "destination":{
        "dest-node":"B"
      },
      "ietf-network-vpn-pm:pm-attributes":{
        "one-way-pm-statistics":{
          "loss-statistics":{
            "packet-loss-count":{}
          }
        },
        "vpn-underlay-transport-type":"ietf-vpn-common:gre"
      }
    }
  }
},
"ietf-yang-push:periodic":{
  "ietf-yang-push:period":"500"
}
}

```

Figure 8: Pub/Sub Retrieval

A.2. Example of VPN Performance Snapshot

This example, depicted in Figure 9, illustrates an VPN PM instance example in which a client uses RESTCONF [RFC8040] to fetch the performance data of the link and TP belonged to "VPN1".

```

{
  "ietf-network:networks": {
    "network": {
      "network-id": "foo:vpn1",
      "node": [
        {
          "node-id": "A",
          "ietf-network-vpn-pm:pm-attributes": {
            "node-type": "PE"
          },
          "termination-point": {
            "tp-id": "1-0-1",
            "ietf-network-vpn-pm:pm-statistics": {
              "inbound-octets": "100",
              "outbound-octets": "150"
            }
          }
        },
        {
          "node-id": "B",
          "ietf-network-vpn-pm:pm-attributes": {
            "node-type": "PE"
          },
          "termination-point": {
            "tp-id": "2-0-1",
            "ietf-network-vpn-pm:pm-statistics": {
              "inbound-octets": "150",
              "outbound-octets": "100"
            }
          }
        }
      ],
      "ietf-network-topology:link": {
        "link-id": "A-B",
        "source": { "source-node": "A" },
        "destination": { "dest-node": "B" },
        "ietf-network-pm:pm-attributes": {
          "one-way-pm-statistics": {
            "loss-statistics": { "packet-loss-count": "120" }
          },
          "vpn-underlay-transport-type": "ietf-vpn-common:gre"
        }
      }
    }
  }
}

```

Figure 9

A.3. Example of Percentile Monitoring

The following shows an example of a percentile measurement for a VPN link.

```
{
  "ietf-network-topology:link": [
    {
      "link-id": "foo:vpn1-link1",
      "source": {
        "source-node": "vpn-node1"
      },
      "destination": {
        "dest-node": "vpn-node3"
      },
      "ietf-network-vpn-pm:pm-attributes": {
        "low-percentile": "20.00",
        "intermediate-percentile": "50.00",
        "high-percentile": "90.00",
        "one-way-pm-statistics": {
          "delay-statistics": {
            "unit-value": "lime:milliseconds",
            "min-delay-value": "43",
            "max-delay-value": "99",
            "low-delay-percentile": "64",
            "intermediate-delay-percentile": "77",
            "high-delay-percentile": "98"
          }
        },
        "vpn-pm-type": {
          "inter-vpn-access-interface": [null]
        }
      }
    }
  ]
}
```

Authors' Addresses

Bo Wu (editor)
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: lana.wubo@huawei.com

Qin Wu (editor)
Huawei
101 Software Avenue, Yuhua District
Nanjing
Jiangsu, 210012
China
Email: bill.wu@huawei.com

Mohamed Boucadair (editor)
Orange
Rennes 35000
France
Email: mohamed.boucadair@orange.com

Oscar Gonzalez de Dios
Telefonica
Madrid
Spain
Email: oscar.gonzalezdedios@telefonica.com

Bin Wen
Comcast
Email: bin_wen@comcast.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 19, 2020

E. Lear
Cisco Systems
S. Rose
NIST
May 18, 2020

SBOM Extension for MUD
draft-lear-opsawg-mud-sbom-00

Abstract

Software bills of materials (SBOMs) are formal descriptions of what pieces of software are included in a product. This memo specifies a means for manufacturers to state how SBOMs may be retrieved through an extension to manufacturer usage descriptions (MUD).

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 November 19, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. How This Information Is Used	3
1.2. SBOM formats	3
1.3. Discussion points	3
2. The mud-sbom extension model extension	4
3. The mud-sbom augmentation to the MUD YANG model	4
4. Examples	7
4.1. Without ACLS	7
4.2. Located on the Device	8
4.3. SBOM Obtained from Contact Information	9
4.4. With ACLS	9
5. Security Considerations	12
6. IANA Considerations	12
6.1. MUD Extension	12
6.2. Well-Known Prefix	12
7. References	13
7.1. Normative References	13
7.2. Informative References	13
Appendix A. Changes from Earlier Versions	13
Authors' Addresses	14

1. Introduction

Manufacturer Usage Descriptions (MUD) [RFC8520] provides a means for devices to identify what they are and what sort of network access they need. This memo specifies a YANG model [RFC6991] for reporting and a means for transmitting the report, and appropriate extensions to the MUD file to indicate how to report and how often.

Software bills of material (SBOMs) are descriptions of what software, including versioning and dependencies, a device contains. There are different SBOM formats such as Software Package Data Exchange [SPDX] and Software Identity Tags [SWID].

This memo extends the MUD YANG schema to provide location information of an SBOM.

These SBOMs are typically found in one of three ways:

- o on devices themselves
- o on a web site (e.g., via URI)
- o through direct contact with the manufacturer.

Some devices will have interfaces that permit direct SBOM retrieval. Examples of these interfaces might be 'ssh' or an HTTP endpoint for retrieval. There may also be private interfaces as well.

When a web site is used, versioning information about the SBOM is implicit based on the MUD file.

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.

1.1. How This Information Is Used

SBOMs are used for numerous purposes, including vulnerability assessment, license management, and inventory management. This memo provides means for either automated or semi-automated collection of that information. For devices that can output a MUD URL, the mechanism may be highly automated. For devices that have a MUD URL in either their documentation or within a QR code on a box, the mechanism is semi-automated (someone has to scan the QR code or enter the URL).

Note that SBOMs may change more frequently than access control requirements. A change to software does not necessarily mean a change to control channels that are used. Therefore, it is important to retrieve the MUD file as suggested by the manufacturer in the cache-validity period. In many cases, only the SBOM list will have been updated.

1.2. SBOM formats

There are multiple ways to express an SBOM. When these are retrieved either directly from the device or directly from a web server, tools will need to observe the content-type header to determine precisely which format is being transmitted. Because IoT devices in particular have limited capabilities, use of a specific Accept: header in HTTP or the Accept Option in CoAP is NOT RECOMMENDED. Instead, backend tooling MUST silently discard SBOM information sent with a media type that is not understood.

1.3. Discussion points

The following is discussion to be removed at time of RFC publication.

- o Is the model structured correctly?

- o Are there other retrieval mechanisms that need to be specified?
- o Do we need to be more specific in how to authenticate and retrieve SBOMs?
- o What are the implications if the MUD URL is an extension in a certificate (e.g. an IDevID cert)?

2. The mud-sbom extension model extension

We now formally define this extension. This is done in two parts. First, the extension name "sbom" is listed in the "extensions" array of the MUD file.

Second, the "mud" container is augmented with a list of SBOM sources.

This is done as follows:

```
module: ietf-mud-sbom
  augment /mud:mud:
    +--rw sboms* [version-info]
      +--rw version-info          string
      +--rw (sbom-type)?
        +--:(url)
          | +--rw sbom-url?       inet:uri
        +--:(local-uri)
          | +--rw sbom-local*     enumeration
        +--:(contact-info)
          +--rw contact-uri?     inet:uri
```

3. The mud-sbom augmentation to the MUD YANG model

```
<CODE BEGINS>file "ietf-mud-sbom@2020-03-06.yang"
module ietf-mud-sbom {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-mud-sbom";
  prefix mud-sbom;

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

  organization
    "IETF OPSAWG (Ops Area) Working Group";
  contact
```

```
"WG
  Web: http://tools.ietf.org/wg/opsawg/
  WG List: opsawg@ietf.org
  Author: Eliot Lear lear@cisco.com ";
description
  "This YANG module augments the ietf-mud model to provide for
  reporting of SBOMs.

  Copyright (c) 2019 IETF Trust and the persons identified as
  authors of the code. All rights reserved.

  Redistribution and use in source and binary forms, with or
  without modification, is permitted pursuant to, and subject to
  the license terms contained in, the Simplified BSD License set
  forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (https://trustee.ietf.org/license-info).

  This version of this YANG module is part of RFC XXXX
  (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself for
  full legal notices.

  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 (RFC 2119) (RFC 8174) when, and only when,
  they appear in all capitals, as shown here. ";

revision 2020-03-06 {
  description
    "Initial proposed standard.";
  reference
    "RFC XXXX: Extension for MUD Reporting";
}

grouping mud-sbom-extension {
  description
    "SBOM extension grouping";
  list sboms {
    key "version-info";
    leaf version-info {
      type string;
      description
        "A version string that is applicable for this SBOM list entry.
        The format of this string is left to the device manufacturer.
        How the network administrator determines the version of
        software running on the device is beyond the scope of this
        memo.";
```

```
}
choice sbom-type {
  case url {
    leaf sbom-url {
      type inet:uri;
      description
        "A statically located URI.";
    }
  }
  case local-uri {
    leaf-list sbom-local {
      type enumeration {
        enum coap {
          description
            "Use COAP schema to retrieve SBOM";
        }
        enum coaps {
          description
            "Use COAPS schema to retrieve SBOM";
        }
        enum http {
          description
            "Use HTTP schema to retrieve SBOM";
        }
        enum https {
          description
            "Use HTTPS schema to retrieve SBOM";
        }
      }
    }
    description
      "The choice of sbom-local means that the SBOM resides at
      a location indicated by an indicted scheme for the
      device in question, at well known location
      '/.well-known/sbom'. For example, if the MUD file
      indicates that coaps is to be used and the host is
      located at address 10.1.2.3, the SBOM could be retrieved
      at 'coaps://10.1.2.3/.well-known/sbom'. N.B., coap and
      http schemes are NOT RECOMMENDED.";
  }
}
case contact-info {
  leaf contact-uri {
    type inet:uri;
    description
      "This MUST be either a tel, http, https, or
      mailto uri schema that customers can use to
      contact someone for SBOM information.";
  }
}
```

```
    }
    description
      "choices for SBOM retrieval.";
  }
  description
    "list of methods to get an SBOM.";
}

augment "/mud:mud" {
  description
    "Add extension for SBOMs.";
  uses mud-sbom-extension;
}
```

<CODE ENDS>

4. Examples

In this example MUD file that uses a cloud service, the Frobinator presents a location of the SBOM in a URL. Note, the ACLs in a MUD file are NOT required, although they are a very good idea for IP-based devices. The first MUD file demonstrates how to get the SBOM without ACLs, and the second has ACLs.

4.1. Without ACLS

```
{
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://iot-device.example.com/dnsname",
    "last-update": "2019-01-15T10:22:47+00:00",
    "cache-validity": 48,
    "is-supported": true,
    "systeminfo": "device that wants to talk to a cloud service",
    "mfg-name": "Example, Inc.",
    "documentation": "https://frobinator.example.com/doc/frob2000",
    "model-name": "Frobinator 2000",
    "extensions" : [
      "sbom"
    ],
    "sboms" : [
      {
        "version-info" : "FrobOS Release 1.1",
        "sbom-url" : "https://frobinator.example.com/sboms/f20001.1",
      }
    ]
  }
}
```

4.2. Located on the Device

```
{
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://iot-device.example.com/dnsname",
    "last-update": "2019-01-15T10:22:47+00:00",
    "cache-validity": 48,
    "is-supported": true,
    "systeminfo": "device that wants to talk to a cloud service",
    "mfg-name": "Example, Inc.",
    "documentation": "https://frobinator.example.com/doc/frob2000",
    "model-name": "Frobinator 2000",
    "extensions" : [
      "sbom"
    ],
    "sboms" : [
      {
        "version-info" : "FrobOS Release 1.1",
        "sbom-local" : "coaps:///well-known/sbom",
      }
    ]
  }
}
```

4.3. SBOM Obtained from Contact Information

```
{
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://iot-device.example.com/dnsname",
    "last-update": "2019-01-15T10:22:47+00:00",
    "cache-validity": 48,
    "is-supported": true,
    "systeminfo": "device that wants to talk to a cloud service",
    "mfg-name": "Example, Inc.",
    "documentation": "https://frobinator.example.com/doc/frob2000",
    "model-name": "Frobinator 2000",
    "extensions" : [
      "sbom"
    ],
    "sboms" : [
      {
        "version-info" : "FrobOS Release 1.1",
        "contact-uri" : "mailto:sbom-request@example.com",
      }
    ]
  }
}
```

4.4. With ACLS

```
{
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://iot-device.example.com/dnsname",
    "last-update": "2019-01-15T10:22:47+00:00",
    "cache-validity": 48,
    "is-supported": true,
    "systeminfo": "device that wants to talk to a cloud service",
    "mfg-name": "Example, Inc.",
    "documentation": "https://frobinator.example.com/doc/frob2000",
    "model-name": "Frobinator 2000",
    "extensions" : [
      "sbom"
    ],
    "sboms" : [
      {
        "version-info" : "FrobOS Release 1.1",
        "sbom-url" : "https://frobinator.example.com/sboms/f20001.1",
      }
    ],
    "from-device-policy": {
```

```
"access-lists": {
  "access-list": [
    {
      "name": "mud-96898-v4fr"
    },
    {
      "name": "mud-96898-v6fr"
    }
  ]
},
"to-device-policy": {
  "access-lists": {
    "access-list": [
      {
        "name": "mud-96898-v4to"
      },
      {
        "name": "mud-96898-v6to"
      }
    ]
  }
},
"ietf-access-control-list:acls": {
  "acl": [
    {
      "name": "mud-96898-v4to",
      "type": "ipv4-acl-type",
      "aces": {
        "ace": [
          {
            "name": "cl0-todev",
            "matches": {
              "ipv4": {
                "ietf-acldns:src-dnsname": "cloud-service.example.com"
              }
            },
            "actions": {
              "forwarding": "accept"
            }
          }
        ]
      }
    }
  ]
},
{
  "name": "mud-96898-v4fr",
  "type": "ipv4-acl-type",
```

```
"aces": {
  "ace": [
    {
      "name": "cl0-frdev",
      "matches": {
        "ipv4": {
          "ietf-acldns:dst-dnsname": "cloud-service.example.com"
        }
      },
      "actions": {
        "forwarding": "accept"
      }
    }
  ]
},
{
  "name": "mud-96898-v6to",
  "type": "ipv6-acl-type",
  "aces": {
    "ace": [
      {
        "name": "cl0-todev",
        "matches": {
          "ipv6": {
            "ietf-acldns:src-dnsname": "cloud-service.example.com"
          }
        },
        "actions": {
          "forwarding": "accept"
        }
      }
    ]
  }
},
{
  "name": "mud-96898-v6fr",
  "type": "ipv6-acl-type",
  "aces": {
    "ace": [
      {
        "name": "cl0-frdev",
        "matches": {
          "ipv6": {
            "ietf-acldns:dst-dnsname": "cloud-service.example.com"
          }
        },
        "actions": {
```

```
        "forwarding": "accept"
      }
    }
  ]
}
}
```

At this point, the management system can attempt to retrieve the SBOM, and determine which format is in use through the content-type header on the response to a GET request.

5. Security Considerations

SBOMs provide an inventory of software. If firmware is available to an attacker, the attacker may well already be able to derive this very same software inventory. Manufacturers MAY restrict access to SBOM information using appropriate authorization semantics within HTTP. In particular, if a system attempts to retrieve an SBOM via HTTP, if the client is not authorized, the server MUST produce an appropriate error, with instructions on how to register a particular client. One example may be to issue a certificate to the client for this purpose after a registration process has taken place. Another example would involve the use of OAUTH in combination with a federations of SBOM servers.

To further mitigate attacks against a device, manufacturers SHOULD recommend access controls through the normal MUD mechanism.

6. IANA Considerations

6.1. MUD Extension

The IANA is requested to add "controller-candidate" to the MUD extensions registry as follows:

Extension Name: sbom
Standard reference: This document

6.2. Well-Known Prefix

The following well known URI is requested in accordance with [RFC8615]:

URI suffix: "sbom"
Change controller: "IETF"
Specification document: This memo
Related information: See ISO/IEC 19970-2 and SPDX.org

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [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>>.
- [RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <<https://www.rfc-editor.org/info/rfc8520>>.
- [RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <<https://www.rfc-editor.org/info/rfc8615>>.

7.2. Informative References

- [SPDX] The Linux Foundation, "SPDX Specification 2.1", 2016.
- [SWID] ISO/IEC, "Information technology -- IT asset management -- Part 2: Software identification tag", ISO 19770-2:2015, 2015.

Appendix A. Changes from Earlier Versions

Draft -00:

- o Initial revision

Authors' Addresses

Eliot Lear
Cisco Systems
Richtistrasse 7
Wallisellen CH-8304
Switzerland

Phone: +41 44 878 9200
Email: lear@cisco.com

Scott Rose
NIST
100 Bureau Dr
Gaithersburg MD 20899
USA

Phone: +1 301-975-8439
Email: scott.rose@nist.gov

QUIC
Internet-Draft
Intended status: Standards Track
Expires: 6 May 2021

R. Marx
Hasselt University
2 November 2020

QUIC and HTTP/3 event definitions for qlog
draft-marx-qlog-event-definitions-quic-h3-02

Abstract

This document describes concrete qlog event definitions and their metadata for QUIC and HTTP/3-related events. These events can then be embedded in the higher level schema defined in [QLOG-MAIN].

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 6 May 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	5
1.1. Notational Conventions	5
2. Overview	5
2.1. Importance	6
2.2. Custom fields	7
3. Events not belonging to a single connection	7
4. QUIC and HTTP/3 fields	8
4.1. Raw packet and frame information	8
5. QUIC event definitions	10
5.1. connectivity	10
5.1.1. server_listening	10
5.1.2. connection_started	10
5.1.3. connection_closed	11
5.1.4. connection_id_updated	12
5.1.5. spin_bit_updated	12
5.1.6. connection_retried	12
5.1.7. connection_state_updated	13
5.1.8. MIGRATION-related events	15
5.2. security	15
5.2.1. key_updated	15
5.2.2. key_retired	15
5.3. transport	16
5.3.1. version_information	16
5.3.2. alpn_information	17
5.3.3. parameters_set	18
5.3.4. parameters_restored	20
5.3.5. packet_sent	20
5.3.6. packet_received	21
5.3.7. packet_dropped	22
5.3.8. packet_buffered	23
5.3.9. packets_acked	24
5.3.10. datagrams_sent	24
5.3.11. datagrams_received	25
5.3.12. datagram_dropped	25
5.3.13. stream_state_updated	26
5.3.14. frames_processed	27
5.3.15. data_moved	28
5.4. recovery	30
5.4.1. parameters_set	30
5.4.2. metrics_updated	30
5.4.3. congestion_state_updated	31
5.4.4. loss_timer_updated	32
5.4.5. packet_lost	33
5.4.6. marked_for_retransmit	34
6. HTTP/3 event definitions	34
6.1. http	34

6.1.1.	parameters_set	34
6.1.2.	parameters_restored	35
6.1.3.	stream_type_set	36
6.1.4.	frame_created	36
6.1.5.	frame_parsed	37
6.1.6.	push_resolved	37
6.2.	qpack	38
6.2.1.	state_updated	38
6.2.2.	stream_state_updated	39
6.2.3.	dynamic_table_updated	39
6.2.4.	headers_encoded	39
6.2.5.	headers_decoded	40
6.2.6.	instruction_created	40
6.2.7.	instruction_parsed	41
7.	Generic events and Simulation indicators	41
7.1.	generic	41
7.1.1.	error	42
7.1.2.	warning	42
7.1.3.	info	42
7.1.4.	debug	42
7.1.5.	verbose	43
7.2.	simulation	43
7.2.1.	scenario	43
7.2.2.	marker	44
8.	Security Considerations	44
9.	IANA Considerations	44
10.	References	44
10.1.	Normative References	44
10.2.	Informative References	45
Appendix A.	QUIC data field definitions	45
A.1.	IPAddress	45
A.2.	PacketType	45
A.3.	PacketNumberSpace	45
A.4.	PacketHeader	45
A.5.	Token	46
A.6.	KeyType	46
A.7.	QUIC Frames	47
A.7.1.	PaddingFrame	47
A.7.2.	PingFrame	47
A.7.3.	AckFrame	47
A.7.4.	ResetStreamFrame	48
A.7.5.	StopSendingFrame	48
A.7.6.	CryptoFrame	49
A.7.7.	NewTokenFrame	49
A.7.8.	StreamFrame	49
A.7.9.	MaxDataFrame	50
A.7.10.	MaxStreamDataFrame	50
A.7.11.	MaxStreamsFrame	50

A.7.12.	DataBlockedFrame	50
A.7.13.	StreamDataBlockedFrame	50
A.7.14.	StreamsBlockedFrame	50
A.7.15.	NewConnectionIDFrame	51
A.7.16.	RetireConnectionIDFrame	51
A.7.17.	PathChallengeFrame	51
A.7.18.	PathResponseFrame	51
A.7.19.	ConnectionCloseFrame	52
A.7.20.	HandshakeDoneFrame	52
A.7.21.	UnknownFrame	52
A.7.22.	TransportError	52
A.7.23.	CryptoError	53
Appendix B.	HTTP/3 data field definitions	53
B.1.	HTTP/3 Frames	53
B.1.1.	DataFrame	53
B.1.2.	HeadersFrame	54
B.1.3.	CancelPushFrame	54
B.1.4.	SettingsFrame	54
B.1.5.	PushPromiseFrame	54
B.1.6.	GoAwayFrame	55
B.1.7.	MaxPushIDFrame	55
B.1.8.	DuplicatePushFrame	55
B.1.9.	ReservedFrame	55
B.1.10.	UnknownFrame	55
B.2.	ApplicationError	55
Appendix C.	QPACK DATA type definitions	56
C.1.	QPACK Instructions	56
C.1.1.	SetDynamicTableCapacityInstruction	56
C.1.2.	InsertWithNameReferenceInstruction	56
C.1.3.	InsertWithoutNameReferenceInstruction	57
C.1.4.	DuplicateInstruction	57
C.1.5.	HeaderAcknowledgementInstruction	57
C.1.6.	StreamCancellationInstruction	57
C.1.7.	InsertCountIncrementInstruction	58
C.2.	QPACK Header compression	58
C.2.1.	IndexedHeaderField	58
C.2.2.	LiteralHeaderFieldWithName	58
C.2.3.	LiteralHeaderFieldWithoutName	59
C.2.4.	QPackHeaderBlockPrefix	59
Appendix D.	Change Log	59
D.1.	Since draft-01:	59
D.2.	Since draft-00:	61
Appendix E.	Design Variations	61
Appendix F.	Acknowledgements	61
Author's Address	61

1. Introduction

This document describes the values of the qlog name ("category" + "event") and "data" fields and their semantics for the QUIC and HTTP/3 protocols. This document is based on draft-29 of the QUIC and HTTP/3 I-Ds QUIC-TRANSPORT [QUIC-HTTP] and draft-16 of the QPACK I-D [QUIC-QPACK].

Feedback and discussion welcome at <https://github.com/quiclog/internet-drafts> (<https://github.com/quiclog/internet-drafts>). Readers are advised to refer to the "editor's draft" at that URL for an up-to-date version of this document.

Concrete examples of integrations of this schema in various programming languages can be found at <https://github.com/quiclog/qlog/> (<https://github.com/quiclog/qlog/>).

1.1. Notational Conventions

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

The examples and data definitions in this document are expressed in a custom data definition language, inspired by JSON and TypeScript, and described in [QLOG-MAIN].

2. Overview

This document describes the values of the qlog "name" ("category" + "event") and "data" fields and their semantics for the QUIC and HTTP/3 protocols.

This document assumes the usage of the encompassing main qlog schema defined in [QLOG-MAIN]. Each subsection below defines a separate category (for example connectivity, transport, http) and each subsubsection is an event type (for example "packet_received").

For each event type, its importance and data definition is laid out, often accompanied by possible values for the optional "trigger" field. For the definition and semantics of "trigger", see the main schema document.

Most of the complex datastructures, enums and re-usable definitions are grouped together on the bottom of this document for clarity.

2.1. Importance

Many of the events defined in this document map directly to concepts seen in the QUIC and HTTP/3 documents, while others act as aggregating events that combine data from several possible protocol behaviours or code paths into one. This is done to reduce the amount of unique event definitions, as reflecting each possible protocol event as a separate qlog entity would cause an explosion of event types. Similarly, we prevent logging duplicate packet data as much as possible. As such, especially packet header value updates are split out into separate events (for example `spin_bit_updated`, `connection_id_updated`), as they are expected to change sparingly.

Consequently, many events that can be directly inferred from data on the wire (for example flow control limit changes) if the implementation is bug-free, are currently not explicitly defined as stand-alone events. Exceptions can be made for common events that benefit from being easily identifiable or individually logged (for example the `"packets_acked"` event). This can in turn give rise to separate events logging similar data, where it is not always clear which event should be logged (for example the separate `"connection_started"` event, whereas the more general `"connection_state_updated"` event also allows indicating that a connection was started).

To aid in this decision making, each event has an "importance indicator" with one of three values, in decreasing order of importance and expected usage:

- * Core
- * Base
- * Extra

The "Core" events are the events that SHOULD be present in all qlog files. These are mostly tied to basic packet and frame parsing and creation, as well as listing basic internal metrics. Tool implementers SHOULD expect and add support for these events, though SHOULD NOT expect all Core events to be present in each qlog trace.

The "Base" events add additional debugging options and CAN be present in qlog files. Most of these can be implicitly inferred from data in Core events (if those contain all their properties), but for many it is better to log the events explicitly as well, making it clearer how the implementation behaves. These events are for example tied to passing data around in buffers, to how internal state machines change and help show when decisions are actually made based on received data. Tool implementers SHOULD at least add support for showing the contents of these events, if they do not handle them explicitly.

The "Extra" events are considered mostly useful for low-level debugging of the implementation, rather than the protocol. They allow more fine-grained tracking of internal behaviour. As such, they CAN be present in qlog files and tool implementers CAN add support for these, but they are not required to.

Note that in some cases, implementers might not want to log for example frame-level details in the "Core" events due to performance or privacy considerations. In this case, they SHOULD use (a subset of) relevant "Base" events instead to ensure usability of the qlog output. As an example, implementations that do not log "packet_received" events and thus also not which (if any) ACK frames the packet contain, SHOULD log "packets_acked" events instead.

Finally, for event types whose data (partially) overlap with other event types' definitions, where necessary this document includes guidance on which to use in specific situations.

2.2. Custom fields

Note that implementers are free to define new category and event types, as well as values for the "trigger" property within the "data" field, or other member fields of the "data" field, as they see fit. They SHOULD NOT however expect non-specialized tools to recognize or visualize this custom data. However, tools SHOULD make an effort to visualize even unknown data if possible in the specific tool's context.

3. Events not belonging to a single connection

For several types of events, it is sometimes impossible to tie them to a specific conceptual QUIC connection (e.g., a packet_dropped event triggered because the packet has an unknown connection_id in the header). Since qlog events in a trace are typically associated with a single connection, it is unclear how to log these events.

Ideally, implementers SHOULD create a separate, individual "endpoint-level" trace file (or group_id value), not associated with a specific connection (for example a "server.qlog" or group_id = "client"), and log all events that do not belong to a single connection to this grouping trace. However, this is not always practical, depending on the implementation. Because the semantics of most of these events are well-defined in the protocols and because they are difficult to mis-interpret as belonging to a connection, implementers MAY choose to log events not belonging to a particular connection in any other trace, even those strongly associated with a single connection.

Note that this can make it difficult to match logs from different vantage points with each other. For example, from the client side, it is easy to log connections with version negotiation or retry in the same trace, while on the server they would most likely be logged in separate traces. Servers can take extra efforts (and keep additional state) to keep these events combined in a single trace however (for example by also matching connections on their four-tuple instead of just the connection ID).

4. QUIC and HTTP/3 fields

This document re-uses all the fields defined in the main qlog schema (e.g., name, category, type, data, group_id, protocol_type, the time-related fields, etc.).

The value of the "protocol_type" qlog field MUST be "QUIC_HTTP3".

When the qlog "group_id" field is used, it is recommended to use QUIC's Original Destination Connection ID (ODCID, the CID chosen by the client when first contacting the server), as this is the only value that does not change over the course of the connection and can be used to link more advanced QUIC packets (e.g., Retry, Version Negotiation) to a given connection. Similarly, the ODCID should be used as the qlog filename or file identifier, potentially suffixed by the vantagepoint type (For example, abcd1234_server.qlog would contain the server-side trace of the connection with ODCID abcd1234).

4.1. Raw packet and frame information

While qlog is a more high-level logging format, it also allows the inclusion of most raw wire image information, such as byte lengths and even raw byte values. This can be useful when for example investigating or tuning packetization behaviour or determining encoding/framing overheads. However, these fields are not always necessary and can take up considerable space if logged for each packet or frame. As such, they are grouped in a separate optional field called "raw" of type RawInfo (where applicable).

```
class RawInfo {  
    length?:uint64; // full packet/frame length, including header and AEAD authentication tag lengths (where applicable)  
    payload_length?:uint64; // length of the packet/frame payload, excluding AEAD tag. For many control frames, this will have a value of zero  
  
    data?:bytes; // full packet/frame contents, including header and AEAD authentication tag (where applicable)  
}
```

Note: QUIC packets always include an AEAD authentication tag at the end. As this tag is always the same size for a given connection (it depends on the used TLS cipher), we do not have a separate "aead_tag_length" field here. Instead, this field is reflected in "transport:parameters_set" and can be logged only once.

Note: There is intentionally no explicit header_length field in RawInfo. QUIC and HTTP/3 use many Variable-Length Integer Encoded (VLIE) values in their packet and frame headers, which are of a dynamic length. Note too that because of this, we cannot deterministically reconstruct the header encoding/length from qlog data, as implementations might not necessarily employ the most efficient VLIE scheme for all values. As such, it is typically easier to log just the total packet/frame length and the payload length. The header length can be calculated by tools as:

For QUIC packets: $\text{header_length} = \text{length} - \text{payload_length} - \text{aead_tag_length}$

For QUIC and HTTP/3 frames: $\text{header_length} = \text{length} - \text{payload_length}$

For UDP datagrams: $\text{header_length} = \text{length} - \text{payload_length}$

Note: In some cases, the length fields are also explicitly reflected inside of frame/packet headers. For example, the QUIC STREAM frame has a "length" field indicating its payload size. Similarly, all HTTP/3 frames include their explicit payload lengths in the frame header. Finally, the QUIC Long Header has a "length" field which is equal to the payload length plus the packet number length. In these cases, those fields are intentionally preserved in the event definitions. Even though this can lead to duplicate data when the full RawInfo is logged, it allows a more direct mapping of the QUIC and HTTP/3 specifications to qlog, making it easier for users to interpret.

Note: as described in [QLOG-MAIN], the RawInfo:data field can be truncated for privacy or security purposes (for example excluding payload data). In this case, the length properties should still indicate the non-truncated lengths.

5. QUIC event definitions

Each subheading in this section is a qlog event category, while each sub-subheading is a qlog event type. Concretely, for the following two items, we have the category "connectivity" and event type "server_listening", resulting in a concatenated qlog "name" field value of "connectivity:server_listening".

5.1. connectivity

5.1.1. server_listening

Importance: Extra

Emitted when the server starts accepting connections.

Data:

```
{
  ip_v4?: IPAddress,
  ip_v6?: IPAddress,
  port_v4?: uint32,
  port_v6?: uint32,

  retry_required?:boolean // the server will always answer client initials with
  a retry (no 1-RTT connection setups by choice)
}
```

Note: some QUIC stacks do not handle sockets directly and are thus unable to log IP and/or port information.

5.1.2. connection_started

Importance: Base

Used for both attempting (client-perspective) and accepting (server-perspective) new connections. Note that this event has overlap with `connection_state_updated` and this is a separate event mainly because of all the additional data that should be logged.

Data:

```

{
  ip_version?: "v4" | "v6",
  src_ip?: IPAddress,
  dst_ip?: IPAddress,

  protocol?: string, // transport layer protocol (default "QUIC")
  src_port?: uint32,
  dst_port?: uint32,

  src_cid?: bytes,
  dst_cid?: bytes,
}

```

Note: some QUIC stacks do not handle sockets directly and are thus unable to log IP and/or port information.

5.1.3. connection_closed

Importance: Base

Used for logging when a connection was closed, typically when an error or timeout occurred. Note that this event has overlap with `connectivity:connection_state_updated`, as well as the `CONNECTION_CLOSE` frame. However, in practice, when analyzing large deployments, it can be useful to have a single event representing a `connection_closed` event, which also includes an additional reason field to provide additional information. Additionally, it is useful to log closures due to timeouts, which are difficult to reflect using the other options.

In QUIC there are two main connection-closing error categories: connection and application errors. They have well-defined error codes and semantics. Next to these however, there can be internal errors that occur that may or may not get mapped to the official error codes in implementation-specific ways. As such, multiple error codes can be set on the same event to reflect this.

```

{
  owner?: "local" | "remote", // which side closed the connection

  connection_code?: TransportError | CryptoError | uint32,
  application_code?: ApplicationError | uint32,
  internal_code?: uint32,

  reason?: string
}

```

Triggers: * clean * handshake_timeout * idle_timeout * error // this is called the "immediate close" in the QUIC specification * stateless_reset * version_mismatch * application // for example HTTP/3's GOAWAY frame

5.1.4. connection_id_updated

Importance: Base

This event is emitted when either party updates their current Connection ID. As this typically happens only sparingly over the course of a connection, this event allows loggers to be more efficient than logging the observed CID with each packet in the .header field of the "packet_sent" or "packet_received" events.

This is viewed from the perspective of the one applying the new id. As such, if we receive a new connection id from our peer, we will see the dst_ fields are set. If we update our own connection id (e.g., NEW_CONNECTION_ID frame), we log the src_ fields.

Data:

```
{
  owner: "local" | "remote",

  old?:bytes,
  new?:bytes,
}
```

5.1.5. spin_bit_updated

Importance: Base

To be emitted when the spin bit changes value. It SHOULD NOT be emitted if the spin bit is set without changing its value.

Data:

```
{
  state: boolean
}
```

5.1.6. connection_retried

TODO

5.1.7. connection_state_updated

Importance: Base

This event is used to track progress through QUIC's complex handshake and connection close procedures. It is intended to provide exhaustive options to log each state individually, but also provides a more basic, simpler set for implementations less interested in tracking each smaller state transition. As such, users should not expect to see -all- these states reflected in all qlogs and implementers should focus on support for the SimpleConnectionState set.

Data: ~~~ { old?: ConnectionState | SimpleConnectionState, new: ConnectionState | SimpleConnectionState }

```
enum ConnectionState { attempted, // initial sent/received
peer_validated, // peer address validated by: client sent Handshake
packet OR client used CONNID chosen by the server. transport-draft-
32, section-8.1 handshake_started, early_write, // 1 RTT can be sent,
but handshake isn't done yet handshake_complete, // TLS handshake
complete: Finished received and sent. tls-draft-32, section-4.1.1
handshake_confirmed, // HANDSHAKE_DONE sent/received (connection is
now "active", 1RTT can be sent). tls-draft-32, section-4.1.2 closing,
draining, // connection_close sent/received closed // draining period
done, connection state discarded }
```

```
enum SimpleConnectionState { attempted, handshake_started,
handshake_confirmed, closed } ~~~
```

These states correspond to the following transitions for both client and server:

Client:

* send initial

- state = attempted

* get initial

- state = validated _(not really "needed" at the client, but somewhat useful to indicate progress nonetheless)_

* get first Handshake packet

- state = handshake_started

- * get Handshake packet containing ServerFinished
 - state = handshake_complete
- * send ClientFinished
 - state = early_write (1RTT can now be sent)
- * get HANDSHAKE_DONE
 - state = handshake_confirmed
- *Server:*
- * get initial
 - state = attempted
- * send initial _(don't think this needs a separate state, since some handshake will always be sent in the same flight as this?)_
- * send handshake EE, CERT, CV, ...
 - state = handshake_started
- * send ServerFinished
 - state = early_write (1RTT can now be sent)
- * get first handshake packet / something using a server-issued CID of min length
 - state = validated
- * get handshake packet containing ClientFinished
 - state = handshake_complete
- * send HANDSHAKE_DONE
 - state = handshake_confirmed

Note: connection_state_changed with a new state of "attempted" is the same conceptual event as the connection_started event above from the client's perspective. Similarly, a state of "closing" or "draining" corresponds to the connection_closed event.

5.1.8. MIGRATION-related events

e.g., path_updated

TODO: read up on the draft how migration works and whether to best fit this here or in TRANSPORT TODO: integrate
<https://tools.ietf.org/html/draft-deconinck-quic-multipath-02>

For now, infer from other connectivity events and path_challenge/
path_response frames

5.2. security

5.2.1. key_updated

Importance: Base

Note: secret_updated would be more correct, but in the draft it's called KEY_UPDATE, so stick with that for consistency

Data:

```
{
  key_type:KeyType,
  old?:bytes,
  new:bytes,
  generation?:uint32 // needed for 1RTT key updates
}
```

Triggers:

- * "tls" // (e.g., initial, handshake and 0-RTT keys are generated by TLS)
- * "remote_update"
- * "local_update"

5.2.2. key_retired

Importance: Base

Data:

```
{
    key_type:KeyType,
    key?:bytes,
    generation?:uint32 // needed for 1RTT key updates
}
```

Triggers:

- * "tls" // (e.g., initial, handshake and 0-RTT keys are dropped implicitly)
- * "remote_update"
- * "local_update"

5.3. transport

5.3.1. version_information

Importance: Core

QUIC endpoints each have their own list of of QUIC versions they support. The client uses the most likely version in their first initial. If the server does support that version, it replies with a version_negotiation packet, containing supported versions. From this, the client selects a version. This event aggregates all this information in a single event type. It also allows logging of supported versions at an endpoint without actual version negotiation needing to happen.

Data:

```
{
    server_versions?:Array<bytes>,
    client_versions?:Array<bytes>,
    chosen_version?:bytes
}
```

Intended use:

- * When sending an initial, the client logs this event with client_versions and chosen_version set
- * Upon receiving a client initial with a supported version, the server logs this event with server_versions and chosen_version set

- * Upon receiving a client initial with an unsupported version, the server logs this event with `server_versions` set and `client_versions` to the single-element array containing the client's attempted version. The absence of `chosen_version` implies no overlap was found.
- * Upon receiving a version negotiation packet from the server, the client logs this event with `client_versions` set and `server_versions` to the versions in the version negotiation packet and `chosen_version` to the version it will use for the next initial packet

5.3.2. `alpn_information`

Importance: Core

QUIC implementations each have their own list of application level protocols and versions thereof they support. The client includes a list of their supported options in its first initial as part of the TLS Application Layer Protocol Negotiation (alpn) extension. If there are common option(s), the server chooses the most optimal one and communicates this back to the client. If not, the connection is closed.

Data:

```
{
  server_alpns?:Array<string>,
  client_alpns?:Array<string>,
  chosen_alpn?:string
}
```

Intended use:

- * When sending an initial, the client logs this event with `client_alpns` set
- * When receiving an initial with a supported alpn, the server logs this event with `server_alpns` set, `client_alpns` equalling the client-provided list, and `chosen_alpn` to the value it will send back to the client.
- * When receiving an initial with an alpn, the client logs this event with `chosen_alpn` to the received value.
- * Alternatively, a client can choose to not log the first event, but wait for the receipt of the server initial to log this event with both `client_alpns` and `chosen_alpn` set.

5.3.3. parameters_set

Importance: Core

This event groups settings from several different sources (transport parameters, TLS ciphers, etc.) into a single event. This is done to minimize the amount of events and to decouple conceptual setting impacts from their underlying mechanism for easier high-level reasoning.

All these settings are typically set once and never change. However, they are typically set at different times during the connection, so there will typically be several instances of this event with different fields set.

Note that some settings have two variations (one set locally, one requested by the remote peer). This is reflected in the "owner" field. As such, this field **MUST** be correct for all settings included a single event instance. If you need to log settings from two sides, you **MUST** emit two separate event instances.

In the case of connection resumption and 0-RTT, some of the server's parameters are stored up-front at the client and used for the initial connection startup. They are later updated with the server's reply. In these cases, utilize the separate "parameters_restored" event to indicate the initial values, and this event to indicate the updated values, as normal.

Data:

```

{
    owner?: "local" | "remote",

    resumption_allowed?: boolean, // valid session ticket was received
    early_data_enabled?: boolean, // early data extension was enabled on the TLS layer
    tls_cipher?: string, // (e.g., "AES_128_GCM_SHA256")
    aead_tag_length?: uint8, // depends on the TLS cipher, but it's easier to be explicit. Default value is 16

    // transport parameters from the TLS layer:
    original_destination_connection_id?: bytes,
    initial_source_connection_id?: bytes,
    retry_source_connection_id?: bytes,
    stateless_reset_token?: Token,
    disable_active_migration?: boolean,

    max_idle_timeout?: uint64,
    max_udp_payload_size?: uint32,
    ack_delay_exponent?: uint16,
    max_ack_delay?: uint16,
    active_connection_id_limit?: uint32,

    initial_max_data?: uint64,
    initial_max_stream_data_bidi_local?: uint64,
    initial_max_stream_data_bidi_remote?: uint64,
    initial_max_stream_data_uni?: uint64,
    initial_max_streams_bidi?: uint64,
    initial_max_streams_uni?: uint64,

    preferred_address?: PreferredAddress
}

interface PreferredAddress {
    ip_v4: IPAddress,
    ip_v6: IPAddress,

    port_v4: uint16,
    port_v6: uint16,

    connection_id: bytes,
    stateless_reset_token: Token
}

```

Additionally, this event can contain any number of unspecified fields. This is to reflect setting of for example unknown (greased) transport parameters or employed (proprietary) extensions.

5.3.4. parameters_restored

Importance: Base

When using QUIC 0-RTT, clients are expected to remember and restore the server's transport parameters from the previous connection. This event is used to indicate which parameters were restored and to which values when utilizing 0-RTT. Note that not all transport parameters should be restored (many are even prohibited from being re-utilized). The ones listed here are the ones expected to be useful for correct 0-RTT usage.

Data:

```
{
  disable_active_migration?:boolean,

  max_idle_timeout?:uint64,
  max_udp_payload_size?:uint32,
  active_connection_id_limit?:uint32,

  initial_max_data?:uint64,
  initial_max_stream_data_bidi_local?:uint64,
  initial_max_stream_data_bidi_remote?:uint64,
  initial_max_stream_data_uni?:uint64,
  initial_max_streams_bidi?:uint64,
  initial_max_streams_uni?:uint64,
}
```

Note that, like parameters_set above, this event can contain any number of unspecified fields to allow for additional/custom parameters.

5.3.5. packet_sent

Importance: Core

Data:

```
{
  header:PacketHeader,

  frames?:Array<QuicFrame>, // see appendix for the definitions

  is_coalesced?:boolean, // default value is false

  retry_token?:Token, // only if header.packet_type === retry

  stateless_reset_token?:bytes, // only if header.packet_type === stateless_reset. Is always 128 bits in length.

  supported_versions:Array<bytes>, // only if header.packet_type === version_negotiation

  raw?:RawInfo,
  datagram_id?:uint32
}
```

Note: We do not explicitly log the encryption_level or packet_number_space: the header.packet_type specifies this by inference (assuming correct implementation)

Triggers:

- * "retransmit_reordered" // draft-23 5.1.1
- * "retransmit_timeout" // draft-23 5.1.2
- * "pto_probe" // draft-23 5.3.1
- * "retransmit_crypto" // draft-19 6.2
- * "cc_bandwidth_probe" // needed for some CCs to figure out bandwidth allocations when there are no normal sends

Note: for more details on "datagram_id", see Section 5.3.10. It is only needed when keeping track of packet coalescing.

5.3.6. packet_received

Importance: Core

Data:

```
{
  header:PacketHeader,

  frames?:Array<QuicFrame>, // see appendix for the definitions

  is_coalesced?:boolean,

  retry_token?:Token, // only if header.packet_type === retry

  stateless_reset_token?:bytes, // only if header.packet_type === stateless_res
et. Is always 128 bits in length.

  supported_versions:Array<bytes>, // only if header.packet_type === version_ne
gotiation

  raw?:RawInfo,
  datagram_id?:uint32
}
```

Note: We do not explicitly log the encryption_level or packet_number_space: the header.packet_type specifies this by inference (assuming correct implementation)

Triggers:

* "keys_available" // if packet was buffered because it couldn't be decrypted before

Note: for more details on "datagram_id", see Section 5.3.10. It is only needed when keeping track of packet coalescing.

5.3.7. packet_dropped

Importance: Base

This event indicates a QUIC-level packet was dropped after partial or no parsing.

Data:

```
{
  header?:PacketHeader, // primarily packet_type should be filled here, as othe
r fields might not be parseable

  raw?:RawInfo,
  datagram_id?:uint32
}
```

For this event, the "trigger" field SHOULD be set (for example to one of the values below), as this helps tremendously in debugging.

Triggers:

- * "key_unavailable"
- * "unknown_connection_id"
- * "header_parse_error"
- * "payload_decrypt_error"
- * "protocol_violation"
- * "dos_prevention"
- * "unsupported_version"
- * "unexpected_packet"
- * "unexpected_source_connection_id"
- * "unexpected_version"
- * "duplicate"
- * "invalid_initial"

Note: sometimes packets are dropped before they can be associated with a particular connection (e.g., in case of "unsupported_version"). This situation is discussed more in Section 3.

Note: for more details on "datagram_id", see Section 5.3.10. It is only needed when keeping track of packet coalescing.

5.3.8. packet_buffered

Importance: Base

This event is emitted when a packet is buffered because it cannot be processed yet. Typically, this is because the packet cannot be parsed yet, and thus we only log the full packet contents when it was parsed in a packet_received event.

Data:

```
{
  header?:PacketHeader, // primarily packet_type and possible packet_number should
  // be filled here, as other elements might not be available yet

  raw?:RawInfo,
  datagram_id?:uint32
}
```

Note: for more details on "datagram_id", see Section 5.3.10. It is only needed when keeping track of packet coalescing.

Triggers:

- * "backpressure" // indicates the parser cannot keep up, temporarily buffers packet for later processing
- * "keys_unavailable" // if packet cannot be decrypted because the proper keys were not yet available

5.3.9. packets_acked

Importance: Extra

This event is emitted when a (group of) sent packet(s) is acknowledged by the remote peer `_for the first time_`. This information could also be deduced from the contents of received ACK frames. However, ACK frames require additional processing logic to determine when a given packet is acknowledged for the first time, as QUIC uses ACK ranges which can include repeated ACKs. Additionally, this event can be used by implementations that do not log frame contents.

Data: ~~~ { packet_number_space?:PacketNumberSpace,
packet_numbers?:Array<uint64> } ~~~

Note: if packet_number_space is omitted, it assumes the default value of PacketNumberSpace.application_data, as this is by far the most prevalent packet number space a typical QUIC connection will use.

5.3.10. datagrams_sent

Importance: Extra

When we pass one or more UDP-level datagrams to the socket. This is useful for determining how QUIC packet buffers are drained to the OS.

Data:

```

{
    count?:uint16, // to support passing multiple at once
    raw?:Array<RawInfo>, // RawInfo:length field indicates total length of the da
tagrams, including UDP header length

    datagram_ids?:Array<uint32>
}

```

Note: QUIC itself does not have a concept of a "datagram_id". This field is a purely qlong-specific construct to allow tracking how multiple QUIC packets are coalesced inside of a single UDP datagram, which is an important optimization during the QUIC handshake. For this, implementations assign a (per-endpoint) unique ID to each datagram and keep track of which packets were coalesced into the same datagram. As packet coalescing typically only happens during the handshake (as it requires at least one long header packet), this can be done without much overhead.

5.3.11. datagrams_received

Importance: Extra

When we receive one or more UDP-level datagrams from the socket. This is useful for determining how datagrams are passed to the user space stack from the OS.

Data:

```

{
    count?:uint16, // to support passing multiple at once
    raw?:Array<RawInfo>, // RawInfo:length field indicates total length of the da
tagrams, including UDP header length

    datagram_ids?:Array<uint32>
}

```

Note: for more details on "datagram_ids", see Section 5.3.10.

5.3.12. datagram_dropped

Importance: Extra

When we drop a UDP-level datagram. This is typically if it does not contain a valid QUIC packet (in that case, use packet_dropped instead).

Data:

```
{  
    raw?:RawInfo  
}
```

5.3.13. stream_state_updated

Importance: Base

This event is emitted whenever the internal state of a QUIC stream is updated, as described in QUIC transport draft-23 section 3. Most of this can be inferred from several types of frames going over the wire, but it's much easier to have explicit signals for these state changes.

Data:

```
{
    stream_id:uint64,
    stream_type?:"unidirectional"|"bidirectional", // mainly useful when opening
the stream

    old?:StreamState,
    new:StreamState,

    stream_side?:"sending"|"receiving"
}

enum StreamState {
    // bidirectional stream states, draft-23 3.4.
    idle,
    open,
    half_closed_local,
    half_closed_remote,
    closed,

    // sending-side stream states, draft-23 3.1.
    ready,
    send,
    data_sent,
    reset_sent,
    reset_received,

    // receive-side stream states, draft-23 3.2.
    receive,
    size_known,
    data_read,
    reset_read,

    // both-side states
    data_received,

    // qlog-defined
    destroyed // memory actually freed
}
```

Note: QUIC implementations SHOULD mainly log the simplified bidirectional (HTTP/2-alike) stream states (e.g., idle, open, closed) instead of the more finegrained stream states (e.g., data_sent, reset_received). These latter ones are mainly for more in-depth debugging. Tools SHOULD be able to deal with both types equally.

5.3.14. frames_processed

Importance: Extra

This event's main goal is to prevent a large proliferation of specific purpose events (e.g., `packets_acknowledged`, `flow_control_updated`, `stream_data_received`). We want to give implementations the opportunity to (selectively) log this type of signal without having to log packet-level details (e.g., in `packet_received`). Since for almost all cases, the effects of applying a frame to the internal state of an implementation can be inferred from that frame's contents, we aggregate these events in this single `"frames_processed"` event.

Note: This event can be used to signal internal state change not resulting directly from the actual "parsing" of a frame (e.g., the frame could have been parsed, data put into a buffer, then later processed, then logged with this event).

Note: Implementations logging `"packet_received"` and which include all of the packet's constituent frames therein, are not expected to emit this `"frames_processed"` event (contrary to the HTTP-level `"frames_parsed"` event). Rather, implementations not wishing to log full packets or that wish to explicitly convey extra information about when frames are processed (if not directly tied to their reception) can use this event.

Note: for some events, this approach will lose some information (e.g., for which encryption level are packets being acknowledged?). If this information is important, please use the `packet_received` event instead.

Note: in some implementations, it can be difficult to log frames directly, even when using `packet_sent` and `packet_received` events. For these cases, this event also contains the direct `packet_number` field, which can be used to more explicitly link this event to the `packet_sent/received` events.

Data:

```
{
  frames:Array<QuicFrame>, // see appendix for the definitions
  packet_number?:uint64
}
```

5.3.15. `data_moved`

Importance: Base

Used to indicate when data moves between the different layers (for example passing from HTTP/3 to QUIC stream buffers and vice versa) or between HTTP/3 and the actual user application on top (for example a browser engine). This helps make clear the flow of data, how long data remains in various buffers and the overheads introduced by individual layers.

For example, this helps make clear whether received data on a QUIC stream is moved to the HTTP layer immediately (for example per received packet) or in larger batches (for example, all QUIC packets are processed first and afterwards the HTTP layer reads from the streams with newly available data). This in turn can help identify bottlenecks or scheduling problems.

Data:

```
{
  stream_id?:uint64,
  offset?:uint64,
  length?:uint64, // byte length of the moved data

  from?:string, // typically: use either of "application","http","transport"
  to?:string, // typically: use either of "application","http","transport"

  data?:bytes // raw bytes that were transferred
}
```

Note: we do not for example use a "direction" field (with values "up" and "down") to specify the data flow. This is because in some optimized implementations, data might skip some individual layers. Additionally, using explicit "from" and "to" fields is more flexible and allows the definition of other conceptual "layers" (for example to indicate data from QUIC CRYPTO frames being passed to a TLS library ("security") or from HTTP/3 to QPACK ("qpack")).

Note: this event type is part of the "transport" category, but really spans all the different layers. This means we have a few leaky abstractions here (for example, the stream_id or stream offset might not be available at some logging points, or the raw data might not be in a byte-array form). In these situations, implementers can decide to define new, in-context fields to aid in manual debugging.

5.4. recovery

Note: most of the events in this category are kept generic to support different recovery approaches and various congestion control algorithms. Tool creators SHOULD make an effort to support and visualize even unknown data in these events (e.g., plot unknown congestion states by name on a timeline visualization).

5.4.1. parameters_set

Importance: Base

This event groups initial parameters from both loss detection and congestion control into a single event. All these settings are typically set once and never change. Implementation that do, for some reason, change these parameters during execution, MAY emit the parameters_set event twice.

Data:

```
{
  // Loss detection, see recovery draft-23, Appendix A.2
  reordering_threshold?:uint16, // in amount of packets
  time_threshold?:float, // as RTT multiplier
  timer_granularity?:uint16, // in ms
  initial_rtt?:float, // in ms

  // congestion control, Appendix B.1.
  max_datagram_size?:uint32, // in bytes // Note: this could be updated after p
mtud
  initial_congestion_window?:uint64, // in bytes
  minimum_congestion_window?:uint32, // in bytes // Note: this could change whe
n max_datagram_size changes
  loss_reduction_factor?:float,
  persistent_congestion_threshold?:uint16 // as PTO multiplier
}
```

Additionally, this event can contain any number of unspecified fields to support different recovery approaches.

5.4.2. metrics_updated

Importance: Core

This event is emitted when one or more of the observable recovery metrics changes value. This event SHOULD group all possible metric updates that happen at or around the same time in a single event (e.g., if `min_rtt` and `smoothed_rtt` change at the same time, they should be bundled in a single `metrics_updated` entry, rather than split out into two). Consequently, a `metrics_updated` event is only guaranteed to contain at least one of the listed metrics.

Data:

```
{
  // Loss detection, see recovery draft-23, Appendix A.3
  min_rtt?:float, // in ms or us, depending on the overarching qlog's configura
tion
  smoothed_rtt?:float, // in ms or us, depending on the overarching qlog's conf
iguration
  latest_rtt?:float, // in ms or us, depending on the overarching qlog's config
uration
  rtt_variance?:float, // in ms or us, depending on the overarching qlog's conf
iguration

  pto_count?:uint16,

  // Congestion control, Appendix B.2.
  congestion_window?:uint64, // in bytes
  bytes_in_flight?:uint64,

  ssthresh?:uint64, // in bytes

  // qlog defined
  packets_in_flight?:uint64, // sum of all packet number spaces

  pacing_rate?:uint64 // in bps
}
```

Note: to make logging easier, implementations MAY log values even if they are the same as previously reported values (e.g., two subsequent `METRIC_UPDATE` entries can both report the exact same value for `min_rtt`). However, applications SHOULD try to log only actual updates to values.

Additionally, this event can contain any number of unspecified fields to support different recovery approaches.

5.4.3. `congestion_state_updated`

Importance: Base

This event signifies when the congestion controller enters a significant new state and changes its behaviour. This event's definition is kept generic to support different Congestion Control algorithms. For example, for the algorithm defined in the Recovery draft ("enhanced" New Reno), the following states are defined:

- * slow_start
- * congestion_avoidance
- * application_limited
- * recovery

Data:

```
{
  old?:string,
  new:string
}
```

The "trigger" field SHOULD be logged if there are multiple ways in which a state change can occur but MAY be omitted if a given state can only be due to a single event occurring (e.g., slow start is exited only when ssthresh is exceeded).

Some triggers for ("enhanced" New Reno):

- * persistent_congestion
- * ECN

5.4.4. loss_timer_updated

Importance: Extra

This event is emitted when a recovery loss timer changes state. The three main event types are:

- * set: the timer is set with a delta timeout for when it will trigger next
- * expired: when the timer effectively expires after the delta timeout
- * cancelled: when a timer is cancelled (e.g., all outstanding packets are acknowledged, start idle period)

Note: to indicate an active timer's timeout update, a new "set" event is used.

Data:

```
{
  timer_type?: "ack" | "pto", // called "mode" in draft-23 A.9.
  packet_number_space?: PacketNumberSpace,

  event_type: "set" | "expired" | "cancelled",

  delta?: float // if event_type === "set": delta time in ms or us (see configur
ation) from this event's timestamp until when the timer will trigger
}
```

TODO: how about CC algo's that use multiple timers? How generic do these events need to be? Just support QUIC-style recovery from the spec or broader?

TODO: read up on the loss detection logic in draft-27 onward and see if this suffices

5.4.5. packet_lost

Importance: Core

This event is emitted when a packet is deemed lost by loss detection.

Data:

```
{
  header?: PacketHeader, // should include at least the packet_type and packet_n
umber

  // not all implementations will keep track of full packets, so these are opti
onal
  frames?: Array<QuicFrame> // see appendix for the definitions
}
```

For this event, the "trigger" field SHOULD be set (for example to one of the values below), as this helps tremendously in debugging.

Triggers:

- * "reordering_threshold",
- * "time_threshold"
- * "pto_expired" // draft-23 section 5.3.1, MAY

5.4.6. marked_for_retransmit

Importance: Extra

This event indicates which data was marked for retransmit upon detecting a packet loss (see `packet_lost`). Similar to our reasoning for the "frames_processed" event, in order to keep the amount of different events low, we group this signal for all types of retransmittable data in a single event based on existing QUIC frame definitions.

Implementations retransmitting full packets or frames directly can just log the constituent frames of the lost packet here (or do away with this event and use the contents of the `packet_lost` event instead). Conversely, implementations that have more complex logic (e.g., marking ranges in a stream's data buffer as in-flight), or that do not track sent frames in full (e.g., only stream offset + length), can translate their internal behaviour into the appropriate frame instance here even if that frame was never or will never be put on the wire.

Note: much of this data can be inferred if implementations log `packet_sent` events (e.g., looking at overlapping stream data offsets and length, one can determine when data was retransmitted).

Data:

```
{
  frames:Array<QuicFrame>, // see appendix for the definitions
}
```

6. HTTP/3 event definitions

6.1. http

Note: like all category values, the "http" category is written in lowercase.

6.1.1. parameters_set

Importance: Base

This event contains HTTP/3 and QPACK-level settings, mostly those received from the HTTP/3 SETTINGS frame. All these parameters are typically set once and never change. However, they are typically set at different times during the connection, so there can be several instances of this event with different fields set.

Note that some settings have two variations (one set locally, one requested by the remote peer). This is reflected in the "owner" field. As such, this field MUST be correct for all settings included a single event instance. If you need to log settings from two sides, you MUST emit two separate event instances.

Data:

```
{
  owner?: "local" | "remote",

  max_header_list_size?: uint64, // from SETTINGS_MAX_HEADER_LIST_SIZE
  max_table_capacity?: uint64, // from SETTINGS_QPACK_MAX_TABLE_CAPACITY
  blocked_streams_count?: uint64, // from SETTINGS_QPACK_BLOCKED_STREAMS

  // qlog-defined
  waits_for_settings?: boolean // indicates whether this implementation waits for
  a SETTINGS frame before processing requests
}
```

Note: enabling server push is not explicitly done in HTTP/3 by use of a setting or parameter. Instead, it is communicated by use of the MAX_PUSH_ID frame, which should be logged using the frame_created and frame_parsed events below.

Additionally, this event can contain any number of unspecified fields. This is to reflect setting of for example unknown (greased) settings or parameters of (proprietary) extensions.

6.1.2. parameters_restored

Importance: Base

When using QUIC 0-RTT, clients are expected to remember and reuse the server's SETTINGS from the previous connection. This event is used to indicate which settings were restored and to which values when utilizing 0-RTT.

Data:

```
{
  max_header_list_size?: uint64,
  max_table_capacity?: uint64,
  blocked_streams_count?: uint64
}
```

Note that, like for parameters_set above, this event can contain any number of unspecified fields to allow for additional and custom settings.

6.1.3. stream_type_set

Importance: Base

Emitted when a stream's type becomes known. This is typically when a stream is opened and the stream's type indicator is sent or received.

Note: most of this information can also be inferred by looking at a stream's id, since id's are strictly partitioned at the QUIC level. Even so, this event has a "Base" importance because it helps a lot in debugging to have this information clearly spelled out.

Data:

```
{
  stream_id:uint64,

  owner?:"local"|"remote"

  old?:StreamType,
  new:StreamType,

  associated_push_id?:uint64 // only when new == "push"
}

enum StreamType {
  data, // bidirectional request-response streams
  control,
  push,
  reserved,
  qpack_encode,
  qpack_decode
}
```

6.1.4. frame_created

Importance: Core

HTTP equivalent to the packet_sent event. This event is emitted when the HTTP/3 framing actually happens. Note: this is not necessarily the same as when the HTTP/3 data is passed on to the QUIC layer. For that, see the "data_moved" event.

Data:

```
{
  stream_id:uint64,
  length?:uint64, // payload byte length of the frame
  frame:HTTP3Frame, // see appendix for the definitions,

  raw?:RawInfo
}
```

Note: in HTTP/3, DATA frames can have arbitrarily large lengths to reduce frame header overhead. As such, DATA frames can span many QUIC packets and can be created in a streaming fashion. In this case, the `frame_created` event is emitted once for the frame header, and further streamed data is indicated using the `data_moved` event.

6.1.5. `frame_parsed`

Importance: Core

HTTP equivalent to the `packet_received` event. This event is emitted when we actually parse the HTTP/3 frame. Note: this is not necessarily the same as when the HTTP/3 data is actually received on the QUIC layer. For that, see the `"data_moved"` event.

Data:

```
{
  stream_id:uint64,
  length?:uint64, // payload byte length of the frame
  frame:HTTP3Frame, // see appendix for the definitions,

  raw?:RawInfo
}
```

Note: in HTTP/3, DATA frames can have arbitrarily large lengths to reduce frame header overhead. As such, DATA frames can span many QUIC packets and can be processed in a streaming fashion. In this case, the `frame_parsed` event is emitted once for the frame header, and further streamed data is indicated using the `data_moved` event.

6.1.6. `push_resolved`

Importance: Extra

This event is emitted when a pushed resource is successfully claimed (used) or, conversely, abandoned (rejected) by the application on top of HTTP/3 (e.g., the web browser). This event is added to help debug problems with unexpected PUSH behaviour, which is commonplace with HTTP/2.

```
{
    push_id?:uint64,
    stream_id?:uint64, // in case this is logged from a place that does not have
access to the push_id

    decision:"claimed"|"abandoned"
}
```

6.2. qpack

Note: like all category values, the "qpack" category is written in lowercase.

The QPACK events mainly serve as an aid to debug low-level QPACK issues. The higher-level, plaintext header values SHOULD (also) be logged in the http.frame_created and http.frame_parsed event data (instead).

Note: qpack does not have its own parameters_set event. This was merged with http.parameters_set for brevity, since qpack is a required extension for HTTP/3 anyway. Other HTTP/3 extensions MAY also log their SETTINGS fields in http.parameters_set or MAY define their own events.

6.2.1. state_updated

Importance: Base

This event is emitted when one or more of the internal QPACK variables changes value. Note that some variables have two variations (one set locally, one requested by the remote peer). This is reflected in the "owner" field. As such, this field MUST be correct for all variables included a single event instance. If you need to log settings from two sides, you MUST emit two separate event instances.

Data:

```
{
    owner:"local" | "remote",

    dynamic_table_capacity?:uint64,
    dynamic_table_size?:uint64, // effective current size, sum of all the entries

    known_received_count?:uint64,
    current_insert_count?:uint64
}
```

6.2.2. stream_state_updated

Importance: Core

This event is emitted when a stream becomes blocked or unblocked by header decoding requests or QPACK instructions.

Note: This event is of "Core" importance, as it might have a large impact on HTTP/3's observed performance.

Data:

```
{
  stream_id:uint64,

  state:"blocked"|"unblocked" // streams are assumed to start "unblocked" until
  they become "blocked"
}
```

6.2.3. dynamic_table_updated

Importance: Extra

This event is emitted when one or more entries are inserted or evicted from QPACK's dynamic table.

Data:

```
{
  owner:"local" | "remote", // local = the encoder's dynamic table. remote = th
  e decoder's dynamic table

  update_type:"inserted"|"evicted",

  entries:Array<DynamicTableEntry>
}

class DynamicTableEntry {
  index:uint64;
  name?:string | bytes;
  value?:string | bytes;
}
```

6.2.4. headers_encoded

Importance: Base

This event is emitted when an uncompressed header block is encoded successfully.

Note: this event has overlap with `http.frame_created` for the `HeadersFrame` type. When outputting both events, implementers MAY omit the "headers" field in this event.

Data:

```
{
  stream_id?:uint64,

  headers?:Array<HTTPHeader>,

  block_prefix:QPackHeaderBlockPrefix,
  header_block:Array<QPackHeaderBlockRepresentation>,

  length?:uint32,
  raw?:bytes
}
```

6.2.5. headers_decoded

Importance: Base

This event is emitted when a compressed header block is decoded successfully.

Note: this event has overlap with `http.frame_parsed` for the `HeadersFrame` type. When outputting both events, implementers MAY omit the "headers" field in this event.

Data:

```
{
  stream_id?:uint64,

  headers?:Array<HTTPHeader>,

  block_prefix:QPackHeaderBlockPrefix,
  header_block:Array<QPackHeaderBlockRepresentation>,

  length?:uint32,
  raw?:bytes
}
```

6.2.6. instruction_created

Importance: Base

This event is emitted when a QPACK instruction (both decoder and encoder) is created and added to the encoder/decoder stream.

Data:

```
{
  instruction:QPackInstruction // see appendix for the definitions,
  length?:uint32,
  raw?:bytes
}
```

Note: encoder/decoder semantics and stream_id's are implicit in either the instruction types or can be logged via other events (e.g., http.stream_type_set)

6.2.7. instruction_parsed

Importance: Base

This event is emitted when a QPACK instruction (both decoder and encoder) is read from the encoder/decoder stream.

Data:

```
{
  instruction:QPackInstruction // see appendix for the definitions,
  length?:uint32,
  raw?:bytes
}
```

Note: encoder/decoder semantics and stream_id's are implicit in either the instruction types or can be logged via other events (e.g., http.stream_type_set)

7. Generic events and Simulation indicators

7.1. generic

The main goal of the events in this category is to allow implementations to fully replace their existing text-based logging by qlog. This is done by providing events to log generic strings for typical well-known logging levels (error, warning, info, debug, verbose).

7.1.1. error

Importance: Core

Used to log details of an internal error. For errors that effectively lead to the closure of a QUIC connection, it is recommended to use `transport:connection_closed` instead.

Data:

```
{
  code?:uint32,
  message?:string
}
```

7.1.2. warning

Importance: Base

Used to log details of an internal warning that might not get reflected on the wire.

Data:

```
{
  code?:uint32,
  message?:string
}
```

7.1.3. info

Importance: Extra

Used mainly for implementations that want to use `qlog` as their one and only logging format but still want to support unstructured string messages.

Data:

```
{
  message:string
}
```

7.1.4. debug

Importance: Extra

Used mainly for implementations that want to use qlog as their one and only logging format but still want to support unstructured string messages.

Data:

```
{
  message:string
}
```

7.1.5. verbose

Importance: Extra

Used mainly for implementations that want to use qlog as their one and only logging format but still want to support unstructured string messages.

Data:

```
{
  message:string
}
```

7.2. simulation

When evaluating a protocol evaluation, one typically sets up a series of interoperability or benchmarking tests, in which the test situations can change over time. For example, the network bandwidth or latency can vary during the test, or the network can be fully disable for a short time. In these setups, it is useful to know when exactly these conditions are triggered, to allow for proper correlation with other events.

7.2.1. scenario

Importance: Extra

Used to specify which specific scenario is being tested at this particular instance. This could also be reflected in the top-level qlog's "summary" or "configuration" fields, but having a separate event allows easier aggregation of several simulations into one trace.

```
{
  name?:string,
  details?:any
}
```

7.2.2. marker

Importance: Extra

Used to indicate when specific emulation conditions are triggered at set times (e.g., at 3 seconds in 2% packet loss is introduced, at 10s a NAT rebind is triggered).

```
{
  type?:string,
  message?:string
}
```

8. Security Considerations

TBD

9. IANA Considerations

TBD

10. References

10.1. Normative References

[QLOG-MAIN]

Marx, R., Ed., "Main logging schema for qlog", Work in Progress, Internet-Draft, draft-marx-qlog-main-schema-02, 2 November 2020, <<https://tools.ietf.org/html/draft-marx-qlog-main-schema-02>>.

[QUIC-HTTP]

Bishop, M., Ed., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-32, 1 October 2020, <<https://tools.ietf.org/html/draft-ietf-quic-http-32>>.

[QUIC-QPACK]

Frindell, A., Ed., "QPACK: Header Compression for HTTP/3", Work in Progress, Internet-Draft, draft-ietf-quic-qpack-19, 20 October 2020, <<https://tools.ietf.org/html/draft-ietf-quic-qpack-19>>.

[QUIC-TRANSPORT]

Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-32, 1 October 2020, <<https://tools.ietf.org/html/draft-ietf-quic-transport-32>>.

10.2. Informative References

[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>>.

Appendix A. QUIC data field definitions

A.1. IPAddress

```
class IPAddress : string | bytes;
```

// an IPAddress can either be a "human readable" form (e.g., "127.0.0.1" for v4 or "2001:0db8:85a3:0000:0000:8a2e:0370:7334" for v6) or use a raw byte-form (as the string forms can be ambiguous)

A.2. PacketType

```
enum PacketType {  
    initial,  
    handshake,  
    zerortt = "0RTT",  
    onertt = "1RTT",  
    retry,  
    version_negotiation,  
    stateless_reset,  
    unknown  
}
```

A.3. PacketNumberSpace

```
enum PacketNumberSpace {  
    initial,  
    handshake,  
    application_data  
}
```

A.4. PacketHeader

```

class PacketHeader {
    // Note: short vs long header is implicit through PacketType

    packet_type: PacketType;
    packet_number: uint64;

    flags?: uint8; // the bit flags of the packet headers (spin bit, key update b
it, etc. up to and including the packet number length bits if present) interprete
d as a single 8-bit integer

    token?:Token; // only if packet_type == initial

    length?: uint16, // only if packet_type == initial || handshake || 0RTT. Sign
ifies length of the packet_number plus the payload.

    // only if present in the header
    // if correctly using transport:connection_id_updated events,
    // dcid can be skipped for 1RTT packets
    version?: bytes; // e.g., "ff00001d" for draft-29
    scil?: uint8;
    dcil?: uint8;
    scid?: bytes;
    dcid?: bytes;
}

```

A.5. Token

```

class Token {
    type?: "retry"|"resumption"|"stateless_reset";

    length?:uint32; // byte length of the token
    data?:bytes; // raw byte value of the token

    details?:any; // decoded fields included in the token (typically: peer's IP a
ddress, creation time)
}

```

The token carried in an Initial packet can either be a retry token from a Retry packet, a stateless reset token from a Stateless Reset packet or one originally provided by the server in a NEW_TOKEN frame used when resuming a connection (e.g., for address validation purposes). Retry and resumption tokens typically contain encoded metadata to check the token's validity when it is used, but this metadata and its format is implementation specific. For that, this field includes a general-purpose "details" field.

A.6. KeyType

```
enum KeyType {
    server_initial_secret,
    client_initial_secret,

    server_handshake_secret,
    client_handshake_secret,

    server_0rtt_secret,
    client_0rtt_secret,

    server_1rtt_secret,
    client_1rtt_secret
}
```

A.7. QUIC Frames

```
type QuicFrame = PaddingFrame | PingFrame | AckFrame | ResetStreamFrame | StopSendingFrame | CryptoFrame | NewTokenFrame | StreamFrame | MaxDataFrame | MaxStreamDataFrame | MaxStreamsFrame | DataBlockedFrame | StreamDataBlockedFrame | StreamsBlockedFrame | NewConnectionIDFrame | RetireConnectionIDFrame | PathChallengeFrame | PathResponseFrame | ConnectionCloseFrame | HandshakeDoneFrame | UnknownFrame;
```

A.7.1. PaddingFrame

In QUIC, PADDING frames are simply identified as a single byte of value 0. As such, each padding byte could be theoretically interpreted and logged as an individual `PaddingFrame`.

However, as this leads to heavy logging overhead, implementations SHOULD instead emit just a single `PaddingFrame` and set the `payload_length` property to the amount of PADDING bytes/frames included in the packet.

```
class PaddingFrame{
    frame_type:string = "padding";

    length?:uint32; // total frame length, including frame header
    payload_length?:uint32;
}
```

A.7.2. PingFrame

```
class PingFrame{
    frame_type:string = "ping";

    length?:uint32; // total frame length, including frame header
    payload_length?:uint32;
}
```

A.7.3. AckFrame

```

class AckFrame{
    frame_type:string = "ack";

    ack_delay?:float; // in ms

    // first number is "from": lowest packet number in interval
    // second number is "to": up to and including // highest packet number in interval
    // e.g., looks like [[1,2],[4,5]]
    acked_ranges?:Array<[uint64, uint64]|[uint64]>;

    // ECN (explicit congestion notification) related fields (not always present)
    ect1?:uint64;
    ect0?:uint64;
    ce?:uint64;

    length?:uint32; // total frame length, including frame header
    payload_length?:uint32;
}

```

Note: the packet ranges in `AckFrame.acked_ranges` do not necessarily have to be ordered (e.g., `[[5,9],[1,4]]` is a valid value).

Note: the two numbers in the packet range can be the same (e.g., `[120,120]` means that packet with number 120 was ACKed). However, in that case, implementers SHOULD log `[120]` instead and tools MUST be able to deal with both notations.

A.7.4. ResetStreamFrame

```

class ResetStreamFrame{
    frame_type:string = "reset_stream";

    stream_id:uint64;
    error_code:ApplicationError | uint32;
    final_size:uint64; // in bytes

    length?:uint32; // total frame length, including frame header
    payload_length?:uint32;
}

```

A.7.5. StopSendingFrame

```

class StopSendingFrame{
    frame_type:string = "stop_sending";

    stream_id:uint64;
    error_code:ApplicationError | uint32;

    length?:uint32; // total frame length, including frame header
    payload_length?:uint32;
}

```

A.7.6. CryptoFrame

```

class CryptoFrame{
    frame_type:string = "crypto";

    offset:uint64;
    length:uint64;

    payload_length?:uint32;
}

```

A.7.7. NewTokenFrame

```

class NewTokenFrame{
    frame_type:string = "new_token";

    token:Token
}

```

A.7.8. StreamFrame

```

class StreamFrame{
    frame_type:string = "stream";

    stream_id:uint64;

    // These two MUST always be set
    // If not present in the Frame type, log their default values
    offset:uint64;
    length:uint64;

    // this MAY be set any time, but MUST only be set if the value is "true"
    // if absent, the value MUST be assumed to be "false"
    fin?:boolean;

    raw?:bytes;
}

```

A.7.9. MaxDataFrame

```
class MaxDataFrame{
    frame_type:string = "max_data";

    maximum:uint64;
}
```

A.7.10. MaxStreamDataFrame

```
class MaxStreamDataFrame{
    frame_type:string = "max_stream_data";

    stream_id:uint64;
    maximum:uint64;
}
```

A.7.11. MaxStreamsFrame

```
class MaxStreamsFrame{
    frame_type:string = "max_streams";

    stream_type:string = "bidirectional" | "unidirectional";
    maximum:uint64;
}
```

A.7.12. DataBlockedFrame

```
class DataBlockedFrame{
    frame_type:string = "data_blocked";

    limit:uint64;
}
```

A.7.13. StreamDataBlockedFrame

```
class StreamDataBlockedFrame{
    frame_type:string = "stream_data_blocked";

    stream_id:uint64;
    limit:uint64;
}
```

A.7.14. StreamsBlockedFrame

```
class StreamsBlockedFrame{
    frame_type:string = "streams_blocked";

    stream_type:string = "bidirectional" | "unidirectional";
    limit:uint64;
}
```

A.7.15. NewConnectionIDFrame

```
class NewConnectionIDFrame{
    frame_type:string = "new_connection_id";

    sequence_number:uint32;
    retire_prior_to:uint32;

    connection_id_length?:uint8;
    connection_id:bytes;

    stateless_reset_token?:Token;
}
```

A.7.16. RetireConnectionIDFrame

```
class RetireConnectionIDFrame{
    frame_type:string = "retire_connection_id";

    sequence_number:uint32;
}
```

A.7.17. PathChallengeFrame

```
class PathChallengeFrame{
    frame_type:string = "path_challenge";

    data?:bytes; // always 64-bit
}
```

A.7.18. PathResponseFrame

```
class PathResponseFrame{
    frame_type:string = "path_response";

    data?:bytes; // always 64-bit
}
```

A.7.19. ConnectionCloseFrame

raw_error_code is the actual, numerical code. This is useful because some error types are spread out over a range of codes (e.g., QUIC's crypto_error).

```
type ErrorSpace = "transport" | "application";

class ConnectionCloseFrame{
  frame_type:string = "connection_close";

  error_space?:ErrorSpace;
  error_code?:TransportError | ApplicationError | uint32;
  raw_error_code?:uint32;
  reason?:string;

  trigger_frame_type?:uint64 | string; // For known frame types, the appropriate "frame_type" string. For unknown frame types, the hex encoded identifier value
}
```

A.7.20. HandshakeDoneFrame

```
class HandshakeDoneFrame{
  frame_type:string = "handshake_done";
}
```

A.7.21. UnknownFrame

```
class UnknownFrame{
  frame_type:string = "unknown";
  raw_frame_type:uint64;

  raw_length?:uint32;
  raw?:bytes;
}
```

A.7.22. TransportError

```
enum TransportError {
    no_error,
    internal_error,
    connection_refused,
    flow_control_error,
    stream_limit_error,
    stream_state_error,
    final_size_error,
    frame_encoding_error,
    transport_parameter_error,
    connection_id_limit_error,
    protocol_violation,
    invalid_token,
    application_error,
    crypto_buffer_exceeded
}
```

A.7.23. CryptoError

These errors are defined in the TLS document as "A TLS alert is turned into a QUIC connection error by converting the one-byte alert description into a QUIC error code. The alert description is added to 0x100 to produce a QUIC error code from the range reserved for CRYPTO_ERROR."

This approach maps badly to a pre-defined enum. As such, we define the `crypto_error` string as having a dynamic component here, which should include the hex-encoded value of the TLS alert description.

```
enum CryptoError {
    crypto_error_{TLS_ALERT}
}
```

Appendix B. HTTP/3 data field definitions

B.1. HTTP/3 Frames

```
type HTTP3Frame = DataFrame | HeadersFrame | PriorityFrame | CancelPushFrame | SettingsFrame | PushPromiseFrame | GoAwayFrame | MaxPushIDFrame | DuplicatePushFrame | ReservedFrame | UnknownFrame;
```

B.1.1. DataFrame

```
class DataFrame{
    frame_type:string = "data";

    raw?:bytes;
}
```

B.1.2. HeadersFrame

This represents an `_uncompressed_`, plaintext HTTP Headers frame (e.g., no QPACK compression is applied).

For example:

```
headers: [{"name":":path","value":"/"}, {"name":":method","value":"GET"}, {"name":":authority","value":"127.0.0.1:4433"}, {"name":":scheme","value":"https"}]
```

```
class HeadersFrame{
    frame_type:string = "header";
    headers:Array<HTTPHeader>;
}
```

```
class HTTPHeader {
    name:string;
    value:string;
}
```

B.1.3. CancelPushFrame

```
class CancelPushFrame{
    frame_type:string = "cancel_push";
    push_id:uint64;
}
```

B.1.4. SettingsFrame

```
class SettingsFrame{
    frame_type:string = "settings";
    settings:Array<Setting>;
}
```

```
class Setting{
    name:string;
    value:string;
}
```

B.1.5. PushPromiseFrame

```
class PushPromiseFrame{
    frame_type:string = "push_promise";
    push_id:uint64;

    headers:Array<HTTPHeader>;
}
```

B.1.6. GoAwayFrame

```
class GoAwayFrame{
    frame_type:string = "goaway";
    stream_id:uint64;
}
```

B.1.7. MaxPushIDFrame

```
class MaxPushIDFrame{
    frame_type:string = "max_push_id";
    push_id:uint64;
}
```

B.1.8. DuplicatePushFrame

```
class DuplicatePushFrame{
    frame_type:string = "duplicate_push";
    push_id:uint64;
}
```

B.1.9. ReservedFrame

```
class ReservedFrame{
    frame_type:string = "reserved";
}
```

B.1.10. UnknownFrame

HTTP/3 re-uses QUIC's UnknownFrame definition, since their values and usage overlaps.

B.2. ApplicationError

```
enum ApplicationError{
    http_no_error,
    http_general_protocol_error,
    http_internal_error,
    http_stream_creation_error,
    http_closed_critical_stream,
    http_frame_unexpected,
    http_frame_error,
    http_excessive_load,
    http_id_error,
    http_settings_error,
    http_missing_settings,
    http_request_rejected,
    http_request_cancelled,
    http_request_incomplete,
    http_early_response,
    http_connect_error,
    http_version_fallback
}
```

Appendix C. QPACK DATA type definitions

C.1. QPACK Instructions

Note: the instructions do not have explicit encoder/decoder types, since there is no overlap between the instructions of both types in neither name nor function.

```
type QPackInstruction = SetDynamicTableCapacityInstruction | InsertWithNameReferenceInstruction | InsertWithoutNameReferenceInstruction | DuplicateInstruction | HeaderAcknowledgementInstruction | StreamCancellationInstruction | InsertCountIncrementInstruction;
```

C.1.1. SetDynamicTableCapacityInstruction

```
class SetDynamicTableCapacityInstruction {
    instruction_type:string = "set_dynamic_table_capacity";

    capacity:uint32;
}
```

C.1.2. InsertWithNameReferenceInstruction

```
class InsertWithNameReferenceInstruction {
    instruction_type:string = "insert_with_name_reference";

    table_type:"static"|"dynamic";

    name_index:uint32;

    huffman_encoded_value:boolean;

    value_length?:uint32;
    value?:string;
}
```

C.1.3. InsertWithoutNameReferenceInstruction

```
class InsertWithoutNameReferenceInstruction {
    instruction_type:string = "insert_without_name_reference";

    huffman_encoded_name:boolean;

    name_length?:uint32;
    name?:string;

    huffman_encoded_value:boolean;

    value_length?:uint32;
    value?:string;
}
```

C.1.4. DuplicateInstruction

```
class DuplicateInstruction {
    instruction_type:string = "duplicate";

    index:uint32;
}
```

C.1.5. HeaderAcknowledgementInstruction

```
class HeaderAcknowledgementInstruction {
    instruction_type:string = "header_acknowledgement";

    stream_id:uint64;
}
```

C.1.6. StreamCancellationInstruction

```
class StreamCancellationInstruction {
    instruction_type:string = "stream_cancellation";

    stream_id:uint64;
}
```

C.1.7. InsertCountIncrementInstruction

```
class InsertCountIncrementInstruction {
    instruction_type:string = "insert_count_increment";

    increment:uint32;
}
```

C.2. QPACK Header compression

```
type QPackHeaderBlockRepresentation = IndexedHeaderField | LiteralHeaderFieldWith
Name | LiteralHeaderFieldWithoutName;
```

C.2.1. IndexedHeaderField

Note: also used for "indexed header field with post-base index"

```
class IndexedHeaderField {
    header_field_type:string = "indexed_header";

    table_type:"static"|"dynamic"; // MUST be "dynamic" if is_post_base is true
    index:uint32;

    is_post_base:boolean = false; // to represent the "indexed header field with
post-base index" header field type
}
```

C.2.2. LiteralHeaderFieldWithName

Note: also used for "Literal header field with post-base name reference"

```
class LiteralHeaderFieldWithName {
    header_field_type:string = "literal_with_name";

    preserve_literal:boolean; // the 3rd "N" bit
    table_type:"static"|"dynamic"; // MUST be "dynamic" if is_post_base is true
    name_index:uint32;

    huffman_encoded_value:boolean;
    value_length?:uint32;
    value?:string;

    is_post_base:boolean = false; // to represent the "Literal header field with
    post-base name reference" header field type
}
```

C.2.3. LiteralHeaderFieldWithoutName

```
class LiteralHeaderFieldWithoutName {
    header_field_type:string = "literal_without_name";

    preserve_literal:boolean; // the 3rd "N" bit

    huffman_encoded_name:boolean;
    name_length?:uint32;
    name?:string;

    huffman_encoded_value:boolean;
    value_length?:uint32;
    value?:string;
}
```

C.2.4. QPackHeaderBlockPrefix

```
class QPackHeaderBlockPrefix {
    required_insert_count:uint32;
    sign_bit:boolean;
    delta_base:uint32;
}
```

Appendix D. Change Log

D.1. Since draft-01:

Major changes:

- * Moved data_moved from http to transport. Also made the "from" and "to" fields flexible strings instead of an enum (#111,#65)

- * Moved packet_type fields to PacketHeader. Moved packet_size field out of PacketHeader to RawInfo:length (#40)
- * Made events that need to log packet_type and packet_number use a header field instead of logging these fields individually
- * Added support for logging retry, stateless reset and initial tokens (#94,#86,#117)
- * Moved separate general event categories into a single category "generic" (#47)
- * Added "transport:connection_closed" event (#43,#85,#78,#49)
- * Added version_information and alpn_information events (#85,#75,#28)
- * Added parameters_restored events to help clarify 0-RTT behaviour (#88)

Smaller changes:

- * Merged loss_timer events into one loss_timer_updated event
- * Field data types are now strongly defined (#10,#39,#36,#115)
- * Renamed qpack instruction_received and instruction_sent to instruction_created and instruction_parsed (#114)
- * Updated qpack:dynamic_table_updated.update_type. It now has the value "inserted" instead of "added" (#113)
- * Updated qpack:dynamic_table_updated. It now has an "owner" field to differentiate encoder vs decoder state (#112)
- * Removed push_allowed from http:parameters_set (#110)
- * Removed explicit trigger field indications from events, since this was moved to be a generic property of the "data" field (#80)
- * Updated transport:connection_id_updated to be more in line with other similar events. Also dropped importance from Core to Base (#45)
- * Added length property to PaddingFrame (#34)
- * Added packet_number field to transport:frames_processed (#74)

- * Added a way to generically log packet header flags (first 8 bits) to PacketHeader
- * Added additional guidance on which events to log in which situations (#53)
- * Added "simulation:scenario" event to help indicate simulation details
- * Added "packets_acked" event (#107)
- * Added "datagram_ids" to the datagram_X and packet_X events to allow tracking of coalesced QUIC packets (#91)
- * Extended connection_state_updated with more fine-grained states (#49)

D.2. Since draft-00:

- * Event and category names are now all lowercase
- * Added many new events and their definitions
- * "type" fields have been made more specific (especially important for PacketType fields, which are now called packet_type instead of type)
- * Events are given an importance indicator (issue #22)
- * Event names are more consistent and use past tense (issue #21)
- * Triggers have been redefined as properties of the "data" field and updated for most events (issue #23)

Appendix E. Design Variations

TBD

Appendix F. Acknowledgements

Thanks to Marten Seemann, Jana Iyengar, Brian Trammell, Dmitri Tikhonov, Stephen Petrides, Jari Arkko, Marcus Ihlar, Victor Vasiliev, Mirja Kuehlewind, Jeremy Laine, Kazu Yamamoto, Christian Huitema, and Lucas Pardue for their feedback and suggestions.

Author's Address

Robin Marx
Hasselt University

Email: robin.marx@uhasselt.be

QUIC
Internet-Draft
Intended status: Standards Track
Expires: November 16, 2021

R. Marx, Ed.
KU Leuven
L. Niccolini, Ed.
Facebook
M. Seemann, Ed.
Protocol Labs
May 15, 2021

Main logging schema for qlog
draft-marx-qlog-main-schema-03

Abstract

This document describes a high-level schema for a standardized logging format called qlog. This format allows easy sharing of data and the creation of reusable visualization and debugging tools. The high-level schema in this document is intended to be protocol-agnostic. Separate documents specify how the format should be used for specific protocol data. The schema is also format-agnostic, and can be represented in for example JSON, csv or protobuf.

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 November 16, 2021.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Notational Conventions	4
2. Design goals	5
3. The high level qlog schema	6
3.1. summary	7
3.2. traces	8
3.3. Individual Trace containers	9
3.3.1. configuration	10
3.3.2. vantage_point	12
3.4. Field name semantics	14
3.4.1. timestamps	15
3.4.2. category and event	17
3.4.3. data	18
3.4.4. protocol_type	18
3.4.5. triggers	19
3.4.6. group_id	19
3.4.7. common_fields	21
4. Guidelines for event definition documents	23
4.1. Event design guidelines	23
4.2. Event importance indicators	24
4.3. Custom fields	25
5. Generic events and data classes	25
5.1. Raw packet and frame information	26
5.2. Generic events	27
5.2.1. error	27
5.2.2. warning	27
5.2.3. info	27
5.2.4. debug	28
5.2.5. verbose	28
5.3. Simulation events	28
5.3.1. scenario	29
5.3.2. marker	29
6. Serializing qlog	29
6.1. qlog to JSON mapping	30
6.1.1. numbers	30
6.1.2. bytes	31
6.1.3. Summarizing table	32
6.1.4. Other JSON specifics	33
6.2. qlog to NDJSON mapping	33
6.2.1. Supporting NDJSON in tooling	35

6.3. Other optimized formatting options	35
6.3.1. Data structure optimizations	36
6.3.2. Compression	37
6.3.3. Binary formats	37
6.3.4. Overview and summary	38
6.4. Conversion between formats	39
7. Methods of access and generation	40
7.1. Set file output destination via an environment variable .	40
7.2. Access logs via a well-known endpoint	41
8. Tooling requirements	42
9. Security and privacy considerations	42
10. IANA Considerations	43
11. References	43
11.1. Normative References	43
11.2. Informative References	43
11.3. URIs	44
Appendix A. Change Log	45
A.1. Since draft-marx-qlog-main-schema-draft-02:	45
A.2. Since draft-marx-qlog-main-schema-01:	45
A.3. Since draft-marx-qlog-main-schema-00:	46
Appendix B. Design Variations	46
Appendix C. Acknowledgements	46
Authors' Addresses	46

1. Introduction

There is currently a lack of an easily usable, standardized endpoint logging format. Especially for the use case of debugging and evaluating modern Web protocols and their performance, it is often difficult to obtain structured logs that provide adequate information for tasks like problem root cause analysis.

This document aims to provide a high-level schema and harness that describes the general layout of an easily usable, shareable, aggregatable and structured logging format. This high-level schema is protocol agnostic, with logging entries for specific protocols and use cases being defined in other documents (see for example [QLOG-QUIC] for QUIC and [QLOG-H3] for HTTP/3 and QPACK-related event definitions).

The goal of this high-level schema is to provide amenities and default characteristics that each logging file should contain (or should be able to contain), such that generic and reusable toolsets can be created that can deal with logs from a variety of different protocols and use cases.

As such, this document contains concepts such as versioning, metadata inclusion, log aggregation, event grouping and log file size reduction techniques.

Feedback and discussion are welcome at <https://github.com/quiclog/internet-drafts> [1]. Readers are advised to refer to the "editor's draft" at that URL for an up-to-date version of this document.

1.1. Notational Conventions

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

While the qlog schema's are format-agnostic, for readability the qlog documents will use a JSON-inspired format ([RFC8259]) for examples and definitions.

As qlog can be serialized both textually but also in binary, we employ a custom datatype definition language, inspired loosely by the "TypeScript" language [2].

This document describes how to employ JSON and NDJSON as textual serializations for qlog in Section 6. Other documents will describe how to utilize other concrete serialization options, though tips and requirements for these are also listed in this document (Section 6).

The main general conventions in this document a reader should be aware of are:

- o `obj?` : this object is optional
- o `type1 | type2` : a union of these two types (object can be either `type1` OR `type2`)
- o `obj:type` : this object has this concrete type
- o `obj:array<type>` : this object is an array of this type
- o `class` : defines a new type
- o `//` : single-line comment

The main data types are:

- o `int8` : signed 8-bit integer
- o `int16` : signed 16-bit integer

- o int32 : signed 32-bit integer
- o int64 : signed 64-bit integer
- o uint8 : unsigned 8-bit integer
- o uint16 : unsigned 16-bit integer
- o uint32 : unsigned 32-bit integer
- o uint64 : unsigned 64-bit integer
- o float : 32-bit floating point value
- o double : 64-bit floating point value
- o byte : an individual raw byte (8-bit) value (use array<byte> or the shorthand "bytes" to specify a binary blob)
- o string : list of Unicode (typically UTF-8) encoded characters
- o boolean : boolean
- o enum: fixed list of values (Unless explicitly defined, the value of an enum entry is the string version of its name (e.g., initial = "initial"))
- o any : represents any object type. Mainly used here as a placeholder for more concrete types defined in related documents (e.g., specific event types)

All timestamps and time-related values (e.g., offsets) in qlog are logged as doubles in the millisecond resolution.

Other qlog documents can define their own data types (e.g., separately for each Packet type that a protocol supports).

2. Design goals

The main tenets for the qlog schema design are:

- o Streamable, event-based logging
- o Flexibility in the format, complexity in the tooling (e.g., few components are a MUST, tools need to deal with this)
- o Extensible and pragmatic (e.g., no complex fixed schema with extension points)

- o Aggregation and transformation friendly (e.g., the top-level element is a container for individual traces, `group_id` can be used to tag events to a particular context)
- o Metadata is stored together with event data

3. The high level qlog schema

A qlog file should be able to contain several individual traces and logs from multiple vantage points that are in some way related. To that end, the top-level element in the qlog schema defines only a small set of "header" fields and an array of component traces. For this document, the required "qlog_version" field MUST have a value of "qlog-03-WIP".

Note: there have been several previously broadly deployed qlog versions based on older drafts of this document (see draft-marx-qlog-main-schema). The old values for the "qlog_version" field were "draft-00", "draft-01" and "draft-02". When qlog was moved to the QUIC working group, we decided to increment the existing counter, rather than reverting back to -00. As such, any numbering indicating in the "qlog_version" field is explicitly not tied to a particular version of the draft documents.

As qlog can be serialized in a variety of ways, the "qlog_format" field is used to indicate which serialization option was chosen. Its value MUST either be one of the options defined in this document (e.g., Section 6) or the field must be omitted entirely, in which case it assumes the default value of "JSON".

In order to make it easier to parse and identify qlog files and their serialization format, the "qlog_version" and "qlog_format" fields and their values SHOULD be in the first 256 characters/bytes of the resulting log file.

An example of the qlog file's top-level structure is shown in Figure 1.

Definition:

```
class QlogFile {  
    qlog_version:string,  
    qlog_format?:string,  
    title?:string,  
    description?:string,  
    summary?: Summary,  
    traces: array<Trace|TraceError>  
}
```

JSON serialization:

```
{  
    "qlog_version": "draft-03-WIP",  
    "qlog_format": "JSON",  
    "title": "Name of this particular qlog file (short)",  
    "description": "Description for this group of traces (long)",  
    "summary": {  
        ...  
    },  
    "traces": [...]  
}
```

Figure 1: Top-level element

3.1. summary

In a real-life deployment with a large amount of generated logs, it can be useful to sort and filter logs based on some basic summarized or aggregated data (e.g., log length, packet loss rate, log location, presence of error events, ...). The summary field (if present) SHOULD be on top of the qlog file, as this allows for the file to be processed in a streaming fashion (i.e., the implementation could just read up to and including the summary field and then only load the full logs that are deemed interesting by the user).

As the summary field is highly deployment-specific, this document does not specify any default fields or their semantics. Some examples of potential entries are shown in Figure 2.

Definition (purely illustrative example):

```
class Summary {  
    "trace_count":uint32, // amount of traces in this file  
    "max_duration":uint64, // time duration of the longest trace in ms  
    "max_outgoing_loss_rate":float, // highest loss rate for outgoing packets over  
    // all traces  
    "total_event_count":uint64, // total number of events across all traces,  
    "error_count":uint64 // total number of error events in this trace  
}
```

JSON serialization:

```
{  
    "trace_count": 1,  
    "max_duration": 5006,  
    "max_outgoing_loss_rate": 0.013,  
    "total_event_count": 568,  
    "error_count": 2  
}
```

Figure 2: Summary example definition

3.2. traces

It is often advantageous to group several related qlog traces together in a single file. For example, we can simultaneously perform logging on the client, on the server and on a single point on their common network path. For analysis, it is useful to aggregate these three individual traces together into a single file, so it can be uniquely stored, transferred and annotated.

As such, the "traces" array contains a list of individual qlog traces. Typical qlogs will only contain a single trace in this array. These can later be combined into a single qlog file by taking the "traces" entry/entries for each qlog file individually and copying them to the "traces" array of a new, aggregated qlog file. This is typically done in a post-processing step.

The "traces" array can thus contain both normal traces (for the definition of the Trace type, see Section 3.3), but also "error" entries. These indicate that we tried to find/convert a file for inclusion in the aggregated qlog, but there was an error during the process. Rather than silently dropping the erroneous file, we can opt to explicitly include it in the qlog file as an entry in the "traces" array, as shown in Figure 3.

Definition:

```
class TraceError {  
    error_description: string, // A description of the error  
    uri?: string, // the original URI at which we attempted to find the file  
    vantage_point?: VantagePoint // see {{vantage_point}}: the vantage point we w  
ere expecting to include here  
}
```

JSON serialization:

```
{  
  "error_description": "File could not be found",  
  "uri": "/srv/traces/today/latest.qlog",  
  "vantage_point": { type: "server" }  
}
```

Figure 3: TraceError definition

Note that another way to combine events of different traces in a single qlog file is through the use of the "group_id" field, discussed in Section 3.4.6.

3.3. Individual Trace containers

The exact conceptual definition of a Trace can be fluid. For example, a trace could contain all events for a single connection, for a single endpoint, for a single measurement interval, for a single protocol, etc. As such, a Trace container contains some metadata in addition to the logged events, see Figure 4.

In the normal use case however, a trace is a log of a single data flow collected at a single location or vantage point. For example, for QUIC, a single trace only contains events for a single logical QUIC connection for either the client or the server.

The semantics and context of the trace can mainly be deduced from the entries in the "common_fields" list and "vantage_point" field.

Definition:

```
class Trace {  
    title?: string,  
    description?: string,  
    configuration?: Configuration,  
    common_fields?: CommonFields,  
    vantage_point: VantagePoint,  
    events: array<Event>  
}
```

JSON serialization:

```
{  
  "title": "Name of this particular trace (short)",  
  "description": "Description for this trace (long)",  
  "configuration": {  
    "time_offset": 150  
  },  
  "common_fields": {  
    "ODCID": "abcde1234",  
    "time_format": "absolute"  
  },  
  "vantage_point": {  
    "name": "backend-67",  
    "type": "server"  
  },  
  "events": [...]  
}
```

Figure 4: Trace container definition

3.3.1. configuration

We take into account that a qlog file is usually not used in isolation, but by means of various tools. Especially when aggregating various traces together or preparing traces for a demonstration, one might wish to persist certain tool-based settings inside the qlog file itself. For this, the configuration field is used.

The configuration field can be viewed as a generic metadata field that tools can fill with their own fields, based on per-tool logic. It is best practice for tools to prefix each added field with their tool name to prevent collisions across tools. This document only defines two optional, standard, tool-independent configuration settings: "time_offset" and "original_uris".

Definition:

```
class Configuration {  
    time_offset:double, // in ms,  
    original_uris: array<string>,  
  
    // list of fields with any type  
}
```

JSON serialization:

```
{  
  "time_offset": 150, // starts 150ms after the first timestamp indicates  
  "original_uris": [  
    "https://example.org/trace1.qlog",  
    "https://example.org/trace2.qlog"  
  ]  
}
```

Figure 5: Configuration definition

3.3.1.1. time_offset

The `time_offset` field indicates by how many milliseconds the starting time of the current trace should be offset. This is useful when comparing logs taken from various systems, where clocks might not be perfectly synchronous. Users could use manual tools or automated logic to align traces in time and the found optimal offsets can be stored in this field for future usage. The default value is 0.

3.3.1.2. original_uris

The `original_uris` field is used when merging multiple individual qlog files or other source files (e.g., when converting .pcaps to qlog). It allows to keep better track where certain data came from. It is a simple array of strings. It is an array instead of a single string, since a single qlog trace can be made up out of an aggregation of multiple component qlog traces as well. The default value is an empty array.

3.3.1.3. custom fields

Tools can add optional custom metadata to the "configuration" field to store state and make it easier to share specific data viewpoints and view configurations.

Two examples from the qvis toolset [3] are shown in Figure 6.

```

{
  "configuration" : {
    "qvis" : {
      // when loaded into the qvis toolsuite's congestion graph tool
      // zoom in on the period between 1s and 2s and select the 124th event
      defined in this trace
      "congestion_graph": {
        "startX": 1000,
        "endX": 2000,
        "focusOnEventIndex": 124
      }

      // when loaded into the qvis toolsuite's sequence diagram tool
      // automatically scroll down the timeline to the 555th event defined
      in this trace
      "sequence_diagram" : {
        "focusOnEventIndex": 555
      }
    }
  }
}

```

Figure 6: Custom configuration fields example

3.3.2. vantage_point

The `vantage_point` field describes the vantage point from which the trace originates, see Figure 7. Each trace can have only a single `vantage_point` and thus all events in a trace MUST BE from the perspective of this `vantage_point`. To include events from multiple `vantage_points`, implementers can for example include multiple traces, split by `vantage_point`, in a single qlog file.

Definition:

```
class VantagePoint {
  name?: string,
  type: VantagePointType,
  flow?: VantagePointType
}

class VantagePointType {
  server, // endpoint which initiates the connection.
  client, // endpoint which accepts the connection.
  network, // observer in between client and server.
  unknown
}
```

JSON serialization examples:

```
{
  "name": "aioquic client",
  "type": "client",
}

{
  "name": "wireshark trace",
  "type": "network",
  "flow": "client"
}
```

Figure 7: VantagePoint definition

The flow field is only required if the type is "network" (for example, the trace is generated from a packet capture). It is used to disambiguate events like "packet sent" and "packet received". This is indicated explicitly because for multiple reasons (e.g., privacy) data from which the flow direction can be otherwise inferred (e.g., IP addresses) might not be present in the logs.

Meaning of the different values for the flow field: * "client" indicates that this vantage point follows client data flow semantics (a "packet sent" event goes in the direction of the server). * "server" indicates that this vantage point follow server data flow semantics (a "packet sent" event goes in the direction of the client). * "unknown" indicates that the flow's direction is unknown.

Depending on the context, tools confronted with "unknown" values in the vantage_point can either try to heuristically infer the semantics from protocol-level domain knowledge (e.g., in QUIC, the client

always sends the first packet) or give the user the option to switch between client and server perspectives manually.

3.4. Field name semantics

Inside of the "events" field of a qlog trace is a list of events logged by the endpoint. Each event is specified as a generic object with a number of member fields and their associated data. Depending on the protocol and use case, the exact member field names and their formats can differ across implementations. This section lists the main, pre-defined and reserved field names with specific semantics and expected corresponding value formats.

Each qlog event at minimum requires the "time" (Section 3.4.1), "name" (Section 3.4.2) and "data" (Section 3.4.3) fields. Other typical fields are "time_format" (Section 3.4.1), "protocol_type" (Section 3.4.4), "trigger" (Section 3.4.5), and "group_id" (Section 3.4.6). As especially these later fields typically have identical values across individual event instances, they are normally logged separately in the "common_fields" (Section 3.4.7).

The specific values for each of these fields and their semantics are defined in separate documents, specific per protocol or use case. For example: event definitions for QUIC, HTTP/3 and QPACK can be found in [QLOG-QUIC] and [QLOG-H3].

Other fields are explicitly allowed by the qlog approach, and tools SHOULD allow for the presence of unknown event fields, but their semantics depend on the context of the log usage (e.g., for QUIC, the ODCID field is used), see [QLOG-QUIC].

An example of a qlog event with its component fields is shown in Figure 8.

Definition:

```
class Event {
  time: double,
  name: string,
  data: any,

  protocol_type?: Array<string>,
  group_id?: string|uint32,

  time_format?: "absolute"|"delta"|"relative",

  // list of fields with any type
}
```

JSON serialization:

```
{
  time: 1553986553572,

  name: "transport:packet_sent",
  data: { ... }

  protocol_type: ["QUIC","HTTP3"],
  group_id: "127ecc830d98f9d54a42c4f0842aa87e181a",

  time_format: "absolute",

  ODCID: "127ecc830d98f9d54a42c4f0842aa87e181a", // QUIC specific
}
```

Figure 8: Event fields definition

3.4.1. timestamps

The "time" field indicates the timestamp at which the event occurred. Its value is typically the Unix timestamp since the 1970 epoch (number of milliseconds since midnight UTC, January 1, 1970, ignoring leap seconds). However, qlog supports two more succinct timestamps formats to allow reducing file size. The employed format is indicated in the "time_format" field, which allows one of three values: "absolute", "delta" or "relative":

- o Absolute: Include the full absolute timestamp with each event. This approach uses the largest amount of characters. This is also the default value of the "time_format" field.

- o **Delta:** Delta-encode each time value on the previously logged value. The first event in a trace typically logs the full absolute timestamp. This approach uses the least amount of characters.
- o **Relative:** Specify a full "reference_time" timestamp (typically this is done up-front in "common_fields", see Section 3.4.7) and include only relatively-encoded values based on this reference_time with each event. The "reference_time" value is typically the first absolute timestamp. This approach uses a medium amount of characters.

The first option is good for stateless loggers, the second and third for stateful loggers. The third option is generally preferred, since it produces smaller files while being easier to reason about. An example for each option can be seen in Figure 9.

The absolute approach will use:
1500, 1505, 1522, 1588

The delta approach will use:
1500, 5, 17, 66

The relative approach will:
- set the reference_time to 1500 in "common_fields"
- use: 0, 5, 22, 88

Figure 9: Three different approaches for logging timestamps

One of these options is typically chosen for the entire trace (put differently: each event has the same value for the "time_format" field). Each event **MUST** include a timestamp in the "time" field.

Events in each individual trace **SHOULD** be logged in strictly ascending timestamp order (though not necessarily absolute value, for the "delta" format). Tools **CAN** sort all events on the timestamp before processing them, though are not required to (as this could impose a significant processing overhead). This can be a problem especially for multi-threaded and/or streaming loggers, who could consider using a separate postprocessor to order qlog events in time if a tool do not provide this feature.

Timestamps do not have to use the UNIX epoch timestamp as their reference. For example for privacy considerations, any initial reference timestamps (for example "endpoint uptime in ms" or "time since connection start in ms") can be chosen. Tools **SHOULD NOT** assume the ability to derive the absolute Unix timestamp from qlog

traces, nor allow on them to relatively order events across two or more separate traces (in this case, clock drift should also be taken into account).

3.4.2. category and event

Events differ mainly in the type of metadata associated with them. To help identify a given event and how to interpret its metadata in the "data" field (see Section 3.4.3), each event has an associated "name" field. This can be considered as a concatenation of two other fields, namely event "category" and event "type".

Category allows a higher-level grouping of events per specific event type. For example for QUIC and HTTP/3, the different categories could be "transport", "http", "qpack", and "recovery". Within these categories, the event Type provides additional granularity. For example for QUIC and HTTP/3, within the "transport" Category, there would be "packet_sent" and "packet_received" events.

Logging category and type separately conceptually allows for fast and high-level filtering based on category and the re-use of event types across categories. However, it also considerably inflates the log size and this flexibility is not used extensively in practice at the time of writing.

As such, the default approach in qlog is to concatenate both field values using the ":" character in the "name" field, as can be seen in Figure 10. As such, qlog category and type names MUST NOT include this character.

JSON serialization using separate fields:

```
{
  category: "transport",
  type: "packet_sent"
}
```

JSON serialization using ":" concatenated field:

```
{
  name: "transport:packet_sent"
}
```

Figure 10: Ways of logging category, type and name of an event.

Certain serializations CAN emit category and type as separate fields, and qlog tools SHOULD be able to deal with both the concatenated "name" field, and the separate "category" and "type" fields. Text-based serializations however are encouraged to employ the concatenated "name" field for efficiency.

3.4.3. data

The data field is a generic object. It contains the per-event metadata and its form and semantics are defined per specific sort of event. For example, data field value definitions for QUIC and HTTP/3 can be found in [QLOG-QUIC] and [QLOG-H3].

One purely illustrative example for a QUIC "packet_sent" event is shown in Figure 11.

Definition:

```
class TransportPacketSentEvent {
  packet_size?:uint32,
  header:PacketHeader,
  frames?:Array<QuicFrame>
}
```

JSON serialization:

```
{
  packet_size: 1280,
  header: {
    packet_type: "1RTT",
    packet_number: 123
  },
  frames: [
    {
      frame_type: "stream",
      length: 1000,
      offset: 456
    },
    {
      frame_type: "padding"
    }
  ]
}
```

Figure 11: Example of the 'data' field for a QUIC packet_sent event

3.4.4. protocol_type

The "protocol_type" array field indicates to which protocols (or protocol "stacks") this event belongs. This allows a single qlog file to aggregate traces of different protocols (e.g., a web server offering both TCP+HTTP/2 and QUIC+HTTP/3 connections).

For example, QUIC and HTTP/3 events have the "QUIC" and "HTTP3" protocol_type entry values, see [QLOG-QUIC] and [QLOG-H3].

Typically however, all events in a single trace are of the same few protocols, and this array field is logged once in "common_fields", see Section 3.4.7.

3.4.5. triggers

Sometimes, additional information is needed in the case where a single event can be caused by a variety of other events. In the normal case, the context of the surrounding log messages gives a hint as to which of these other events was the cause. However, in highly-parallel and optimized implementations, corresponding log messages might be separated in time. Another option is to explicitly indicate these "triggers" in a high-level way per-event to get more fine-grained information without much additional overhead.

In qlog, the optional "trigger" field contains a string value describing the reason (if any) for this event instance occurring. While this "trigger" field could be a property of the qlog Event itself, it is instead a property of the "data" field instead. This choice was made because many event types do not include a trigger value, and having the field at the Event-level would cause overhead in some serializations. Additional information on the trigger can be added in the form of additional member fields of the "data" field value, yet this is highly implementation-specific, as are the trigger field's string values.

One purely illustrative example of some potential triggers for QUIC's "packet_dropped" event is shown in Figure 12.

Definition:

```
class QuicPacketDroppedEvent {
    packet_type?:PacketType,
    raw_length?:uint32,

    trigger?: "key_unavailable" | "unknown_connection_id" | "decrypt_error" | "un
supported_version"
}
```

Figure 12: Trigger example

3.4.6. group_id

As discussed in Section 3.3, a single qlog file can contain several traces taken from different vantage points. However, a single trace from one endpoint can also contain events from a variety of sources.

For example, a server implementation might choose to log events for all incoming connections in a single large (streamed) qlog file. As such, we need a method for splitting up events belonging to separate logical entities.

The simplest way to perform this splitting is by associating a "group identifier" to each event that indicates to which conceptual "group" each event belongs. A post-processing step can then extract events per group. However, this group identifier can be highly protocol and context-specific. In the example above, we might use QUIC's "Original Destination Connection ID" to uniquely identify a connection. As such, they might add a "ODCID" field to each event. However, a middlebox logging IP or TCP traffic might rather use four-tuples to identify connections, and add a "four_tuple" field.

As such, to provide consistency and ease of tooling in cross-protocol and cross-context setups, qlog instead defines the common "group_id" field, which contains a string value. Implementations are free to use their preferred string serialization for this field, so long as it contains a unique value per logical group. Some examples can be seen in Figure 13.

JSON serialization for events grouped by four tuples and QUIC connection IDs:

```
events: [
  {
    time: 1553986553579,
    protocol_type: ["TCP", "TLS", "HTTP2"],
    group_id: "ip1=2001:67c:1232:144:9498:6df6:f450:110b,ip2=2001:67c:2b0:1c1
::198,port1=59105,port2=80",
    name: "transport:packet_received",
    data: { ... },
  },
  {
    time: 1553986553581,
    protocol_type: ["QUIC", "HTTP3"],
    group_id: "127ecc830d98f9d54a42c4f0842aa87e181a",
    name: "transport:packet_sent",
    data: { ... },
  }
]
```

Figure 13: Example of group_id usage

Note that in some contexts (for example a Multipath transport protocol) it might make sense to add additional contextual per-event fields (for example "path_id"), rather than use the group_id field for that purpose.

Note also that, typically, a single trace only contains events belonging to a single logical group (for example, an individual QUIC connection). As such, instead of logging the "group_id" field with an identical value for each event instance, this field is typically logged once in "common_fields", see Section 3.4.7.

3.4.7. common_fields

As discussed in the previous sections, information for a typical qlog event varies in three main fields: "time", "name" and associated data. Additionally, there are also several more advanced fields that allow mixing events from different protocols and contexts inside of the same trace (for example "protocol_type" and "group_id"). In most "normal" use cases however, the values of these advanced fields are consistent for each event instance (for example, a single trace contains events for a single QUIC connection).

To reduce file size and making logging easier, qlog uses the "common_fields" list to indicate those fields and their values that are shared by all events in this component trace. This prevents these fields from being logged for each individual event. An example of this is shown in Figure 14.

JSON serialization with repeated field values per-event instance:

```
{
  events: [{
    group_id: "127ecc830d98f9d54a42c4f0842aa87e181a",
    protocol_type: ["QUIC", "HTTP3"],
    time_format: "relative",
    reference_time: "1553986553572",

    time: 2,
    name: "transport:packet_received",
    data: { ... }
  }, {
    group_id: "127ecc830d98f9d54a42c4f0842aa87e181a",
    protocol_type: ["QUIC", "HTTP3"],
    time_format: "relative",
    reference_time: "1553986553572",

    time: 7,
    name: "http:frame_parsed",
    data: { ... }
  }
]
```

JSON serialization with repeated field values extracted to common_fields:

```
{
  common_fields: {
    group_id: "127ecc830d98f9d54a42c4f0842aa87e181a",
    protocol_type: ["QUIC", "HTTP3"],
    time_format: "relative",
    reference_time: "1553986553572"
  },
  events: [
    {
      time: 2,
      name: "transport:packet_received",
      data: { ... }
    }, {
      7,
      name: "http:frame_parsed",
      data: { ... }
    }
  ]
}
```

Figure 14: Example of common_fields usage

The "common_fields" field is a generic dictionary of key-value pairs, where the key is always a string and the value can be of any type, but is typically also a string or number. As such, unknown entries in this dictionary MUST be disregarded by the user and tools (i.e., the presence of an unknown field is explicitly NOT an error).

The list of default qlog fields that are typically logged in common_fields (as opposed to as individual fields per event instance) are:

- o time_format
- o reference_time
- o protocol_type
- o group_id

Tools MUST be able to deal with these fields being defined either on each event individually or combined in common_fields. Note that if at least one event in a trace has a different value for a given field, this field MUST NOT be added to common_fields but instead defined on each event individually. Good example of such fields are "time" and "data", who are divergent by nature.

4. Guidelines for event definition documents

This document only defines the main schema for the qlog format. This is intended to be used together with specific, per-protocol event definitions that specify the name (category + type) and data needed for each individual event. This is with the intent to allow the qlog main schema to be easily re-used for several protocols. Examples include the QUIC event definitions [QLOG-QUIC] and HTTP/3 and QPACK event definitions [QLOG-H3].

This section defines some basic annotations and concepts the creators of event definition documents SHOULD follow to ensure a measure of consistency, making it easier for qlog implementers to extrapolate from one protocol to another.

4.1. Event design guidelines

TODO: pending QUIC working group discussion. This text reflects the initial (qlog draft 01 and 02) setup.

There are several ways of defining qlog events. In practice, we have seen two main types used so far: a) those that map directly to concepts seen in the protocols (e.g., "packet_sent") and b) those

that act as aggregating events that combine data from several possible protocol behaviours or code paths into one (e.g., "parameters_set"). The latter are typically used as a means to reduce the amount of unique event definitions, as reflecting each possible protocol event as a separate qlog entity would cause an explosion of event types.

Additionally, logging duplicate data is typically prevented as much as possible. For example, packet header values that remain consistent across many packets are split into separate events (for example "spin_bit_updated" or "connection_id_updated" for QUIC).

Finally, we have typically refrained from adding additional state change events if those state changes can be directly inferred from data on the wire (for example flow control limit changes) if the implementation is bug-free and spec-compliant. Exceptions have been made for common events that benefit from being easily identifiable or individually logged (for example "packets_acked").

4.2. Event importance indicators

Depending on how events are designed, it may be that several events allow the logging of similar or overlapping data. For example the separate QUIC "connection_started" event overlaps with the more generic "connection_state_updated". In these cases, it is not always clear which event should be logged or used, and which event should take precedence if e.g., both are present and provide conflicting information.

To aid in this decision making, we recommend that each event SHOULD have an "importance indicator" with one of three values, in decreasing order of importance and expected usage:

- o Core
- o Base
- o Extra

The "Core" events are the events that SHOULD be present in all qlog files for a given protocol. These are typically tied to basic packet and frame parsing and creation, as well as listing basic internal metrics. Tool implementers SHOULD expect and add support for these events, though SHOULD NOT expect all Core events to be present in each qlog trace.

The "Base" events add additional debugging options and CAN be present in qlog files. Most of these can be implicitly inferred from data in

Core events (if those contain all their properties), but for many it is better to log the events explicitly as well, making it clearer how the implementation behaves. These events are for example tied to passing data around in buffers, to how internal state machines change and help show when decisions are actually made based on received data. Tool implementers SHOULD at least add support for showing the contents of these events, if they do not handle them explicitly.

The "Extra" events are considered mostly useful for low-level debugging of the implementation, rather than the protocol. They allow more fine-grained tracking of internal behaviour. As such, they CAN be present in qlog files and tool implementers CAN add support for these, but they are not required to.

Note that in some cases, implementers might not want to log for example data content details in the "Core" events due to performance or privacy considerations. In this case, they SHOULD use (a subset of) relevant "Base" events instead to ensure usability of the qlog output. As an example, implementations that do not log QUIC "packet_received" events and thus also not which (if any) ACK frames the packet contains, SHOULD log "packets_acked" events instead.

Finally, for event types whose data (partially) overlap with other event types' definitions, where necessary the event definition document should include explicit guidance on which to use in specific situations.

4.3. Custom fields

Event definition documents are free to define new category and event types, top-level fields (e.g., a per-event field indicating its privacy properties or path_id in multipath protocols), as well as values for the "trigger" property within the "data" field, or other member fields of the "data" field, as they see fit.

They however SHOULD NOT expect non-specialized tools to recognize or visualize this custom data. However, tools SHOULD make an effort to visualize even unknown data if possible in the specific tool's context. If they do not, they MUST ignore these unknown fields.

5. Generic events and data classes

There are some event types and data classes that are common across protocols, applications and use cases that benefit from being defined in a single location. This section specifies such common definitions.

5.1. Raw packet and frame information

While qlog is a more high-level logging format, it also allows the inclusion of most raw wire image information, such as byte lengths and even raw byte values. This can be useful when for example investigating or tuning packetization behaviour or determining encoding/framing overheads. However, these fields are not always necessary and can take up considerable space if logged for each packet or frame. They can also have a considerable privacy and security impact. As such, they are grouped in a separate optional field called "raw" of type RawInfo (where applicable).

```
class RawInfo {  
    length?:uint64; // the full byte length of the entity (e.g., packet or frame)  
    including headers and trailers  
    payload_length?:uint64; // the byte length of the entity's payload, without h  
eaders or trailers  
  
    data?:bytes; // the contents of the full entity, including headers and traile  
rs  
}
```

Note: The RawInfo:data field can be truncated for privacy or security purposes (for example excluding payload data). In this case, the length properties should still indicate the non-truncated lengths.

Note: We do not specify explicit header_length or trailer_length fields. In most protocols, header_length can be calculated by subtracing the payload_length from the length (e.g., if trailer_length is always 0). In protocols with trailers (e.g., QUIC's AEAD tag), event definitions documents SHOULD define other ways of logging the trailer_length to make the header_length calculation possible.

The exact definitions entities, headers, trailers and payloads depend on the protocol used. If this is non-trivial, event definitions documents SHOULD include a clear explanation of how entities are mapped into the RawInfo structure.

Note: Relatedly, many modern protocols use Variable-Length Integer Encoded (VLIE) values in their headers, which are of a dynamic length. Because of this, we cannot deterministically reconstruct the header encoding/length from non-RawInfo qlog data, as implementations might not necessarily employ the most efficient VLIE scheme for all values. As such, to make exact size-analysis possible, implementers should use explicit lengths in RawInfo rather than reconstructing them from other qlog data. Similarly, tool developers should only utilize RawInfo (and related information) in such tools to prevent errors.

5.2. Generic events

In typical logging setups, users utilize a discrete number of well-defined logging categories, levels or severities to log freeform (string) data. This generic events category replicates this approach to allow implementations to fully replace their existing text-based logging by qlog. This is done by providing events to log generic strings for the typical well-known logging levels (error, warning, info, debug, verbose).

For the events defined below, the "category" is "generic" and their "type" is the name of the heading in lowercase (e.g., the "name" of the error event is "generic:error").

5.2.1. error

Importance: Core

Used to log details of an internal error that might not get reflected on the wire.

Data:

```
{
  code?:uint32,
  message?:string
}
```

5.2.2. warning

Importance: Base

Used to log details of an internal warning that might not get reflected on the wire.

Data:

```
{
  code?:uint32,
  message?:string
}
```

5.2.3. info

Importance: Extra

Used mainly for implementations that want to use qlog as their one and only logging format but still want to support unstructured string messages.

Data:

```
{  
    message:string  
}
```

5.2.4. debug

Importance: Extra

Used mainly for implementations that want to use qlog as their one and only logging format but still want to support unstructured string messages.

Data:

```
{  
    message:string  
}
```

5.2.5. verbose

Importance: Extra

Used mainly for implementations that want to use qlog as their one and only logging format but still want to support unstructured string messages.

Data:

```
{  
    message:string  
}
```

5.3. Simulation events

When evaluating a protocol implementation, one typically sets up a series of interoperability or benchmarking tests, in which the test situations can change over time. For example, the network bandwidth or latency can vary during the test, or the network can be fully disable for a short time. In these setups, it is useful to know when exactly these conditions are triggered, to allow for proper correlation with other events.

For the events defined below, the "category" is "simulation" and their "type" is the name of the heading in lowercase (e.g., the "name" of the scenario event is "simulation:scenario").

5.3.1. scenario

Importance: Extra

Used to specify which specific scenario is being tested at this particular instance. This could also be reflected in the top-level qlog's "summary" or "configuration" fields, but having a separate event allows easier aggregation of several simulations into one trace (e.g., split by "group_id").

```
{
  name?:string,
  details?:any
}
```

5.3.2. marker

Importance: Extra

Used to indicate when specific emulation conditions are triggered at set times (e.g., at 3 seconds in 2% packet loss is introduced, at 10s a NAT rebind is triggered).

```
{
  type?:string,
  message?:string
}
```

6. Serializing qlog

This document and other related qlog schema definitions are intentionally serialization-format agnostic. This means that implementers themselves can choose how to represent and serialize qlog data practically on disk or on the wire. Some examples of possible formats are JSON, CBOR, CSV, protocol buffers, flatbuffers, etc.

All these formats make certain tradeoffs between flexibility and efficiency, with textual formats like JSON typically being more flexible but also less efficient than binary formats like protocol buffers. The format choice will depend on the practical use case of the qlog user. For example, for use in day to day debugging, a plaintext readable (yet relatively large) format like JSON is probably preferred. However, for use in production, a more optimized

yet restricted format can be better. In this latter case, it will be more difficult to achieve interoperability between qlog implementations of various protocol stacks, as some custom or tweaked events from one might not be compatible with the format of the other. This will also reflect in tooling: not all tools will support all formats.

This being said, the authors prefer JSON as the basis for storing qlog, as it retains full flexibility and maximum interoperability. Storage overhead can be managed well in practice by employing compression. For this reason, this document details both how to practically transform qlog schema definitions to JSON and to the streamable NDJSON. We discuss concrete options to bring down JSON size and processing overheads in Section 6.3.

As depending on the employed format different deserializers/parsers should be used, the "qlog_format" field is used to indicate the chosen serialization approach. This field is always a string, but can be made hierarchical by the use of the "." separator between entries. For example, a value of "JSON.optimizationA" can indicate that a default JSON format is being used, but that a certain optimization of type A was applied to the file as well (see also Section 6.3).

6.1. qlog to JSON mapping

When mapping qlog to normal JSON, the "qlog_format" field MUST have the value "JSON". This is also the default qlog serialization and default value of this field.

To facilitate this mapping, the qlog documents employ a format that is close to pure JSON for its examples and data definitions. Still, as JSON is not a typed format, there are some practical peculiarities to observe.

6.1.1. numbers

While JSON has built-in support for integers up to 64 bits in size, not all JSON parsers do. For example, none of the major Web browsers support full 64-bit integers at this time, as all numerical values (both floating-point numbers and integers) are internally represented as floating point IEEE 754 [4] values. In practice, this limits their integers to a maximum value of $2^{53}-1$. Integers larger than that are either truncated or produce a JSON parsing error. While this is expected to improve in the future (as "BigInt" support [5] has been introduced in most Browsers, though not yet integrated into JSON parsers), we still need to deal with it here.

When transforming an `int64`, `uint64` or `double` from qlog to JSON, the implementer can thus choose to either log them as JSON numbers (taking the risk of truncation or un-parseability) or to log them as strings instead. Logging as strings should however only be practically needed if the value is likely to exceed $2^{53}-1$. In practice, even though protocols such as QUIC allow 64-bit values for for example stream identifiers, these high numbers are unlikely to be reached for the overwhelming majority of cases. As such, it is probably a valid trade-off to take the risk and log 64-bit values as JSON numbers instead of strings.

Tools processing JSON-based qlog SHOULD however be able to deal with 64-bit fields being serialized as either strings or numbers.

6.1.2. bytes

Unlike most binary formats, JSON does not allow the logging of raw binary blobs directly. As such, when serializing a byte or `array<byte>`, a scheme needs to be chosen.

To represent qlog bytes in JSON, they MUST be serialized to their lowercase hexadecimal equivalents (with 0 prefix for values lower than 10). All values are directly appended to each other, without delimiters. The full value is not prefixed with 0x (as is sometimes common). An example is given in Figure 15.

For the five raw unsigned byte input values of: 5 20 40 171 255, the JSON serialization is:

```
{
  raw: "051428abff"
}
```

Figure 15: Example for serializing bytes

As such, the resulting string will always have an even amount of characters and the original byte-size can be retrieved by dividing the string length by 2.

6.1.2.1. Truncated values

In some cases, it can be interesting not to log a full raw blob but instead a truncated value (for example, only the first 100 bytes of an HTTP response body to be able to discern which file it actually contained). In these cases, the original byte-size length cannot be obtained from the serialized value directly. As such, all qlog schema definitions SHOULD include a separate, length-indicating field for all fields of type `array<byte>` they specify. This allows always retrieving the original length, but also allows the omission of any

raw value bytes of the field completely (e.g., out of privacy or security considerations).

To reduce overhead however and in the case the full raw value is logged, the extra length-indicating field can be left out. As such, tools MUST be able to deal with this situation and derive the length of the field from the raw value if no separate length-indicating field is present. All possible permutations are shown by example in Figure 16.

```
// both the full raw value and its length are present (length is redundant)
{
  "raw_length": 5,
  "raw": "051428abff"
}

// only the raw value is present, indicating it represents the fields full value
// the byte length is obtained by calculating raw.length / 2
{
  "raw": "051428abff"
}

// only the length field is present, meaning the value was omitted
{
  "raw_length": 5,
}

// both fields are present and the lengths do not match: the value was truncated
// to the first three bytes.
{
  "raw_length": 5,
  "raw": "051428"
}
```

Figure 16: Example for serializing truncated bytes

6.1.3. Summarizing table

By definition, JSON strings are serialized surrounded by quotes. Numbers without.

qlog type	JSON type
int8	number
int16	number
int32	number
uint8	number
uint16	number
uint32	number
float	number
int64	number or string
uint64	number or string
double	number or string
bytes	string (lowercase hex value)
string	string
boolean	string ("true" or "false")
enum	string (full value/name, not index)
any	object ({ ... })
array	array ([...])

6.1.4. Other JSON specifics

JSON files by definition ([RFC8259]) MUST utilize the UTF-8 encoding, both for the file itself and the string values.

Most JSON parsers strictly follow the JSON specification. This includes the rule that trailing comma's are not allowed. As it is frequently annoying to remove these trailing comma's when logging events in a streaming fashion, tool implementers SHOULD allow the last event entry of a qlog trace to be an empty object. This allows loggers to simply close the qlog file by appending "{}]]]]" after their last added event.

Finally, while not specifically required by the JSON specification, all qlog field names in a JSON serialization MUST be lowercase.

6.2. qlog to NDJSON mapping

One of the downsides of using pure JSON is that it is inherently a non-streamable format. Put differently, it is not possible to simply append new qlog events to a log file without "closing" this file at the end by appending "]]]]". Without these closing tags, most JSON parsers will be unable to parse the file entirely. As most platforms do not provide a standard streaming JSON parser (which would be able to deal with this problem), this document also provides a qlog mapping to a streamable JSON format called Newline-Delimited JSON (NDJSON) [6].

When mapping qlog to NDJSON, the "qlog_format" field MUST have the value "NDJSON".

NDJSON is very similar to JSON, except that it interprets each line in a file as a fully separate JSON object. Put differently, unlike default JSON, it does not require a file to be wrapped as a full object with "{ ... }" or "[...]". Using this setup, qlog events can simply be appended as individually serialized lines at the back of a streamed logging file.

For this to work, some qlog definitions have to be adjusted however. Mainly, events are no longer part of the "events" array in the Trace object, but are instead logged separately from the qlog "file header" (QlogFile class in Section 3). Additionally, qlog's NDJSON mapping does not allow logging multiple individual traces in a single qlog file. As such, the QlogFile:traces field is replaced by the singular "trace" field, which simply contains the Trace data directly. An example can be seen in Figure 17. Note that the "group_id" field can still be used on a per-event basis to include events from conceptually different sources in a single NDJSON qlog file.

Note as well from Figure 17 that the file's header (QlogFileNDJSON) also needs to be fully serialized on a single line to be NDJSON compatible.

Definition:

```
class QlogFileNDJSON {
    qlog_format: "NDJSON",

    qlog_version:string,
    title?:string,
    description?:string,
    summary?: Summary,
    trace: Trace
}
// list of qlog events, separated by newlines
```

NDJSON serialization:

```
{"qlog_format":"NDJSON","qlog_version":"draft-03-WIP","title":"Name of this parti
cular NDJSON qlog file (short)","description":"Description for this NDJSON qlog f
ile (long)","trace":{"common_fields":{"protocol_type": ["QUIC","HTTP3"],"group_id
":"127ecc830d98f9d54a42c4f0842aa87e181a","time_format":"relative","reference_time
":"1553986553572"},"vantage_point":{"name":"backend-67","type":"server"}}}
{"time": 2, "name": "transport:packet_received", "data": { ... } }
{"time": 7, "name": "http:frame_parsed", "data": { ... } }
```

Figure 17: Top-level element

Finally, while not specifically required by the NDJSON specification, all qlog field names in a NDJSON serialization MUST be lowercase.

6.2.1. Supporting NDJSON in tooling

Note that NDJSON is not supported in most default programming environments (unlike normal JSON). However, several custom NDJSON parsing libraries exist [7] that can be used and the format is easy enough to parse with existing implementations (i.e., by splitting the file into its component lines and feeding them to a normal JSON parser individually, as each line by itself is a valid JSON object).

6.3. Other optimized formatting options

Both the JSON and NDJSON formatting options described above are serviceable in general small to medium scale (debugging) setups. However, these approaches tend to be relatively verbose, leading to larger file sizes. Additionally, generalized (ND)JSON (de)serialization performance is typically (slightly) lower than that of more optimized and predictable formats. Both aspects make these formats more challenging (though still practical [8]) to use in large scale setups.

During the development of qlog, we compared a multitude of alternative formatting and optimization options. The results of this study are summarized on the qlog github repository [9]. The rest of this section discusses some of these approaches implementations could choose and the expected gains and tradeoffs inherent therein. Tools SHOULD support mainly the compression options listed in Section 6.3.2, as they provide the largest wins for the least cost overall.

Over time, specific qlog formats and encodings can be created that more formally define and combine some of the discussed optimizations or add new ones. We choose to define these schemes in separate documents to keep the main qlog definition clean and generalizable, as not all contexts require the same performance or flexibility as others and qlog is intended to be a broadly usable and extensible format (for example more flexibility is needed in earlier stages of protocol development, while more performance is typically needed in later stages). This is also the main reason why the general qlog format is the less optimized JSON instead of a more performant option.

To be able to easily distinguish between these options in qlog compatible tooling (without the need to have the user provide out-of-band information or to (heuristically) parse and process files in a multitude of ways, see also Section 8), we recommend using explicit file extensions to indicate specific formats. As there are no standards in place for this type of extension to format mapping, we employ a commonly used scheme here. Our approach is to list the

applied optimizations in the extension in ascending order of application (e.g., if a qlog file is first optimized with technique A and then compressed with technique B, the resulting file would have the extension ".qlog.A.B"). This allows tooling to start at the back of the extension to "undo" applied optimizations to finally arrive at the expected qlog representation.

6.3.1. Data structure optimizations

The first general category of optimizations is to alter the representation of data within an (ND)JSON qlog file to reduce file size.

The first option is to employ a scheme similar to the CSV (comma separated value [rfc4180]) format, which utilizes the concept of column "headers" to prevent repeating field names for each datapoint instance. Concretely for JSON qlog, several field names are repeated with each event (i.e., time, name, data). These names could be extracted into a separate list, after which qlog events could be serialized as an array of values, as opposed to a full object. This approach was a key part of the original qlog format (prior to draft 02) using the "event_fields" field. However, tests showed that this optimization only provided a mean file size reduction of 5% (100MB to 95MB) while significantly increasing the implementation complexity, and this approach was abandoned in favor of the default JSON setup. Implementations using this format should not employ a separate file extension (as it still uses JSON), but rather employ a new value of "JSON.namedheaders" (or "NDJSON.namedheaders") for the "qlog_format" field (see Section 3).

The second option is to replace field values and/or names with indices into a (dynamic) lookup table. This is a common compression technique and can provide significant file size reductions (up to 50% in our tests, 100MB to 50MB). However, this approach is even more difficult to implement efficiently and requires either including the (dynamic) table in the resulting file (an approach taken by for example Chromium's NetLog format [10]) or defining a (static) table up-front and sharing this between implementations. Implementations using this approach should not employ a separate file extension (as it still uses JSON), but rather employ a new value of "JSON.dictionary" (or "NDJSON.dictionary") for the "qlog_format" field (see Section 3).

As both options either proved difficult to implement, reduced qlog file readability, and provided too little improvement compared to other more straightforward options (for example Section 6.3.2), these schemes are not inherently part of qlog.

6.3.2. Compression

The second general category of optimizations is to utilize a (generic) compression scheme for textual data. As qlog in the (ND)JSON format typically contains a large amount of repetition, off-the-shelf (text) compression techniques typically succeed very well in bringing down file sizes (regularly with up to two orders of magnitude in our tests, even for "fast" compression levels). As such, utilizing compression is recommended before attempting other optimization options, even though this might (somewhat) increase processing costs due to the additional compression step.

The first option is to use GZIP compression ([RFC1952]). This generic compression scheme provides multiple compression levels (providing a trade-off between compression speed and size reduction). Utilized at level 6 (a medium setting thought to be applicable for streaming compression of a qlog stream in commodity devices), gzip compresses qlog JSON files to 7% of their initial size on average (100MB to 7MB). For this option, the file extension .qlog.gz SHOULD BE used. The "qlog_format" field should still reflect the original JSON formatting of the qlog data (e.g., "JSON" or "NDJSON").

The second option is to use Brotli compression ([RFC7932]). While similar to gzip, this more recent compression scheme provides a better efficiency. It also allows multiple compression levels. Utilized at level 4 (a medium setting thought to be applicable for streaming compression of a qlog stream in commodity devices), brotli compresses qlog JSON files to 7% of their initial size on average (100MB to 7MB). For this option, the file extension .qlog.br SHOULD BE used. The "qlog_format" field should still reflect the original JSON formatting of the qlog data (e.g., "JSON" or "NDJSON").

Other compression algorithms of course exist (for example xz, zstd, and lz4). We mainly recommend gzip and brotli because of their tweakable behaviour and wide support in web-based environments, which we envision as the main tooling ecosystem (see also Section 8).

6.3.3. Binary formats

The third general category of optimizations is to use a more optimized (often binary) format instead of the textual JSON format. This approach inherently produces smaller files and often has better (de)serialization performance. However, the resultant files are no longer human readable and some formats require hard tradeoffs between flexibility for performance.

The first option is to use the CBOR (Concise Binary Object Representation [rfc7049]) format. For our purposes, CBOR can be

viewed as a straightforward binary variant of JSON. As such, existing JSON qlog files can be trivially converted to and from CBOR (though slightly more work is needed for NDJSON qlogs). While CBOR thus does retain the full qlog flexibility, it only provides a 25% file size reduction (100MB to 75MB) compared to textual (ND)JSON. As CBOR support in programming environments is not as widespread as that of textual JSON and the format lacks human readability, CBOR was not chosen as the default qlog format. For this option, the file extension `.qlog.cbor` SHOULD BE used. The `"qlog_format"` field should still reflect the original JSON formatting of the qlog data (e.g., `"JSON"` or `"NDJSON"`).

A second option is to use a more specialized binary format, such as Protocol Buffers [11] (protobuf). This format is battle-tested, has support for optional fields and has libraries in most programming languages. Still, it is significantly less flexible than textual JSON or CBOR, as it relies on a separate, pre-defined schema (a `.proto` file). As such, it is not possible to (easily) log new event types in protobuf files without adjusting this schema as well, which has its own practical challenges. As qlog is intended to be a flexible, general purpose format, this type of format was not chosen as its basic serialization. The lower flexibility does lead to significantly reduced file sizes. Our straightforward mapping of the qlog main schema and QUIC/HTTP3 event types to protobuf created qlog files 24% as large as the raw JSON equivalents (100MB to 24MB). For this option, the file extension `.qlog.protobuf` SHOULD BE used. The `"qlog_format"` field should reflect the different internal format, for example: `"qlog_format": "protobuf"`.

Note that binary formats can (and should) also be used in conjunction with compression (see Section 6.3.2). For example, CBOR compresses well (to about 6% of the original textual JSON size (100MB to 6MB) for both gzip and brotli) and so does protobuf (5% (gzip) to 3% (brotli)). However, these gains are similar to the ones achieved by simply compressing the textual JSON equivalents directly (7%, see Section 6.3.2). As such, since compression is still needed to achieve optimal file size reductions even with binary formats, we feel the more flexible compressed textual JSON options are a better default for the qlog format in general.

6.3.4. Overview and summary

In summary, textual JSON was chosen as the main qlog format due to its high flexibility and because its inefficiencies can be largely solved by the utilization of compression techniques (which are needed to achieve optimal results with other formats as well).

Still, qlog implementers are free to define other qlog formats depending on their needs and context of use. These formats should be described in their own documents, the discussion in this document mainly acting as inspiration and high-level guidance. Implementers are encouraged to add concrete qlog formats and definitions to the designated public repository [12].

The following table provides an overview of all the discussed qlog formatting options with examples:

format	qlog_format	extension
JSON Section 6.1	JSON	.qlog
NDJSON Section 6.2	NDJSON	.qlog
named headers Section 6.3.1	(ND)JSON.namedheaders	.qlog
dictionary Section 6.3.1	(ND)JSON.dictionary	.qlog
CBOR Section 6.3.3	(ND)JSON	.qlog.cbor
protobuf Section 6.3.3	protobuf	.qlog.protobuf
gzip Section 6.3.2	no change	.gz suffix
brrotli Section 6.3.2	no change	.br suffix

6.4. Conversion between formats

As discussed in the previous sections, a qlog file can be serialized in a multitude of formats, each of which can conceivably be transformed into or from one another without loss of information. For example, a number of NDJSON streamed qlogs could be combined into a JSON formatted qlog for later processing. Similarly, a captured binary qlog could be transformed to JSON for easier interpretation and sharing.

Secondly, we can also consider other structured logging approaches that contain similar (though typically not identical) data to qlog, like raw packet capture files (for example .pcap files from tcpdump) or endpoint-specific logging formats (for example the NetLog format in Google Chrome). These are sometimes the only options, if an implementation cannot or will not support direct qlog output for any reason, but does provide other internal or external (e.g., SSLKEYLOGFILE export to allow decryption of packet captures) logging options. For this second category, a (partial) transformation from/to qlog can also be defined.

As such, when defining a new qlog serialization format or wanting to utilize qlog-compatible tools with existing codebases lacking qlog

support, it is recommended to define and provide a concrete mapping from one format to default JSON-serialized qlog. Several of such mappings exist. Firstly, [pcap2qlog] (<https://github.com/quiclog/pcap2qlog>) transforms QUIC and HTTP/3 packet capture files to qlog. Secondly, netlog2qlog [13] converts chromium's internal dictionary-encoded JSON format to qlog. Finally, quictrace2qlog [14] converts the older quictrace format to JSON qlog. Tools can then easily integrate with these converters (either by incorporating them directly or for example using them as a (web-based) API) so users can provide different file types with ease. For example, the qvis [15] toolsuite supports a multitude of formats and qlog serializations.

7. Methods of access and generation

Different implementations will have different ways of generating and storing qlogs. However, there is still value in defining a few default ways in which to steer this generation and access of the results.

7.1. Set file output destination via an environment variable

To provide users control over where and how qlog files are created, we define two environment variables. The first, QLOGFILE, indicates a full path to where an individual qlog file should be stored. This path MUST include the full file extension. The second, QLOGDIR, sets a general directory path in which qlog files should be placed. This path MUST include the directory separator character at the end.

In general, QLOGDIR should be preferred over QLOGFILE if an endpoint is prone to generate multiple qlog files. This can for example be the case for a QUIC server implementation that logs each QUIC connection in a separate qlog file. An alternative that uses QLOGFILE would be a QUIC server that logs all connections in a single file and uses the "group_id" field (Section 3.4.6) to allow post-hoc separation of events.

Implementations SHOULD provide support for QLOGDIR and MAY provide support for QLOGFILE.

When using QLOGDIR, it is up to the implementation to choose an appropriate naming scheme for the qlog files themselves. The chosen scheme will typically depend on the context or protocols used. For example, for QUIC, it is recommended to use the Original Destination Connection ID (ODCID), followed by the vantage point type of the logging endpoint. Examples of all options for QUIC are shown in Figure 18.

Command: `QLOGFILE=/srv/qlogs/client.qlog quicclientbinary`

Should result in the the `quicclientbinary` executable logging a single qlog file named `client.qlog` in the `/srv/qlogs` directory.

This is for example useful in tests when the client sets up just a single connection and then exits.

Command: `QLOGDIR=/srv/qlogs/ quicserverbinary`

Should result in the `quicserverbinary` executable generating several logs files, one for each QUIC connection.

Given two QUIC connections, with ODCID values `"abcde"` and `"12345"` respectively, this would result in two files:

`/srv/qlogs/abcde_server.qlog`

`/srv/qlogs/12345_server.qlog`

Command: `QLOGFILE=/srv/qlogs/server.qlog quicserverbinary`

Should result in the the `quicserverbinary` executable logging a single qlog file named `server.qlog` in the `/srv/qlogs` directory.

Given that the server handled two QUIC connections before it was shut down, with ODCID values `"abcde"` and `"12345"` respectively, this would result in event instances in the qlog file being tagged with the `"group_id"` field with values `"abcde"` and `"12345"`.

Figure 18: Environment variable examples for a QUIC implementation

7.2. Access logs via a well-known endpoint

After generation, qlog implementers MAY make available generated logs and traces on an endpoint (typically the server) via the following .well-known URI:

`.well-known/qlog/IDENTIFIER.extension`

The `IDENTIFIER` variable depends on the context and the protocol. For example for QUIC, the lowercase Original Destination Connection ID (ODCID) is recommended, as it can uniquely identify a connection. Additionally, the extension depends on the chosen format (see Section 6.3.4). For example, for a QUIC connection with ODCID `"abcde"`, the endpoint for fetching its default JSON-formatted .qlog file would be:

`.well-known/qlog/abcde.qlog`

Implementers SHOULD allow users to fetch logs for a given connection on a 2nd, separate connection. This helps prevent pollution of the logs by fetching them over the same connection that one wishes to observe through the log. Ideally, for the QUIC use case, the logs should also be approachable via an HTTP/2 or HTTP/1.1 endpoint (i.e., on TCP port 443), to for example aid debugging in the case where QUIC/UDP is blocked on the network.

qlog implementers SHOULD NOT enable this .well-known endpoint in typical production settings to prevent (malicious) users from

downloading logs from other connections. Implementers are advised to disable this endpoint by default and require specific actions from the end users to enable it (and potentially qlog itself). Implementers MUST also take into account the general privacy and security guidelines discussed in Section 9 before exposing qlogs to outside actors.

8. Tooling requirements

Tools ingestion qlog MUST indicate which qlog version(s), qlog format(s), compression methods and potentially other input file formats (for example .pcap) they support. Tools SHOULD at least support .qlog files in the default JSON format (Section 6.1). Additionally, they SHOULD indicate exactly which values for and properties of the name (category and type) and data fields they look for to execute their logic. Tools SHOULD perform a (high-level) check if an input qlog file adheres to the expected qlog schema. If a tool determines a qlog file does not contain enough supported information to correctly execute the tool's logic, it SHOULD generate a clear error message to this effect.

Tools MUST NOT produce breaking errors for any field names and/or values in the qlog format that they do not recognize. Tools SHOULD indicate even unknown event occurrences within their context (e.g., marking unknown events on a timeline for manual interpretation by the user).

Tool authors should be aware that, depending on the logging implementation, some events will not always be present in all traces. For example, using a circular logging buffer of a fixed size, it could be that the earliest events (e.g., connection setup events) are later overwritten by "newer" events. Alternatively, some events can be intentionally omitted out of privacy or file size considerations. Tool authors are encouraged to make their tools robust enough to still provide adequate output for incomplete logs.

9. Security and privacy considerations

TODO : discuss privacy and security considerations (e.g., what NOT to log, what to strip out of a log before sharing, ...)

TODO: strip out/don't log IPs, ports, specific CIDs, raw user data, exact times, HTTP HEADERS (or at least :path), SNI values

TODO: see if there is merit in encrypting the logs and having the server choose an encryption key (e.g., sent in transport parameters)

Good initial reference: Christian Huitema's blogpost [16]

10. IANA Considerations

TODO: primarily the .well-known URI

11. References

11.1. Normative References

[QLOG-H3] Marx, R., Ed., Niccolini, L., Ed., and M. Seemann, Ed., "HTTP/3 and QPACK event definitions for qlog", draft-marx-qlog-h3-events-00 (work in progress).

[QLOG-QUIC] Marx, R., Ed., Niccolini, L., Ed., and M. Seemann, Ed., "QUIC event definitions for qlog", draft-marx-qlog-quic-events-00 (work in progress).

11.2. Informative References

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, DOI 10.17487/RFC1952, May 1996, <<https://www.rfc-editor.org/info/rfc1952>>.

[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>>.

[rfc4180] Shafranovich, Y., "Common Format and MIME Type for Comma-Separated Values (CSV) Files", RFC 4180, DOI 10.17487/RFC4180, October 2005, <<https://www.rfc-editor.org/info/rfc4180>>.

[rfc7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

[RFC7932] Alakuijala, J. and Z. Szabadka, "Brotli Compressed Data Format", RFC 7932, DOI 10.17487/RFC7932, July 2016, <<https://www.rfc-editor.org/info/rfc7932>>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

11.3. URIs

- [1] <https://github.com/quiclog/internet-drafts>
- [2] <https://www.typescriptlang.org/>
- [3] <https://qvis.edm.uhasselt.be>
- [4] https://en.wikipedia.org/wiki/Floating-point_arithmetic
- [5] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt
- [6] <http://ndjson.org/>
- [7] <http://ndjson.org/libraries.html>
- [8] <https://qlog.edm.uhasselt.be/anrw/>
- [9] <https://github.com/quiclog/internet-drafts/issues/30#issuecomment-617675097>
- [10] <https://www.chromium.org/developers/design-documents/network-stack/netlog>
- [11] <https://developers.google.com/protocol-buffers>
- [12] <https://github.com/quiclog/qlog>
- [13] <https://github.com/quiclog/qvis/tree/master/visualizations/src/components/filemanager/netlogconverter>
- [14] <https://github.com/quiclog/quictrace2qlog>
- [15] <https://qvis.edm.uhasselt.be>
- [16] <https://huitema.wordpress.com/2020/07/21/scrubbing-quic-logs-for-privacy/>
- [17] <https://github.com/google/quic-trace>
- [18] <https://github.com/EricssonResearch/spindump>
- [19] <https://www.wireshark.org/>

Appendix A. Change Log

A.1. Since draft-marx-qlog-main-schema-draft-02:

- o These changes were done in preparation of the adoption of the drafts by the QUIC working group (#137)
- o Moved RawInfo, Importance, Generic events and Simulation events to this document.
- o Added basic event definition guidelines
- o Made protocol_type an array instead of a string (#146)

A.2. Since draft-marx-qlog-main-schema-01:

- o Decoupled qlog from the JSON format and described a mapping instead (#89)
 - * Data types are now specified in this document and proper definitions for fields were added in this format
 - * 64-bit numbers can now be either strings or numbers, with a preference for numbers (#10)
 - * binary blobs are now logged as lowercase hex strings (#39, #36)
 - * added guidance to add length-specifiers for binary blobs (#102)
- o Removed "time_units" from Configuration. All times are now in ms instead (#95)
- o Removed the "event_fields" setup for a more straightforward JSON format (#101, #89)
- o Added a streaming option using the NDJSON format (#109, #2, #106)
- o Described optional optimization options for implementers (#30)
- o Added QLOGDIR and QLOGFILE environment variables, clarified the .well-known URL usage (#26, #33, #51)
- o Overall tightened up the text and added more examples

A.3. Since draft-marx-qlog-main-schema-00:

- o All field names are now lowercase (e.g., category instead of CATEGORY)
- o Triggers are now properties on the "data" field value, instead of separate field types (#23)
- o group_ids in common_fields is now just also group_id

Appendix B. Design Variations

- o Quic-trace [17] takes a slightly different approach based on protocolbuffers.
- o Spindump [18] also defines a custom text-based format for in-network measurements
- o Wireshark [19] also has a QUIC dissector and its results can be transformed into a json output format using tshark.

The idea is that qlog is able to encompass the use cases for both of these alternate designs and that all tooling converges on the qlog standard.

Appendix C. Acknowledgements

Much of the initial work by Robin Marx was done at Hasselt University.

Thanks to Jana Iyengar, Brian Trammell, Dmitri Tikhonov, Stephen Petrides, Jari Arkko, Marcus Ihlar, Victor Vasiliev, Mirja Kuehlewind, Jeremy Laine and Lucas Pardue for their feedback and suggestions.

Authors' Addresses

Robin Marx (editor)
KU Leuven

Email: robin.marx@kuleuven.be

Luca Niccolini (editor)
Facebook

Email: lniccolini@fb.com

Marten Seemann (editor)
Protocol Labs

Email: marten@protocol.ai

IP Flow Information Export
Internet-Draft
Intended status: Standards Track
Expires: August 8, 2021

C. Munukutla
S. Vaid
Juniper Networks, Inc.
A. Mahale
D. Patel
Google, Inc.
February 4, 2021

IP Flow Information Export (IPFIX) Information Elements Extension for
Forwarding Exceptions
draft-mvmd-opsawg-ipfix-fwd-exceptions-02

Abstract

This draft proposes couple of new Forwarding exceptions related Information Elements (IEs) and Templates for the IP Flow Information Export (IPFIX) protocol. These new Information Elements and Exception Template can be used to export information about any forwarding errors in a network. This essential information is adequate to correlate packet drops to any control plane entity and map it to an impacted service. Once exceptions are correlated to a particular entity, an action can be assigned to mitigate such problems essentially enabling self-driving networks.

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 August 8, 2021.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
1.2. Requirements Language	3
2. Scope	4
3. Information Elements	4
4. New Information Elements	6
4.1. Proposed New Information Elements	6
4.2. Definition of Exceptions	6
4.2.1. forwardingExceptionCode	6
4.2.2. forwardingNextHopId	7
5. Exception Templates	8
5.1. IPFIX Exception Template 1 for Forwarding Exceptions	9
5.2. IPFIX Exception Template 2 for Forwarding Exceptions	9
6. IANA Considerations	10
6.1. Information Elements	10
6.2. Forwarding Exception Codes	11
7. Security Considerations	11
8. Contributors	11
9. References	12
9.1. Normative References	12
9.2. Informative References	12
Authors' Addresses	13

1. Introduction

All networks are susceptible to traffic drops due to a number of factors. Traffic drops can go unnoticed unless they are service impacting. In a multi-layered network architecture, it is tedious manual work to localize and root cause traffic blackholing issues. Transient drops are even harder to detect. Existing methodologies that rely on periodically monitoring interfaces on several hosts in a network does not guarantee timely detection, and are not scalable for large networks.

In order to eliminate this tedious monitoring work-flow, objective is to simplify localization and build correlation of dropped packets to

particular entity. The network entity shall identify the dropped packets by monitoring dropped counters or doing a deep packet inspection of the packet discarded by the forwarding ASIC. The implementation of the method used to detect the drop is outside the scope of this document. Dropped packets will be sampled in the forwarding-path and sent to a host or software queue along with type of exception, in/out interface information and other relevant meta data. This will be a push model where the node encountering the error will emit the information about dropped packets and associated meta-data. Techniques for IP Packet Selection [RFC5475] describes Sampling and Filtering techniques for IP packet selection either using Systematic Sampling or Random Sampling.

The IPFIX Protocol Specification [RFC7011] defines a generic exchange mechanism for collecting flow information. It supports source-triggered export of information via the push model approach. The IPFIX Information Model [IANA-IPFIX] defines a list of standard Information Elements (IEs) which can be carried by the IPFIX protocol.

This document focuses on telemetry information for dropped packet exceptions, and proposes an extension to IPFIX message format for collecting sampled exceptions. Some of the IPFIX Information Elements (IEs) already exist, some will be defined along with corresponding formats. It is also possible to achieve sampling of the dropped packets by using sampling methods like SFLOW but details of other sampling methods are outside the scope of this document.

1.1. Terminology

IPFIX-specific terminology (e.g. Information Element, Template, Template Record, Options Template Record, Template Set, Collector, Exporter, Data Record) used in this document is defined in Section 2 of [RFC7011]. As in [RFC7011] these IPFIX-specific terms have the first letter of a word capitalized. This document also makes use of the same terminology and definitions as Section 2 of [RFC5470].

1.2. 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.

2. Scope

This document specifies the information model used for reporting packet-based forwarding exceptions. [RFC7011] provides guidance on the choices of the transport protocols used for IPFIX and their effects. Encoded IPFIX exception packets need to be reliably transported to the collector. The choice of the actual transport protocol is beyond the scope of this document.

This document assumes that all devices reporting exceptions will use existing IPFIX framework/module to send encoded packets to the collector. This would mean that the network device will specify the template that it is going to use for each of the events. The templates can be of varying length, and there could be multiple templates that a network device could use to encode the exceptions.

The implementation details of the collector application are beyond the scope of this document.

3. Information Elements

The Exception template could contain a subset of the IEs shown in Table 1, depending upon the exception reported.

Whenever packet drop happens inside forwarding plane, following information is key to understanding the issue: reason for packet drop, flow which encountered the drop (packet content), additional meta-data e.g. flow direction (ingress/egress), nexthop index, input interface, output interface, etc. on which this packet was flowing.

The following table includes all the existing IEs that a device reporting IPFIX Exceptions using Exception Template would typically need. The formats of IEs and IPFIX IDs are listed in the table below.

Field Name	Size (bits)	IANA IPFIX ID	Description
flowDirection	8	61	The direction of the Flow observed at Observation point.
ingressInterface	32	10	Index of IP interface where packets of this flow are being received.
egressInterface	32	14	Index of IP interface where packets of this flow are being sent.
dataLinkFrameSize	16	312	Specified length of data link frame.
dataLinkFrameSection	65535	315	Carries n octets from data link frame of selected frame.
commonPropertiesID	64	137	Identifier of a set of common properties that is unique per observation domain.

Table 1: Forwarding Exception Information Elements

Information Elements

4. New Information Elements

4.1. Proposed New Information Elements

The proposed new IEs that a device reporting Exceptions using Exception template would need are listed in Table 2 below.

Field Name	Abstract Data Type	Description
forwardingExceptionCode	unsigned32	Unique code for every exception
forwardingNextHopID	unsigned64	Forwarding NH - index associated with packet that encountered this exception

Table 2: New Information Elements

New Information Elements

The Information Elements defined in Table 2 are proposed to be incorporated into the IANA IPFIX Information Elements registry [IANA-IPFIX]

4.2. Definition of Exceptions

Every network will encounter issues like packet loss, from time to time. Some of the causes for such a loss of traffic or a block in transmission of data packets include overloaded system conditions, misconfiguration, profiles and policies that restrict the bandwidth or priority of traffic, network outages, or disruption with physical cable faults. Packet loss could also happen because of incorrect stitching of the forwarding path or a mismatch between control plane and data plane state. Exception code entails the reason/error code due to which this packet has been dropped.

4.2.1. forwardingExceptionCode

forwardingExceptionCode will be defined in "IPFIX Information Elements" registry. This list can be expanded in the future as necessary. The data record will have corresponding exception code value to indicate forwarding error that caused the traffic drop.

An implementation may choose to encode device internal exception codes as forwardingExceptionCode. In such scenarios, Enterprise Bit MUST be set to 1 and corresponding Enterprise Number MUST be present as described in [RFC7011]

There is an existing IE 89 - forwardingStatus [IANA-IPFIX] but it allows a very limited number of exceptions to be reported from the system (6-bit reason code). The exception codes also need to be standardized for use. Different forwarding ASICs would have different pipelines and hence discard reasons (which could be very specific to that pipeline) cannot be generalized. Hence it makes sense to have a standalone IE for reporting exception which not only provides support to report larger number of exceptions but also provides freedom for reporting application specific exceptions using the enterprise bit.

A list of commonly used forwarding Exception codes will be identified and listed as part of Table 3 below.

Forwarding Exception Code	Reason
1	FIREWALL_DISCARD
2	TTL_EXPIRY
3	DISCARD_ROUTE
4	BAD_IPV4_CHECKSUM
5	REJECT_ROUTE
6	BAD_IPV4_HEADER (Version incorrect or IHL < 5)
7	BAD_IPV6_HEADER (Version incorrect)
8	BAD_IPV4_HEADER_LENGTH (V4 frame is too short)
9	BAD_IPV6_HEADER_LENGTH
10	BAD_IPV6_OPTIONS_PACKET (too many option headers)
..	..

Table 3: Exception Codes

4.2.2. forwardingNextHopId

In terms of a network device, next hop is the gateway to which packet should be forwarded corresponding to the path to final destination. A given router doesn't need to store the entire forwarding path information for a destination. As long as it can identify the next hop to be used for forwarding to a destination, the end to end forwarding can happen. This helps reduce size of forwarding table. The nexthop index uniquely identifies the egress path a packet would take to reach the destination. This could include information about

the outgoing interface, layer 2 address to be used, forwarding features configured for the packet path etc.

For instance, consider we have a L3VPN topology like below

CE1 ----- PE1 ----- MPLS Network ----- PE2 ----- CE2

Figure 1: MPLS VPN Network

Figure 1 above illustrates an example where reporting of exception can provide an insight into the error scenario. CE1 and CE2 communicate with each other over an MPLS VPN network. The labels are typically advertised using protocols like RSVP or LDP. When a packet is received from core network on PE1, a lookup on MPLS label results in packet getting forwarded towards CE1. The entries in MPLS table are populated by corresponding protocol. If label entries don't get populated in the MPLS table due to a probable glitch in the protocol configuration or some software inconsistency, the packets traversing on that LSP tunnel path shall get discarded on PE1.

In case of route lookups, that result in hierarchical forwarding chains, the mis-programming may manifest at different levels of the forwarding structure. The forwarding lookup may fail on any level of the hierarchy in the forwarding chain. It is expected that software at least report the nexthop where the lookup terminates. Its desirable for software to report the top level nexthop in the chain.

Using the mechanism described in this RFC, it will be possible to capture such packets and report them in IPFIX format with corresponding exception set (eg. DISCARD_ROUTE) along with relevant packet bytes and meta-data. This can help the operator/software to immediately understand root cause of the problem and take appropriate action.

An implementation may choose to report linecard number, linecard type, forwarding ASIC type and forwarding ASIC number on which an exception occurs, but mechanism to export these fields is out of the scope of this document.

5. Exception Templates

This section presents a list of templates for reporting exceptions using newly proposed IEs in addition to few existing IEs.

5.1. IPFIX Exception Template 1 for Forwarding Exceptions

Exception Template defined in Figure 1 may be used to export forwarding Exceptions.

Set ID = 2										Length = N octets										
Template ID = 256										Field Count = N										
0	forwardingExceptionCode										Field Length = 4									
0	forwardingNextHopId										Field Length = 8									
0	flowDirection										Field Length = 1									
0	ingressInterface										Field Length = 4									
0	egressInterface										Field Length = 4									
0	dataLinkFrameSize										Field Length = 2									
0	dataLinkFrameSection										Field Length = 65535									
Padding (opt)																				

IPFIX Exception Template for Forwarding Exceptions

5.2. IPFIX Exception Template 2 for Forwarding Exceptions

Alternatively, Exception Template defined in Figure 2 may be used. This includes Information Element 137 to represent following fields: forwardingNextHopId, ingressInterface, underlyingIngressInterface and egressInterface.

Set ID = 2										Length = N octets										
Template ID = 256										Field Count = N										
0	forwardingExceptionCode										Field Length = 4									
0	flowDirection										Field Length = 1									
0	commonPropertiesId1										Field Length = 8									
0	commonPropertiesId2										Field Length = 8									
0	commonPropertiesId3										Field Length = 8									
0	commonPropertiesId4										Field Length = 8									
0	dataLinkFrameSize										Field Length = 2									
0	dataLinkFrameSection										Field Length = 65535									
Padding (opt)																				

IPFIX Exception Template 2 for Forwarding Exceptions

6. IANA Considerations

6.1. Information Elements

IANA manages the IPFIX Information Elements registry at [IANA-IPFIX]. This document introduces two new IPFIX Information Elements.

Name: forwardingExceptionCode

ElementID: TBD

Description: Exception code is an identifier uniquely describing cause of irregularity or traffic drop on a device.

Abstract Data Type: unsigned32

Data Type Semantics: identifier

Name: forwardingNexthopId

ElementID: TBD

Description: Nexthop ID is a unique identifier for a Nexthop on a device.

Abstract Data Type: unsigned64

Data Type Semantics: identifier

6.2. Forwarding Exception Codes

This document requests addition of a new registry for Forwarding Exception Codes.

Forwarding Exception Code	Reason
1	FIREWALL_DISCARD
2	TTL_EXPIRY
3	DISCARD_ROUTE
4	BAD_IPV4_CHECKSUM
5	REJECT_ROUTE
6	BAD_IPV4_HEADER (Version incorrect or IHL < 5)
7	BAD_IPV6_HEADER (Version incorrect)
8	BAD_IPV4_HEADER_LENGTH (V4 frame is too short)
9	BAD_IPV6_HEADER_LENGTH
10	BAD_IPV6_OPTIONS_PACKET (too many option headers)
..	..

Table 3: Exception Codes

All assignments in this registry are to be performed via Expert Review.

7. Security Considerations

Security Considerations listed in detail for IPFIX in [RFC7011] apply to this document as well. As described in [RFC7011], the IPFIX messages exchanged between network device and collector MUST be protected to provide confidentiality, integrity, and authenticity. Without those characteristics, the messages are subject to various kinds of attacks. These attacks are described in great detail in [RFC7011].

8. Contributors

Manikandan Musuvathi Poornachary
 Juniper Networks, Inc.
 Electra Exora Business Park~Marathahalli-Sarjapur Outer Ring Road,
 Bangalore, KA - 560103
 India
 Email: mpoornachary@juniper.net

Vishnu Pavan Beeram
Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, CA 94089
USA
Email: vbeeram@juniper.net

Raveendra Torvi
Juniper Networks, Inc.
10 Technology Park Dr
Westford, MA 01886
USA
Email: rtorvi@juniper.net

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5475] Zseby, T., Molina, M., Duffield, N., Niccolini, S., and F. Raspall, "Sampling and Filtering Techniques for IP Packet Selection", RFC 5475, DOI 10.17487/RFC5475, March 2009, <<https://www.rfc-editor.org/info/rfc5475>>.
- [RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <<https://www.rfc-editor.org/info/rfc7011>>.
- [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>>.

9.2. Informative References

- [IANA-IPFIX] IANA, "IP Flow Information Export (IPFIX) Entities", <<https://www.iana.org/assignments/ipfix>>.
- [RFC5470] Sadasivan, G., Brownlee, N., Claise, B., and J. Quittek, "Architecture for IP Flow Information Export", RFC 5470, DOI 10.17487/RFC5470, March 2009, <<https://www.rfc-editor.org/info/rfc5470>>.

Authors' Addresses

Venkata Naga Chaitanya Munukutla
Juniper Networks, Inc.
10 Technology Park Dr
Westford, MA 01886
USA

Email: vmunuku@juniper.net

Shivam Vaid
Juniper Networks, Inc.
Electra, Exora Business Park- Marathahalli-Sarjapur Outer Ring Road
Bangalore, Karnataka 560103
India

Email: shivamv@juniper.net

Aditya Mahale
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
USA

Email: amahale@google.com

Devang Patel
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
USA

Email: pateldevang@google.com

Ops WG
Internet-Draft
Intended status: Standards Track
Expires: May 29, 2021

R. Hartmann, Ed.
Grafana Labs
B. Kochie
GitLab
B. Brazil
Robust Perception
R. Skillington
Chronosphere
November 25, 2020

OpenMetrics, a cloud-native, highly scalable metrics protocol
draft-richih-opsawg-openmetrics-00

Abstract

OpenMetrics specifies today's de-facto standard for transmitting cloud-native metrics at scale, with support for both text representation and Protocol Buffers and brings it into IETF. It supports both pull and push-based data collection.

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 May 29, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	3
1.1. Requirements Language	4
2. Overview	4
2.1. Metrics and Time Series	4
3. Data Model	4
3.1. Data Types	5
3.1.1. Values	5
3.1.2. Timestamps	5
3.1.3. Strings	5
3.1.4. Label	5
3.1.5. LabelSet	5
3.1.6. MetricPoint	5
3.1.7. Exemplars	6
3.1.8. Metric	6
3.1.9. MetricFamily	6
3.2. Metric Types	8
3.2.1. Gauge	8
3.2.2. Counter	8
3.2.3. StateSet	9
3.2.4. Info	9
3.2.5. Histogram	10
3.2.6. GaugeHistogram	11
3.2.7. Summary	11
3.2.8. Unknown	12
4. Data transmission & wire formats	12
4.1. Protocol Negotiation	12
4.1.1. ABNF	13
4.1.2. Overall Structure	14
4.1.3. MetricFamily	17
4.1.4. MetricPoint	18
4.1.5. Metric types	19
4.2. Protobuf format	23
4.2.1. Overall Structure	23
4.2.2. Protobuf schema	24
5. Design Considerations	28
5.1. Scope	28
5.1.1. Out of scope	29
5.2. Extensions and Improvements	29
5.3. Units and Base Units	29
5.4. Statelessness	31
5.5. Exposition Across Time and Metric Evolution	31

5.6.	NaN	32
5.7.	Missing Data	32
5.8.	Exposition Performance	33
5.9.	Concurrency	33
5.10.	Metric Naming and Namespaces	33
5.11.	Label Namespacing	35
5.12.	Metric Names versus Labels	36
5.13.	Types of Metadata	37
5.13.1.	Supporting Target Metadata in both Push-based and Pull-based Systems	37
5.14.	Client Calculations and Derived Metrics	38
5.15.	Number Types	39
5.16.	Exposing Timestamps	39
5.16.1.	Tracking When Metrics Last Changed	40
5.17.	Thresholds	41
5.18.	Size Limits	42
6.	Security Considerations	43
7.	IANA Considerations	43
8.	References	44
8.1.	Normative References	44
8.2.	Informative References	44
	Authors' Addresses	45

1. Introduction

Created in 2012, Prometheus has been the default for cloud-native observability since 2015. A central part of Prometheus' design is its text metric exposition format, called the Prometheus exposition format 0.0.4, stable since 2014. In this format, special care has been taken to make it easy to generate, to ingest, and to understand by humans. As of 2020, there are more than 700 publicly listed exporters, an unknown number of unlisted exporters, and thousands of native library integrations using this format. Dozens of ingestors from various projects and companies support consuming it.

With OpenMetrics, we are cleaning up and tightening the specification with the express purpose of bringing it into IETF. We are documenting a working standard with wide and organic adoption while introducing minimal, largely backwards-compatible, and well-considered changes. As of 2020, dozens of exporters, integrations, and ingestors use and preferentially negotiate OpenMetrics already.

Given the wide adoption and significant coordination requirements in the ecosystem, sweeping changes to either the Prometheus exposition format 0.0.4 or OpenMetrics 1.0 are considered out of scope.

1.1. 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.

2. Overview

Metrics are a specific kind of telemetry data. They represent a snapshot of the current state for a set of data. They are distinct from logs or events, which focus on records or information about individual events.

OpenMetrics is primarily a wire format, independent of any particular transport for that format. The format is expected to be consumed on a regular basis and to be meaningful over successive expositions.

Implementers MUST expose metrics in the OpenMetrics text format in response to a simple HTTP GET request to a documented URL for a given process or device. This endpoint SHOULD be called `"/metrics"`. Implementers MAY also expose OpenMetrics formatted metrics in other ways, such as by regularly pushing metric sets to an operator-configured endpoint over HTTP.

2.1. Metrics and Time Series

This standard expresses all system states as numerical values; counts, current values, enumerations, and boolean states being common examples. Contrary to metrics, singular events occur at a specific time. Metrics tend to aggregate data temporally. While this can lose information, the reduction in overhead is an engineering trade-off commonly chosen in many modern monitoring systems.

Time series are a record of changing information over time. While time series can support arbitrary strings or binary data, only numeric data is in scope for this RFC.

Common examples of metric time series would be network interface counters, device temperatures, BGP connection states, and alert states.

3. Data Model

This section MUST be read together with the ABNF section. In case of disagreements between the two, the ABNF's restrictions MUST take

precedence. This reduces repetition as the text wire format MUST be supported.

3.1. Data Types

3.1.1. Values

Metric values in OpenMetrics MUST be either floating points or integers. Note that ingestors of the format MAY only support float64. The non-real values NaN, +Inf and -Inf MUST be supported. NaN MUST NOT be considered a missing value, but it MAY be used to signal a division by zero.

3.1.1.1. Booleans

Boolean values MUST follow 1==true, 0==false.

3.1.2. Timestamps

Timestamps MUST be Unix Epoch in seconds. Negative timestamps MAY be used.

3.1.3. Strings

Strings MUST only consist of valid UTF-8 characters and MAY be zero length. NULL (ASCII 0x0) MUST be supported.

3.1.4. Label

Labels are key-value pairs consisting of strings.

Label names beginning with underscores are RESERVED and MUST NOT be used unless specified by this standard. Label names MUST follow the restrictions in the ABNF section.

Empty label values SHOULD be treated as if the label was not present.

3.1.5. LabelSet

A LabelSet MUST consist of Labels and MAY be empty. Label names MUST be unique within a LabelSet.

3.1.6. MetricPoint

Each MetricPoint consists of a set of values, depending on the MetricFamily type.

3.1.7. Exemplars

Exemplars are references to data outside of the MetricSet. A common use case are IDs of program traces.

Exemplars MUST consist of a LabelSet and a value, and MAY have a timestamp. They MAY each be different from the MetricPoints' LabelSet and timestamp.

The combined length of the label names and values of an Exemplar's LabelSet MUST NOT exceed 128 UTF-8 characters. Other characters in the text rendering of an exemplar such as ",=" are not included in this limit for implementation simplicity and for consistency between the text and proto formats.

Ingestors MAY discard exemplars.

3.1.8. Metric

Metrics are defined by a unique LabelSet within a MetricFamily. Metrics MUST contain a list of one or more MetricPoints. Metrics with the same name for a given MetricFamily SHOULD have the same set of label names in their LabelSet.

MetricPoints SHOULD NOT have explicit timestamps.

If more than one MetricPoint is exposed for a Metric, then its MetricPoints MUST have monotonically increasing timestamps.

3.1.9. MetricFamily

A MetricFamily MAY have zero or more Metrics. A MetricFamily MUST have a name, HELP, TYPE, and UNIT metadata. Every Metric within a MetricFamily MUST have a unique LabelSet.

3.1.9.1. Name

MetricFamily names are a string and MUST be unique within a MetricSet. Names SHOULD be in snake_case. Metric names MUST follow the restrictions in the ABNF section.

Colons in MetricFamily names are RESERVED to signal that the MetricFamily is the result of a calculation or aggregation of a general purpose monitoring system.

MetricFamily names beginning with underscores are RESERVED and MUST NOT be used unless specified by this standard.

3.1.9.1.1. Suffixes

The name of a MetricFamily MUST NOT result in a potential clash for sample metric names as per the ABNF with another MetricFamily in the Text Format within a MetricSet. An example would be a gauge called "foo_created" as a counter called "foo" could create a "foo_created" in the text format.

Exposers SHOULD avoid names that could be confused with the suffixes that text format sample metric names use.

o Suffixes for the respective types are:

- o Counter: '_total', '_created'
- o Summary: '_count', '_sum', '_created', '' (empty)
- o Histogram: '_count', '_sum', '_bucket', '_created'
- o GaugeHistogram: '_gcount', '_gsum', '_bucket'
- o Info: '_info'
- o Gauge: '' (empty)
- o StateSet: '' (empty)
- o Unknown: '' (empty)

3.1.9.2. Type

Type specifies the MetricFamily type. Valid values are "unknown", "gauge", "counter", "stateset", "info", "histogram", "gaugehistogram", and "summary".

3.1.9.3. Unit

Unit specifies MetricFamily units. If non-empty, it MUST be a suffix of the MetricFamily name separated by an underscore. Be aware that further generation rules might make it an infix in the text format.

3.1.9.4. Help

Help is a string and SHOULD non-empty. It is used to give a brief description of the MetricFamily for human consumption and SHOULD be short enough to be used as a tooltip.

3.1.9.5. MetricSet

A MetricSet is the top level object exposed by OpenMetrics. It MUST consist of MetricFamilies and MAY be empty.

Each MetricFamily name MUST be unique. The same label name and value SHOULD NOT appear on every Metric within a MetricSet.

There is no specific ordering of MetricFamilies required within a MetricSet. An exposor MAY make an exposition easier to read for humans, for example sort alphabetically if the performance tradeoff makes sense.

If present, an Info MetricFamily called "target" per the "Supporting target metadata in both push-based and pull-based systems" section below SHOULD be first.

3.2. Metric Types

3.2.1. Gauge

Gauges are current measurements, such as bytes of memory currently used or the number of items in a queue. For gauges the absolute value is what is of interest to a user.

A MetricPoint in a Metric with the type gauge MUST have a single value.

Gauges MAY increase, decrease, or stay constant over time. Even if they only ever go in one direction, they might still be gauges and not counters. The size of a log file would usually only increase, a resource might decrease, and the limit of a queue size may be constant.

A gauge MAY be used to encode an enum where the enum has many states and changes over time, it is the most efficient but least user friendly.

3.2.2. Counter

Counters measure discrete events. Common examples are the number of HTTP requests received, CPU seconds spent, or bytes sent. For counters how quickly they are increasing over time is what is of interest to a user.

A MetricPoint in a Metric with the type Counter MUST have one value called Total. A Total is a non-NaN and MUST be monotonically non-decreasing over time, starting from 0.

A MetricPoint in a Metric with the type Counter SHOULD have a Timestamp value called Created. This can help ingestors discern between new metrics and long-running ones it did not see before.

A MetricPoint in a Metric's Counter's Total MAY reset to 0. If present, the corresponding Created time MUST also be set to the timestamp of the reset.

A MetricPoint in a Metric's Counter's Total MAY have an exemplar.

3.2.3. StateSet

StateSets represent a series of related boolean values, also called a bitset. If ENUMs need to be encoded this MAY be done via StateSet.

A point of a StateSet metric MAY contain multiple states and MUST contain one boolean per State. States have a name which are Strings.

A StateSet Metric's LabelSet MUST NOT have a label name which is the same as the name of its MetricFamily.

If encoded as a StateSet, ENUMs MUST have exactly one Boolean which is true within a MetricPoint.

This is suitable where the enum value changes over time, and the number of States isn't much more than a handful.

EDITOR'S NOTE: This might be better as Consideration

MetricFamilies of type StateSets MUST have an empty Unit string.

3.2.4. Info

Info metrics are used to expose textual information which SHOULD NOT change during process lifetime. Common examples are an application's version, revision control commit, and the version of a compiler.

A MetricPoint of an Info Metric contains a LabelSet. An Info MetricPoint's LabelSet MUST NOT have a label name which is the same as the name of a label of the LabelSet of its Metric.

Info MAY be used to encode ENUMs whose values do not change over time, such as the type of a network interface.

MetricFamilies of type Info MUST have an empty Unit string.

3.2.5. Histogram

Histograms measure distributions of discrete events. Common examples are the latency of HTTP requests, function runtimes, or I/O request sizes.

A Histogram MetricPoint MUST contain at least one bucket, and SHOULD contain Sum, and Created values. Every bucket MUST have a threshold and a value.

Histogram MetricPoints MUST have at least a bucket with an +Inf threshold. Buckets MUST be cumulative. As an example for a metric representing request latency in seconds its values for buckets with thresholds 1, 2, 3, and +Inf MUST follow $value_1 \leq value_2 \leq value_3 \leq value_{+Inf}$. If ten requests took 1 second each, the values of the 1, 2, 3, and +Inf buckets MUST equal 10.

The +Inf bucket counts all requests. If present, the Sum value MUST equal the Sum of all the measured event values. Bucket thresholds within a MetricPoint MUST be unique.

Semantically, Sum, and buckets values are counters so MUST NOT be NaN or negative. Negative threshold buckets MAY be used, but then the Histogram MetricPoint MUST NOT contain a sum value as it would no longer be a counter semantically. Bucket thresholds MUST NOT equal NaN. Count and bucket values MUST be integers.

A Histogram MetricPoint SHOULD have a Timestamp value called Created. This can help ingestors discern between new metrics and long-running ones it did not see before.

A Histogram's Metric's LabelSet MUST NOT have a "le" label name.

Bucket values MAY have exemplars. Buckets are cumulative to allow monitoring systems to drop any non-+Inf bucket for performance/anti-denial-of-service reasons in a way that loses granularity but is still a valid Histogram.

EDITOR'S NOTE: The second sentence is a consideration, it can be moved if needed

Each bucket covers the values less and or equal to it, and the value of the exemplar MUST be within this range. Exemplars SHOULD be put into the bucket with the highest value. A bucket MUST NOT have more than one exemplar.

3.2.6. GaugeHistogram

GaugeHistograms measure current distributions. Common examples are how long items have been waiting in a queue, or size of the requests in a queue.

A GaugeHistogram MetricPoint MUST have at least one bucket with an +Inf threshold, and SHOULD contain a Gsum value. Every bucket MUST have a threshold and a value.

The buckets for a GaugeHistogram follow all the same rules as for a Histogram.

The bucket and Gsum of a GaugeHistogram are conceptually gauges, however bucket values MUST NOT be negative or NaN. If negative threshold buckets are present, then sum MAY be negative. Gsum MUST NOT be NaN. Bucket values MUST be integers.

A GaugeHistogram's Metric's LabelSet MUST NOT have a "le" label name.

Bucket values can have exemplars.

Each bucket covers the values less and or equal to it, and the value of the exemplar MUST be within this range. Exemplars SHOULD be put into the bucket with the highest value. A bucket MUST NOT have more than one exemplar.

3.2.7. Summary

Summaries also measure distributions of discrete events and MAY be used when Histograms are too expensive and/or an average event size is sufficient.

They MAY also be used for backwards compatibility, because some existing instrumentation libraries expose precomputed quantiles and do not support Histograms. Precomputed quantiles SHOULD NOT be used, because quantiles are not aggregatable and the user often can not deduce what timeframe they cover.

A Summary MetricPoint MAY consist of a Count, Sum, Created, and a set of quantiles.

Semantically, Count and Sum values are counters so MUST NOT be NaN or negative. Count MUST be an integer.

A MetricPoint in a Metric with the type Summary which contains Count or Sum values SHOULD have a Timestamp value called Created. This can help ingestors discern between new metrics and long-running ones it

did not see before. Created MUST NOT relate to the collection period of quantile values.

Quantiles are a map from a quantile to a value. An example is a quantile 0.95 with value 0.2 in a metric called `myapp_http_request_duration_seconds` which means that the 95th percentile latency is 200ms over an unknown timeframe. If there are no events in the relevant timeframe, the value for a quantile MUST be NaN. A Quantile's Metric's LabelSet MUST NOT have "quantile" label name. Quantiles MUST be between 0 and 1 inclusive. Quantile values MUST NOT be negative. Quantile values SHOULD represent the recent values. Commonly this would be over the last 5-10 minutes.

3.2.8. Unknown

Unknown SHOULD NOT be used. Unknown MAY be used when it is impossible to determine the types of individual metrics from 3rd party systems.

A point in a metric with the unknown type MUST have a single value.

4. Data transmission & wire formats

The text wire format MUST be supported and is the default. The protobuf wire format MAY be supported and MUST ONLY be used after negotiation.

The OpenMetrics formats are Regular Chomsky Grammars, making writing quick and small parsers possible. The text format compresses well, and protobuf is already binary and efficiently encoded.

Partial or invalid expositions MUST be considered erroneous in their entirety.

4.1. Protocol Negotiation

All ingestor implementations MUST be able to ingest data secured with TLS 1.2 or later. All exposers SHOULD be able to emit data secured with TLS 1.2 or later. ingestor implementations SHOULD be able to ingest data from HTTP without TLS. All implementations SHOULD use TLS to transmit data.

Negotiation of what version of the OpenMetrics format to use is out-of-band. For example for pull-based exposition over HTTP standard HTTP content type negotiation is used, and MUST default to the oldest version of the standard (i.e. 1.0.0) if no newer version is requested.

Push-based negotiation is inherently more complex, as the exposers typically initiates the connection. Producers MUST use the oldest version of the standard (i.e. 1.0.0) unless requested otherwise by the ingestor. Text format

4.1.1. ABNF

ABNF as per RFC 5234

EDITOR'S NOTE: Should we update to RFC 7405, in particular the case insensitive bits?

"exposition" is the top level token of the ABNF.

```
exposition = metricset HASH SP eof [ LF ]
```

```
metricset = *metricfamily
```

```
metricfamily = *metric-descriptor *metric
```

```
metric-descriptor = HASH SP type SP metricname SP metric-type LF
```

```
metric-descriptor =/ HASH SP help SP metricname SP escaped-string LF
```

```
metric-descriptor =/ HASH SP unit SP metricname SP 1*metricname-char LF
```

```
metric = *sample
```

```
metric-type = counter / gauge / histogram / gaugehistogram / stateset
```

```
metric-type =/ info / summary / unknown
```

```
sample = metricname [labels] SP number [SP timestamp] [exemplar] LF
```

```
exemplar = SP HASH SP labels SP number [SP timestamp]
```

```
labels = "{" [label *(COMMA label)] "}"
```

```
label = label-name EQ DQUOTE escaped-string DQUOTE
```

```
number = realnumber
```

```
; Case insensitive
```

```
number =/ [SIGN] ("inf" / "infinity")
```

```
number =/ "nan"
```

```
timestamp = realnumber
```

```
; Not 100% sure this captures all float corner cases.
```

```
; Leading 0s explicitly okay
```

```
realnumber = [SIGN] 1*DIGIT
```

```
realnumber =/ [SIGN] 1*DIGIT [ "." *DIGIT ] [ "e" [SIGN] 1*DIGIT ]
```

```

realnumber =/ [SIGN] *DIGIT "." 1*DIGIT [ "e" [SIGN] 1*DIGIT ]

; RFC 5234 is case insensitive.
; Uppercase
eof = %d69.79.70
type = %d84.89.80.69
help = %d72.69.76.80
unit = %d85.78.73.84
; Lowercase
counter = %d99.111.117.110.116.101.114
gauge = %d103.97.117.103.101
histogram = %d104.105.115.116.111.103.114.97.109
gaugehistogram = gauge histogram
stateset = %d115.116.97.116.101.115.101.116
info = %d105.110.102.111
summary = %d115.117.109.109.97.114.121
unknown = %d117.110.107.110.111.119.110

BS = "\"
EQ = "="
COMMA = ","
HASH = "#"
SIGN = "-" / "+"

metricname = metricname-initial-char 0*metricname-char

metricname-char = metricname-initial-char / DIGIT
metricname-initial-char = ALPHA / "_" / ":"

label-name = label-name-initial-char *label-name-char

label-name-char = label-name-initial-char / DIGIT
label-name-initial-char = ALPHA / "_"

escaped-string = *escaped-char

escaped-char = normal-char
escaped-char =/ BS ("n" / DQUOTE / BS)

; Any unicode character, except newline, double quote, and backslash
normal-char = %x00-09 / %x0B-21 / %x23-5B / %x5D-D7FF / %xE000-10FFFF

```

4.1.2. Overall Structure

UTF-8 MUST be used. Byte order markers (BOMs) MUST NOT be used. As an important reminder for implementers, byte 0 is valid UTF-8 while, for example, byte 255 is not.

The content type MUST be: application/openmetrics-text;
version=1.0.0; charset=utf-8

Line endings MUST be signalled with line feed (\n) and MUST NOT contain carriage returns (\r). Expositions MUST end with EOF and SHOULD end with 'EOF\n'.

An example of a complete exposition: ~~~~ # TYPE
acme_http_router_request_seconds summary # UNIT
acme_http_router_request_seconds seconds # HELP
acme_http_router_request_seconds Latency though all of ACME's HTTP
request router. acme_http_router_request_seconds_sum{path="/api/
v1",method="GET"} 9036.32
acme_http_router_request_seconds_count{path="/api/v1",method="GET"}
807283.0 acme_http_router_request_seconds_created{path="/api/
v1",method="GET"} 1605281325.0
acme_http_router_request_seconds_sum{path="/api/v2",method="POST"}
479.3 acme_http_router_request_seconds_count{path="/api/
v2",method="POST"} 34.0
acme_http_router_request_seconds_created{path="/api/
v2",method="POST"} 1605281325.0 # TYPE go_goroutines gauge # HELP
go_goroutines Number of goroutines that currently exist.
go_goroutines 69 # TYPE process_cpu_seconds counter # UNIT
process_cpu_seconds seconds # HELP process_cpu_seconds Total user and
system CPU time spent in seconds. process_cpu_seconds_total
4.20072246e+06 # EOF ~~~~

4.1.2.1. Escaping

Where the ABNF notes escaping, the following escaping MUST be applied
Line feed, '\n' (0x0A) -> literally '\n' (Bytecode 0x5c 0x6e) Double
quotes -> '"' (Bytecode 0x5c 0x22) Backslash -> '\\' (Bytecode 0x5c
0x5c)

4.1.2.2. Numbers

Integer numbers MUST NOT have a decimal point. Examples are "23",
"0042", and "1341298465647914".

Floating point numbers MUST be represented either with a decimal
point or using scientific notation. Examples are "8903.123421" and
"1.89e-7". Floating point numbers MUST fit within the range of a
64-bit floating point value as defined by IEEE 754, but MAY require
so many bits in the mantissa that results in lost precision. This
MAY be used to encode nanosecond resolution timestamps.

Arbitrary integer and floating point rendering of numbers MUST NOT be used for "quantile" and "le" label values as in section "Canonical Numbers". They MAY be used anywhere else numbers are used.

4.1.2.2.1. Considerations: Canonical Numbers

Numbers in the "le" label values of histograms and "quantile" label values of summary metrics are special in that they're label values, and label values are intended to be opaque. As end users will likely directly interact with these string values, and as many monitoring systems lack the ability to deal with them as first-class numbers, it would be beneficial if a given number had the exact same text representation.

Consistency is highly desirable, but real world implementations of languages and their runtimes make mandating this impractical. The most important common quantiles are 0.5, 0.95, 0.9, 0.99, 0.999 and bucket values representing values from a millisecond up to 10.0 seconds, because those cover cases like latency SLAs and Apdex for typical web services. Powers of ten are covered to try to ensure that the switch between fixed point and exponential rendering is consistent as this varies across runtimes. The target rendering is equivalent to the default Go rendering of float64 values (i.e. %g), with a .0 appended in case there is no decimal point or exponent to make clear that they are floats.

Exposers MUST produce output for positive infinity as +Inf.

Exposers SHOULD produce output for the values 0.0 up to 10.0 in 0.001 increments in line with the following examples: 0.0 0.001 0.002 0.01 0.1 0.9 0.95 0.99 0.999 1.0 1.7 10.0

Exposers SHOULD produce output for the values 1e-10 up to 1e+10 in powers of ten in line with the following examples: 1e-10 1e-09 1e-05 0.0001 0.1 1.0 100000.0 1e+06 1e+10

Parsers MUST NOT reject inputs which are outside of the canonical values merely because they are not consistent with the canonical values. For example 1.1e-4 must not be rejected, even though it is not the consistent rendering of 0.00011.

Exposers SHOULD follow these patterns for non-canonical numbers, and the intention is by adjusting the rendering algorithm to be consistent for these values that the vast majority of other values will also have consistent rendering. Exposers using only a few particular le/quantile values could also hardcode. In languages such as C where a minimal floating point rendering algorithm such as

Grisu3 such as Grisu3 is not readily available, exposers MAY use a different rendering.

A warning to implementers in C and other languages that share its printf implementation: The standard precision of %f, %e and %g is only six significant digits. 17 significant digits are required for full precision, e.g. "printf("%.17g", d)".

4.1.2.3. Timestamps

Timestamps SHOULD NOT use exponential float rendering for timestamps if nanosecond precision is needed as rendering of a float64 does not have sufficient precision, e.g. 1604676851.123456789.

4.1.3. MetricFamily

There MUST NOT be an explicit separator between MetricFamilies. The next MetricFamily MUST be signalled with either metadata or a new sample metric name which cannot be part of the previous MetricFamily.

MetricFamilies MUST NOT be interleaved.

4.1.3.1. MetricFamily metadata

There are four pieces of metadata: The MetricFamily name, TYPE, UNIT and HELP. An example of the metadata for a counter Metric called foo is:

```
# TYPE foo counter
```

If no TYPE is exposed, the MetricFamily MUST be of type Unknown.

If a unit is specified it MUST be provided in a UNIT metadata line. In addition, an underscore and the unit MUST be the suffix of the MetricFamily name.

A valid example for a foo_seconds metric with a unit of "seconds":
~~~~ # TYPE foo\_seconds counter # UNIT foo\_seconds seconds ~~~~

An invalid example, where the unit is not a suffix on the name: ~~~~  
# TYPE foo counter # UNIT foo seconds ~~~~

It is also valid to have: ~~~~ # TYPE foo\_seconds counter ~~~~

If the unit is known it SHOULD be provided.

The value of a UNIT or HELP line MAY be empty. This MUST be treated as if no metadata line for the MetricFamily existed.

```
# TYPE foo_seconds counter
# UNIT foo_seconds seconds
# HELP foo_seconds Some text and \n some \" escaping
```

There MUST NOT be more than one of each type of metadata line for a MetricFamily. The ordering SHOULD be TYPE, UNIT, HELP.

Aside from this metadata and the EOF line at the end of the message, you MUST NOT expose lines beginning with a #.

#### 4.1.3.2. Metric

Metrics MUST NOT be interleaved.

See the example in "Text format -> MetricPoint". Labels A sample without labels or a timestamp and the value 0 MUST be rendered either like:

```
bar_seconds_count 0
```

or like

```
bar_seconds_count{} 0
```

Label values MAY be any valid UTF-8 value, so escaping MUST be applied as per the ABNF. A valid example with two labels: ~~~~  
bar\_seconds\_count{a="x",b="escaping" example \n "} 0 ~~~~

The rendering of values for a MetricPoint can include additional labels (e.g. the "le" label for a Histogram type), which MUST be rendered in the same way as a Metric's own LabelSet.

#### 4.1.4. MetricPoint

MetricPoints MUST NOT be interleaved.

A correct example where there were multiple MetricPoints and Samples within a MetricFamily would be:

```
# TYPE foo_seconds summary
# UNIT foo_seconds seconds
foo_seconds_count{a="bb"} 0 123
foo_seconds_sum{a="bb"} 0 123
foo_seconds_count{a="bb"} 0 456
foo_seconds_sum{a="bb"} 0 456
foo_seconds_count{a="ccc"} 0 123
foo_seconds_sum{a="ccc"} 0 123
foo_seconds_count{a="ccc"} 0 456
foo_seconds_sum{a="ccc"} 0 456
```

An incorrect example where Metrics are interleaved:

```
# TYPE foo_seconds summary
# UNIT foo_seconds seconds
foo_seconds_count{a="bb"} 0 123
foo_seconds_count{a="ccc"} 0 123
foo_seconds_count{a="bb"} 0 456
foo_seconds_count{a="ccc"} 0 456
```

An incorrect example where MetricPoints are interleaved:

```
# TYPE foo_seconds summary
# UNIT foo_seconds seconds
foo_seconds_count{a="bb"} 0 123
foo_seconds_count{a="bb"} 0 456
foo_seconds_sum{a="bb"} 0 123
foo_seconds_sum{a="bb"} 0 456
```

#### 4.1.5. Metric types

##### 4.1.5.1. Gauge

The Sample MetricName for the value of a MetricPoint for a MetricFamily of type Gauge MUST NOT have a suffix.

An example MetricFamily with a Metric with no labels and a MetricPoint with no timestamp: ~~~~ # TYPE foo gauge foo 17.0 ~~~~

An example of a MetricFamily with two Metrics with a label and MetricPoints with no timestamp: ~~~~ # TYPE foo gauge foo{a="bb"} 17.0 foo{a="ccc"} 17.0 ~~~~

An example of a MetricFamily with no Metrics: ~~~~ # TYPE foo gauge ~~~~

An example with a Metric with a label and a MetricPoint with a timestamp: ~~~~ # TYPE foo gauge foo{a="b"} 17.0 1520879607.789 ~~~~

An example with a Metric with no labels and MetricPoint with a timestamp: ~~~~ # TYPE foo gauge foo 17.0 1520879607.789 ~~~~

An example with a Metric with no labels and two MetricPoints with timestamps: ~~~~ # TYPE foo gauge foo 17.0 123 foo 18.0 456 ~~~~

#### 4.1.5.2. Counter

The MetricPoint's Total Value Sample MetricName MUST have the suffix "\_total". If present the MetricPoint's Created Value Sample MetricName MUST have the suffix "\_created".

An example with a Metric with no labels, and a MetricPoint with no timestamp and no created: ~~~~ # TYPE foo counter foo\_total 17.0 ~~~~

An example with a Metric with no labels, and a MetricPoint with a timestamp and no created: ~~~~ # TYPE foo counter foo\_total 17.0 1520879607.789 ~~~~

An example with a Metric with no labels, and a MetricPoint with no timestamp and a created: ~~~~ # TYPE foo counter foo\_total 17.0 foo\_created 1520430000.123 ~~~~

An example with a Metric with no labels, and a MetricPoint with a timestamp and a created: ~~~~ # TYPE foo counter foo\_total 17.0 1520879607.789 foo\_created 1520430000.123 1520879607.789 ~~~~

Exemplars MAY be attached to the MetricPoint's Total sample.

#### 4.1.5.3. StateSet

The Sample MetricName for the value of a MetricPoint for a MetricFamily of type StateSet MUST NOT have a suffix.

StateSets MUST have one sample per State in the MetricPoint. Each State's sample MUST have a label with the MetricFamily name as the label name and the State name as the label value. The State sample's value MUST be 1 if the State is true and MUST be 0 if the State is false.

An example with the states "a", "bb", and "ccc" in which only the value b is enabled and the metric name is foo:

```
# TYPE foo stateset
foo{foo="a"} 0
foo{foo="bb"} 1
foo{foo="ccc"} 0
```

An example of an "entity" label on the Metric: ~~~~ # TYPE foo  
 stateset foo{entity="controller",foo="a"} 1.0  
 foo{entity="controller",foo="bb"} 0.0  
 foo{entity="controller",foo="ccc"} 0.0 foo{entity="replica",foo="a"}  
 1.0 foo{entity="replica",foo="bb"} 0.0  
 foo{entity="replica",foo="ccc"} 1.0 ~~~~

#### 4.1.5.4. Info

The Sample MetricName for the value of a MetricPoint for a MetricFamily of type Info MUST have the suffix "\_info". The Sample value MUST always be 1.

An example of a Metric with no labels, and one MetricPoint value with "name" and "version" labels: ~~~~ # TYPE foo info  
 foo\_info{name="pretty name",version="8.2.7"} 1 ~~~~

An example of a Metric with label "entity" and one MetricPoint value with "name" and "version" labels: ~~~~ # TYPE foo info  
 foo\_info{entity="controller",name="pretty name",version="8.2.7"} 1.0  
 foo\_info{entity="replica",name="prettier name",version="8.1.9"} 1.0  
 ~~~~

Metric labels and MetricPoint value labels MAY be in any order.

4.1.5.5. Summary

If present, the MetricPoint's Sum Value Sample MetricName MUST have the suffix "_sum". If present, the MetricPoint's Count Value Sample MetricName MUST have the suffix "_count". If present, the MetricPoint's Created Value Sample MetricName MUST have the suffix "_created". If present, the MetricPoint's Quantile Values MUST specify the quantile measured using a label with a label name of "quantile" and with a label value of the quantile measured.

An example of a Metric with no labels and a MetricPoint with Sum, Count and Created values: ~~~~ # TYPE foo summary foo_count 17.0
 foo_sum 324789.3 foo_created 1520430000.123 ~~~~

An example of a Metric with no labels and a MetricPoint with two quantiles: ~~~~ # TYPE foo summary foo{quantile="0.95"} 123.7
 foo{quantile="0.99"} 150.0 ~~~~

Quantiles MAY be in any order.

4.1.5.6. Histogram

The MetricPoint's Bucket Values Sample MetricNames MUST have the suffix "_bucket". If present, the MetricPoint's Sum Value Sample MetricName MUST have the suffix "_sum". If present, the MetricPoint's Created Value Sample MetricName MUST have the suffix "_created". If and only if a Sum Value is present in a MetricPoint, then the MetricPoint's +Inf Bucket value MUST also appear in a Sample with a MetricName with the suffix "_count".

Buckets MUST be sorted in number increasing order of "le", and the value of the "le" label MUST follow the rules for Canonical Numbers.

An example of a Metric with no labels and a MetricPoint with Sum, Count, and Created values, and with 12 buckets. A wide and atypical but valid variety of "le" values is shown on purpose: ~~~~ # TYPE foo histogram foo_bucket{le="0.0"} 0 foo_bucket{le="1e-05"} 0 foo_bucket{le="0.0001"} 5 foo_bucket{le="0.1"} 8 foo_bucket{le="1.0"} 10 foo_bucket{le="10.0"} 11 foo_bucket{le="100000.0"} 11 foo_bucket{le="1e+06"} 15 foo_bucket{le="1e+23"} 16 foo_bucket{le="1.1e+23"} 17 foo_bucket{le="+Inf"} 17 foo_count 17 foo_sum 324789.3 foo_created 1520430000.123 ~~~~

4.1.5.6.1. Exemplars

Exemplars without Labels MUST represent an empty LabelSet as {}.

An example of Exemplars showcasing several valid cases: The "0.01" bucket has no Exemplar. The 0.1 bucket has an Exemplar with no Labels. The 1 bucket has an Exemplar with one Label. The 10 bucket has an Exemplar with a Label and a timestamp. In practice all buckets SHOULD have the same style of Exemplars. ~~~~ # TYPE foo histogram foo_bucket{le="0.01"} 0 foo_bucket{le="0.1"} 8 # {} 0.054 foo_bucket{le="1"} 11 # {trace_id="K005S4vxi0o"} 0.67 foo_bucket{le="10"} 17 # {trace_id="oHg5SJYRHA0"} 9.8 1520879607.789 foo_bucket{le="+Inf"} 17 foo_count 17 foo_sum 324789.3 foo_created 1520430000.123 ~~~~

4.1.5.7. GaugeHistogram

The MetricPoint's Bucket Values Sample MetricNames MUST have the suffix "_bucket". If present, the MetricPoint's Sum Value Sample MetricName MUST have the suffix "_gsum". If present, the MetricPoint's Created Value Sample MetricName MUST have the suffix "_created". If and only if a Sum Value is present in a MetricPoint, then the MetricPoint's +Inf Bucket value MUST also appear in a Sample with a MetricName with the suffix "_gcount".

Buckets MUST be sorted in number increasing order of "le", and the value of the "le" label MUST follow the rules for Canonical Numbers.

An example of a Metric with no labels, and one MetricPoint value with no Exemplar with no Exemplars in the buckets: ~~~~ # TYPE foo gaugehistogram foo_bucket{le="0.01"} 20.0 foo_bucket{le="0.1"} 25.0 foo_bucket{le="1"} 34.0 foo_bucket{le="10"} 34.0 foo_bucket{le="+Inf"} 42.0 foo_gcount 42.0 foo_gsum 3289.3 foo_created 1520430000.123 ~~~~

4.1.5.8. Unknown

The sample metric name for the value of the MetricPoint for a MetricFamily of type Unknown MUST NOT have a suffix.

An example with a Metric with no labels and a MetricPoint with no timestamp: ~~~~ # TYPE foo unknown foo 42.23 ~~~~

4.2. Protobuf format

4.2.1. Overall Structure

Protobuf messages MUST be encoded in binary and MUST have "application/openmetrics-protobuf; version=1.0.0" as their content type.

All payloads MUST be a single binary encoded MetricSet message, as defined by the OpenMetrics protobuf schema.

4.2.1.1. Version

The protobuf format MUST follow the proto3 version of the protocol buffer language.

4.2.1.2. Strings

All string fields MUST be UTF-8 encoded.

4.2.1.3. Timestamps

Timestamp representations in the OpenMetrics protobuf schema MUST follow the published google.protobuf.Timestamp [timestamp] message. The timestamp message MUST be in Unix epoch seconds as an int64 and a non-negative fraction of a second at nanosecond resolution as an int32 that counts forward from the seconds timestamp component. It MUST be within 0 to 999,999,999 inclusive.

4.2.2. Protobuf schema

```
syntax = "proto3";

// The OpenMetrics protobuf schema which defines the protobuf wire
// format.
// Ensure to interpret "required" as semantically required for a valid
// message.
// All string fields MUST be UTF-8 encoded strings.
package openmetrics;

import "google/protobuf/timestamp.proto";

// The top-level container type that is encoded and sent over the wire.
message MetricSet {
    // Each MetricFamily has one or more MetricPoints for a single Metric.
    repeated MetricFamily metric_families = 1;
}

// One or more Metrics for a single MetricFamily, where each Metric
// has one or more MetricPoints.
message MetricFamily {
    // Required.
    string name = 1;

    // Optional.
    MetricType type = 2;

    // Optional.
    string unit = 3;

    // Optional.
    string help = 4;

    // Optional.
    repeated Metric metrics = 5;
}

// The type of a Metric.
enum MetricType {
    // Unknown must use unknown MetricPoint values.
    UNKNOWN = 0;
    // Gauge must use gauge MetricPoint values.
    GAUGE = 1;
    // Counter must use counter MetricPoint values.
    COUNTER = 2;
    // State set must use state set MetricPoint values.
    STATE_SET = 3;
}
```

```
// Info must use info MetricPoint values.
INFO = 4;
// Histogram must use histogram value MetricPoint values.
HISTOGRAM = 5;
// Gauge histogram must use histogram value MetricPoint values.
GAUGE_HISTOGRAM = 6;
// Summary quantiles must use summary value MetricPoint values.
SUMMARY = 7;
}

// A single metric with a unique set of labels within a metric family.
message Metric {
    // Optional.
    repeated Label labels = 1;

    // Optional.
    repeated MetricPoint metric_points = 2;
}

// A name-value pair. These are used in multiple places: identifying
// timeseries, value of INFO metrics, and exemplars in Histograms.
message Label {
    // Required.
    string name = 1;

    // Required.
    string value = 2;
}

// A MetricPoint in a Metric.
message MetricPoint {
    // Required.
    oneof value {
        UnknownValue unknown_value = 1;
        GaugeValue gauge_value = 2;
        CounterValue counter_value = 3;
        HistogramValue histogram_value = 4;
        StateSetValue state_set_value = 5;
        InfoValue info_value = 6;
        SummaryValue summary_value = 7;
    }

    // Optional.
    google.protobuf.Timestamp timestamp = 8;
}

// Value for UNKNOWN MetricPoint.
message UnknownValue {
```

```
// Required.
oneof value {
    double double_value = 1;
    int64 int_value = 2;
}

// Value for GAUGE MetricPoint.
message GaugeValue {
    // Required.
    oneof value {
        double double_value = 1;
        int64 int_value = 2;
    }
}

// Value for COUNTER MetricPoint.
message CounterValue {
    // Required.
    oneof total {
        double double_value = 1;
        uint64 int_value = 2;
    }

    // The time values began being collected for this counter.
    // Optional.
    google.protobuf.Timestamp created = 3;

    // Optional.
    Exemplar exemplar = 4;
}

// Value for HISTOGRAM or GAUGE_HISTOGRAM MetricPoint.
message HistogramValue {
    // Optional.
    oneof sum {
        double double_value = 1;
        int64 int_value = 2;
    }

    // Optional.
    uint64 count = 3;

    // The time values began being collected for this histogram.
    // Optional.
    google.protobuf.Timestamp created = 4;

    // Optional.
```

```
    repeated Bucket buckets = 5;

    // Bucket is the number of values for a bucket in the histogram
    // with an optional exemplar.
    message Bucket {
        // Required.
        uint64 count = 1;

        // Optional.
        double upper_bound = 2;

        // Optional.
        Exemplar exemplar = 3;
    }
}

message Exemplar {
    // Required.
    double value = 1;

    // Optional.
    google.protobuf.Timestamp timestamp = 2;

    // Labels are additional information about the exemplar value
    // (e.g. trace id).
    // Optional.
    repeated Label label = 3;
}

// Value for STATE_SET MetricPoint.
message StateSetValue {
    // Optional.
    repeated State states = 1;

    message State {
        // Required.
        bool enabled = 1;

        // Required.
        string name = 2;
    }
}

// Value for INFO MetricPoint.
message InfoValue {
    // Optional.
    repeated Label info = 1;
}
```

```
// Value for SUMMARY MetricPoint.
message SummaryValue {
  // Optional.
  oneof sum {
    double double_value = 1;
    int64 int_value = 2;
  }

  // Optional.
  uint64 count = 2;

  // The time sum and count values began being collected for this
  // summary.
  // Optional.
  google.protobuf.Timestamp created = 3;

  // Optional.
  repeated Quantile quantile = 4;

  message Quantile {
    // Required.
    double quantile = 1;

    // Required.
    double value = 2;
  }
}
```

5. Design Considerations

5.1. Scope

OpenMetrics is intended to provide telemetry for online systems. It runs over protocols which do not provide hard or soft real time guarantees, so it can not make any real time guarantees itself. Latency and jitter properties of OpenMetrics are as imprecise as the underlying network, operating systems, CPUs, and the like. It is sufficiently accurate for aggregations to be used as a basis for decision-making, but not to reflect individual events.

Systems of all sizes should be supported, from applications that receive a few requests an hour up to monitoring bandwidth usage on a 400Gb network port. Aggregation and analysis of transmitted telemetry should be possible over arbitrary time periods.

It is intended to transport snapshots of state at the time of data transmission at a regular cadence.

5.1.1. Out of scope

How ingestors discover which exposers exist, and vice-versa, is out of scope for and thus not defined in this standard.

5.2. Extensions and Improvements

This first version of OpenMetrics is based upon well established and de facto standard Prometheus text format 0.0.4, deliberately without adding major syntactic or semantic extensions, or optimisations on top of it. For example no attempt has been made to make the text representation of Histogram buckets more compact, relying on compression in the underlying stack to deal with their repetitive nature.

This is a deliberate choice, so that the standard can take advantage of the adoption and momentum of the existing user base. This ensures a relatively easy transition from the Prometheus text format 0.0.4.

It also ensures that there is a basic standard which is easy to implement. This can be built upon in future versions of the standard. The intention is that future versions of the standard will always require support for this 1.0 version, both syntactically and semantically.

We want to allow monitoring systems to get usable information from an OpenMetrics exposition without undue burden. If one were to strip away all metadata and structure and just look at an OpenMetrics exposition as an unordered set of samples that should be usable on its own. As such, there are also no opaque binary types, such as sketches or t-digests which could not be expressed as a mix of gauges and counters as they would require custom parsing and handling.

This principle is applied consistently throughout the standard. For example a MetricFamily's unit is duplicated in the name so that the unit is available for systems that don't understand the unit metadata. The "le" label is a normal label value, rather than getting its own special syntax, so that ingestors don't have to add special histogram handling code to ingest them. As a further example, there are no composite data types. For example, there is no geolocation type for latitude/longitude as this can be done with separate gauge metrics.

5.3. Units and Base Units

For consistency across systems and to avoid confusion, units are largely based on SI base units. Base units include seconds, bytes,

joules, grams, meters, ratios, volts, amperes, and celsius. Units should be provided where they are applicable.

For example, having all duration metrics in seconds, there is no risk of having to guess whether a given metric is nanoseconds, microseconds, milliseconds, seconds, minutes, hours, days or weeks nor having to deal with mixed units. By choosing unprefix units, we avoid situations like ones in which kilomilliseconds were the result of emergent behaviour of complex systems.

As values can be floating point, sub-base-unit precision is built into the standard.

Similarly, mixing bits and bytes is confusing, so bytes are chosen as the base. While Kelvin is a better base unit in theory, in practice most existing hardware exposes Celsius. Kilograms are the SI base unit, however the kilo prefix is problematic so grams are chosen as the base unit.

While base units SHOULD be used in all possible cases, Kelvin is a well-established unit which MAY be used instead of Celsius for use cases such as color or black body temperatures where a comparison between a Celsius and Kelvin metric are unlikely.

Ratios are the base unit, not percentages. Where possible, raw data in the form of gauges or counters for the given numerator and denominator should be exposed. This has better mathematical properties for analysis and aggregation in the ingestors.

Decibels are not a base unit as firstly, deci is a SI prefix and secondly, bels are logarithmic. To expose signal/energy/power ratios exposing the ratio directly would be better, or better still the raw power/energy if possible. Floating point exponents are more than sufficient to cover even extreme scientific uses. An electron volt ($\sim 1\text{e-19 J}$) all the way up to the energy emitted by a supernova ($\sim 1\text{e44 J}$) is 63 orders of magnitude, and a 64-bit floating point number can cover over 2000 orders of magnitude.

If non-base units can not be avoided and conversion is not feasible, the actual unit should still be included in the metric name for clarity. For example, joule is the base unit for both energy and power, as watts can be expressed as a counter with a joule unit. In practice a given 3rd party system may only expose watts, so a gauge expressed in watts would be the only realistic choice in that case.

Not all MetricFamilies have units. For example a count of HTTP requests wouldn't have a unit. Technically the unit would be HTTP requests, but in that sense the entire MetricFamily name is the unit.

Going to that extreme would not be useful. The possibility of having good axes on graphs in downstream systems for human consumption should always be kept in mind.

5.4. Statelessness

The wire format defined by OpenMetrics is stateless across expositions. What information has been exposed before **MUST** have no impact on future expositions. Each exposition is a self-contained snapshot of the current state of the exposers.

The same self-contained exposition **MUST** be provided to existing and new ingestors.

A core design choice is that exposers **MUST NOT** exclude a metric merely because it has had no recent changes, or observations. An exposers must not make any assumptions about how often ingestors are consuming expositions.

5.5. Exposition Across Time and Metric Evolution

Metrics are most useful when their evolution over time can be analysed, so accordingly expositions must make sense over time. Thus, it is not sufficient for one single exposition on its own to be useful and valid. Some changes to metric semantics can also break downstream users.

Parsers commonly optimize by caching previous results. Thus, changing the order in which labels are exposed across expositions **SHOULD** be avoided even though it is technically not breaking. This also tends to make writing unit tests for exposition easier.

Metrics and samples **SHOULD NOT** appear and disappear from exposition to exposition, for example a counter is only useful if it has history. In principle, a given Metric should be present in exposition from when the process starts until the process terminates. It is often not possible to know in advance what Metrics a MetricFamily will have over the lifetime of a given process (e.g. a label value of a latency histogram is a HTTP path, which is provided by an end user at runtime), but once a counter-like Metric is exposed it should continue to be exposed until the process terminates. That a counter is not getting increments doesn't invalidate that it still has its current value. There are cases where it may make sense to stop exposing a given Metric; see the section on Missing Data.

In general changing a MetricFamily's type, or adding or removing a label from its Metrics will be breaking to ingestors.

A notable exception is that adding a label to the value of an Info MetricPoints is not breaking. This is so that you can add additional information to an existing Info MetricFamily where it makes sense to be, rather than being forced to create a brand new info metric with an additional label value. ingestor systems should ensure that they are resilient to such additions.

Changing a MetricFamily's Help is not breaking. For values where it is possible, switching between floats and ints is not breaking. Adding a new state to a stateset is not breaking. Adding unit metadata where it doesn't change the metric name is not breaking.

Histogram buckets SHOULD NOT change from exposition to exposition, as this is likely to both cause performance issues and break ingestors and cause. Similarly all expositions from any consistent binary and environment of an application SHOULD have the same buckets for a given Histogram MetricFamily, so that they can be aggregated by all ingestors without ingestors having to implement histogram merging logic for heterogeneous buckets. An exception might be occasional manual changes to buckets which are considered breaking, but may be a valid tradeoff when performance characteristics change due to a new software release.

Even if changes are not technically breaking, they still carry a cost. For example frequent changes may cause performance issues for ingestors. A Help string that varies from exposition to exposition may cause each Help value to be stored. Frequently switching between int and float values could prevent efficient compression.

5.6. NaN

NaN is a number like any other in OpenMetrics, usually resulting from a division by zero such as for a summary quantile if there have been no observations recently. NaN does not have any special meaning in OpenMetrics, and in particular MUST NOT be used as a marker for missing or otherwise bad data.

5.7. Missing Data

There are valid cases when data stops being present. For example a filesystem can be unmounted and thus its Gauge Metric for free disk space no longer exists. There is no special marker or signal for this situation. Subsequent expositions simply do not include this Metric.

5.8. Exposition Performance

Metrics are only useful if they can be collected in reasonable time frames. Metrics that take minutes to expose are not considered useful.

As a rule of thumb, exposition SHOULD take no more than a second.

Metrics from legacy systems serialized through OpenMetrics may take longer. For this reason, no hard performance assumptions can be made.

Exposition SHOULD be of the most recent state. For example, a thread serving the exposition request SHOULD NOT rely on cached values, to the extent it is able to bypass any such caching

5.9. Concurrency

For high availability and ad-hoc access a common approach is to have multiple ingestors. To support this, concurrent expositions MUST be supported. All BCPs for concurrent systems SHOULD be followed, common pitfalls include deadlocks, race conditions, and overly-coarse grained locking preventing expositions progressing concurrently.

5.10. Metric Naming and Namespaces

EDITOR'S NOTE: This section might be good for a BCP paper.

We aim for a balance between understandability, avoiding clashes, and succinctness in the naming of metrics and label names. Names are separated through underscores, so metric names end up being in "snake_case".

To take an example "http_request_seconds" is succinct but would clash between large numbers of applications, and it's also unclear exactly what this metric is measuring. For example, it might be before or after auth middleware in a complex system.

Metric names should indicate what piece of code they come from. So a company called A Company Manufacturing Everything might prefix all metrics in their code with "acme_", and if they had a HTTP router library measuring latency it might have a metric such as "acme_http_router_request_seconds" with a Help string indicating that it is the overall latency.

It is not the aim to prevent all potential clashes across all applications, as that would require heavy handed solutions such as a global registry of metric namespaces or very long namespaces based on

DNS. Rather the aim is to keep to a lightweight informal approach, so that for a given application that it is very unlikely that there is clash across its constituent libraries.

Across a given deployment of a monitoring system as a whole the aim is that clashes where the same metric name means different things are uncommon. For example `acme_http_router_request_seconds` might end up in hundreds of different applications developed by A Company Manufacturing Everything, which is normal. If Another Corporation Making Entities also used the metric name `acme_http_router_request_seconds` in their HTTP router that's also fine. If applications from both companies were being monitored by the same monitoring system the clash is undesirable, but acceptable as no application is trying to expose both names and no one target is trying to (incorrectly) expose the same metric name twice. If an application wished to contain both My Example Company's and Mega Exciting Company's HTTP router libraries that would be a problem, and one of the metric names would need to be changed somehow.

As a corollary, the more public a library is the better namespaced its metric names should be to reduce the risk of such scenarios arising. `acme_` is not a bad choice for internal use within a company, but these companies might for example choose the prefixes `acmeeverything_` or `acorpme_` for code shared outside their company.

After namespacing by company or organisation, namespacing and naming should continue by library/subsystem/application fractally as needed such as the `http_router` library above. The goal is that if you are familiar with the overall structure of a codebase, you could make a good guess at where the instrumentation for a given metric is given its metric name.

For a common very well known existing piece of software, the name of the software itself may be sufficiently distinguishing. For example `bind_` is probably sufficient for the DNS software, even though `isc_bind_` would be the more usual naming.

Metric names prefixed by `scrape_` are used by ingestors to attach information related to individual expositions, so should not be exposed by applications directly. Metrics that have already been consumed and passed through a general purpose monitoring system may include such metric names on subsequent expositions. If an exposor wishes to provide information about an individual exposition, a metric prefix such as `myexposer_scrape_` may be used. A common example is a gauge `myexposer_scrape_duration_seconds` for how long that exposition took from the exposor's standpoint.

Within the Prometheus ecosystem a set of per-process metrics has emerged that are consistent across all implementations, prefixed with `process_`. For example for open file ulimits the MetricFamilies `process_open_fds` and `process_max_fds` gauges provide both the current and maximum value. (These names are legacy, if such metrics were defined today they would be more likely called `process_fds_open` and `process_fds_limit`). In general it is very challenging to get names with identical semantics like this, which is why different instrumentation should use different names.

Avoid redundancy in metric names. Avoid substrings like "metric", "timer", "stats", "counter", "total", "float64" and so on - by virtue of being a metric with a given type (and possibly unit) exposed via OpenMetrics information like this is already implied so should not be included explicitly. You should not include label names of a metric in the metric name for the same reasons, and in addition subsequent aggregation of the metric by a monitoring system could make such information incorrect.

Avoid including implementation details from other layers of your monitoring system in the metric names contained in your instrumentation. For example a MetricFamily name should not contain the string "openmetrics" merely because it happens to be currently exposed via OpenMetrics somewhere, or "prometheus" merely because your current monitoring system is Prometheus.

5.11. Label Namespacing

For label names no explicit namespacing by company or library is recommended, namespacing from the metric name is sufficient for this when considered against the length increase of the label name. However some minimal care to avoid common clashes is recommended.

There are label names such as `region`, `zone`, `cluster`, `availability_zone`, `az`, `datacenter`, `dc`, `owner`, `customer`, `stage`, `service`, `team`, `job`, `instance`, `environment`, and `env` which are highly likely to clash with labels used to identify targets which a general purpose monitoring system may add. Try to avoid them, adding minimal namespacing may be appropriate in these cases.

The label name "type" is highly generic and should be avoided. For example for HTTP-related metrics "method" would be a better label name if you were distinguishing between GET, POST, and PUT requests.

While there is metadata about metric names such as HELP, TYPE and UNIT there is no metadata for label names. This is as it would be bloating the format for little gain. Out-of-band documentation is one way for exposers could present this their ingestors.

5.12. Metric Names versus Labels

There are situations in which both using multiple Metrics within a MetricFamily or multiple MetricFamilies seem to make sense. Summing or averaging aMetricFamily should be meaningful even if it's not always useful. For example, mixing voltage and fan speed is not meaningful.

As a reminder, OpenMetrics is built with the assumption that ingestors can process and perform aggregations on data.

Exposing a total sum alongside other metrics is wrong, as this would result in double-counting upon aggregation in downstream ingestors.

```
~~~~ wrong_metric{label="a"} 1 wrong_metric{label="b"} 6
wrong_metric{label="total"} 7 ~~~~
```

Labels of a Metric should be to the minimum needed to ensure uniqueness as every extra label is one more that users need to consider when determining what Labels to work with downstream. Labels which could be applied many MetricFamilies are candidates for being moved into _info metrics similar to database [normalization]. If virtually all users of a Metric could be expected to want the additional label, it may be a better trade-off to add it to all MetricFamilies. For example if you had a MetricFamily relating to different SQL statements where uniqueness was provided by a label containing a hash of the full SQL statements, it would be okay to have another label with the first 500 characters of the SQL statement for human readability.

Experience has shown that downstream ingestors find it easier to work with separate total and failure MetricFamilies rather than using {result="success"} and {result="failure"} Labels within one MetricFamily. Also it is usually better to expose separate read & write and send & receive MetricFamilies as full duplex systems are common and downstream ingestors are more likely to care about those values separately than in aggregate.

All of this is not as easy as it may sound. It's an area where experience and engineering trade-offs by domain-specific experts in both exposition and the exposed system are required to find a good balance. Metric and Label Name Characters

OpenMetrics builds on the existing widely adopted Prometheus text exposition format and the ecosystem which formed around it. Backwards compatibility is a core design goal. Expanding or contracting the set of characters that are supported by the Prometheus text format would work against that goal. Breaking backwards compatibility would have wider implications than just the

wire format. In particular, the query languages created or adopted to work with data transmitted within the Prometheus ecosystem rely on these precise character sets. Label values support full UTF-8, so the format can represent multi-lingual metrics.

5.13. Types of Metadata

Metadata can come from different sources. Over the years, two main sources have emerged. While they are often functionally the same, it helps in understanding to talk about their conceptual differences.

"Target metadata" is metadata commonly external to an exposer. Common examples would be data coming from service discovery, a CMDB, or similar, like information about a datacenter region, if a service is part of a particular deployment, or production or testing. This can be achieved by either the exposer or the ingestor adding labels to all Metrics that capture this metadata. Doing this through the ingestor is preferred as it is more flexible and carries less overhead. On flexibility, the hardware maintenance team might care about which server rack a machine is located in, whereas the database team using that same machine might care that it contains replica number 2 of the production database. On overhead, hardcoding or configuring this information needs an additional distribution path.

"Exposer metadata" is coming from within an exposer. Common examples would be software version, compiler version, or Git commit SHA.

5.13.1. Supporting Target Metadata in both Push-based and Pull-based Systems

In push-based consumption, it is typical for the exposer to provide the relevant target metadata to the ingestor. In pull-based consumption the push-based approach could be taken, but more typically the ingestor already knows the metadata of the target a-priori such as from a machine database or service discovery system, and associates it with the metrics as it consumes the exposition.

OpenMetrics is stateless and provides the same exposition to all ingestors, which is in conflict with the push-style approach. In addition the push-style approach would break pull-style ingestors, as unwanted metadata would be exposed.

One approach would be for push-style ingestors to provide target metadata based on operator configuration out-of-band, for example as a HTTP header. While this would transport target metadata for push-style ingestors, and is not precluded by this standard, it has the disadvantage that even though pull-style ingestors should use their

own target metadata, it is still often useful to have access to the metadata the exposers itself is aware of.

The preferred solution is to provide this target metadata as part of the exposition, but in a way that does not impact on the exposition as a whole. Info MetricFamilies are designed for this. An exposers may include an Info MetricFamily called "target" with a single Metric with no labels with the metadata. An example in the text format might be: ~~~~ # TYPE target info # HELP target Target metadata target_info{env="prod",hostname="myhost",datacenter="sdc",region="europe",owner="frontend"} 1 ~~~~

When an exposers is providing this metric for this purpose it SHOULD be first in the exposition. This is for efficiency, so that ingestors relying on it for target metadata don't have to buffer up the rest of the exposition before applying business logic based on its content.

Exposers MUST NOT add target metadata labels to all Metrics from an exposition, unless explicitly configured for a specific ingestor. Exposers MUST NOT prefix MetricFamily names or otherwise vary MetricFamily names based on target metadata. Generally, the same Label should not appear on every Metric of an exposition, but there are rare cases where this can be the result of emergent behaviour. Similarly all MetricFamily names from an exposers may happen to share a prefix in very small expositions. For example an application written in the Go language by A Company Manufacturing Everything would likely include metrics with prefixes of acme_, go_, process_, and metric prefixes from any 3rd party libraries in use.

Exposers can expose exposers metadata as Info MetricFamilies.

The above discussion is in the context of individual exposers. An exposition from a general purpose monitoring system may contain metrics from many individual targets, and thus may expose multiple target info Metrics. The metrics may already have had target metadata added to them as labels as part of ingestion. The metric names MUST NOT be varied based on target metadata. For example it would be incorrect for all metrics to end up being prefixed with staging_ even if they all originated from targets in a staging environment).

5.14. Client Calculations and Derived Metrics

Exposers should leave any math or calculation up to ingestors. A notable exception is the Summary quantile which is unfortunately required for backwards compatibility. Exposition should be of raw values which are useful over arbitrary time periods.

As an example, you should not expose a gauge with the average rate of increase of a counter over the last 5 minutes. Letting the ingestor calculate the increase over the data points they have consumed across expositions has better mathematical properties and is more resilient to scrape failures.

Another example is the average event size of a histogram/summary. Exposing the average rate of increase of a counter since an application started or since a Metric was created has the problems from the earlier example and it also prevents aggregation.

Standard deviation also falls into this category. Exposing a sum of squares as a counter would be the correct approach. It was not included in this standard as a Histogram value because 64bit floating point precision is not sufficient for this to work in practice. Due to the squaring only half the 53bit mantissa would be available in terms of precision. As an example a histogram observing 10k events per second would lose precision within 2 hours. Using 64bit integers would be no better due to the loss of the floating decimal point because a nanosecond resolution integer typically tracking events of a second in length would overflow after 19 observations. This design decision can be revisited when 128bit floating point numbers become common.

Another example is to avoid exposing a request failure ratio, exposing separate counters for failed requests and total requests instead.

5.15. Number Types

For a counter that was incremented a million times per second it would take over a century to begin to lose precision with a float64 as it has a 53 bit mantissa. Yet a 100 Gbps network interface's octet throughput precision could begin to be lost with a float64 within around 20 hours. While losing 1KB of precision over the course of years for a 100Gbps network interface is unlikely to be a problem in practice, int64s are an option for integral data with such a high throughput.

Summary quantiles must be float64, as they are estimates and thus fundamentally inaccurate.

5.16. Exposing Timestamps

One of the core assumptions of OpenMetrics is that exposers expose the most up to date snapshot of what they're exposing.

While there are limited use cases for attaching timestamps to exposed data, these are very uncommon. Data which had timestamps previously attached, in particular data which has been ingested into a general purpose monitoring system may carry timestamps. Live or raw data should not carry timestamps. It is valid to expose the same metric MetricPoint value with the same timestamp across expositions, however it is invalid to do so if the underlying metric is now missing.

Time synchronization is a hard problem and data should be internally consistent in each system. As such, ingestors should be able to attach the current timestamp from their perspective to data rather than based on the system time of the exposer device.

With timestamped metrics it is not generally possible to detect the time when a Metric went missing across expositions. However with non-timestamped metrics the ingestor can use its own timestamp from the exposition where the Metric is no longer present.

All of this is to say that, in general, MetricPoint timestamps should not be exposed, as it should be up to the ingestor to apply their own timestamps to samples they ingest.

5.16.1. Tracking When Metrics Last Changed

Presume you had a counter `my_counter` which was initialized, and then later incremented by 1 at time 123. This would be a correct way to expose it in the text format: `~~~~ # HELP my_counter Good increment example # TYPE my_counter counter my_counter_total 1 ~~~~` As per the parent section, ingestors should be free to attach their own timestamps, so this would be incorrect: `~~~~ # HELP my_counter Bad increment example # TYPE my_counter counter my_counter_total 1 123 ~~~~`

In case the specific time of the last change of a counter matters, this would be the correct way: `~~~~ # HELP my_counter Good increment example # TYPE my_counter counter my_counter_total 1 # HELP my_counter_last_increment_timestamp_seconds When my_counter was last incremented # TYPE my_counter_last_increment_timestamp_seconds gauge # UNIT my_counter_last_increment_timestamp_seconds seconds my_counter_last_increment_timestamp_seconds 123 ~~~~`

By putting the timestamp of last change into its own Gauge as a value, ingestors are free to attach their own timestamp to both Metrics.

Experience has shown that exposing absolute timestamps (epoch is considered absolute here) is more robust than time elapsed, seconds since, or similar. In either case, they would be gauges. For

```

example ~~~~ # TYPE my_boot_time_seconds gauge # HELP
my_boot_time_seconds Boot time of the machine # UNIT
my_boot_time_seconds seconds my_boot_time_seconds 1256060124 ~~~~

Is better than ~~~~ # TYPE my_time_since_boot_seconds gauge # HELP
my_time_since_boot_seconds Time elapsed since machine booted # UNIT
my_time_since_boot_seconds seconds my_time_since_boot_seconds 123
~~~~

```

Conversely, there are no best practice restrictions on exemplars timestamps. Keep in mind that due to race conditions or time not being perfectly synced across devices, that an exemplar timestamp may appear to be slightly in the future relative to a ingestor's system clock or other metrics from the same exposition. Similarly it is possible that a "_created" for a MetricPoint could appear to be slightly after an exemplar or sample timestamp for that same MetricPoint.

Keep in mind that there are monitoring systems in common use which support everything from nanosecond to second resolution, so having two MetricPoints that have the same timestamp when truncated to second resolution may cause an apparent duplicate in the ingestor. In this case the MetricPoint with the earliest timestamp MUST be used.

5.17. Thresholds

Exposing desired bounds for a system can make sense, but proper care needs to be taken. For values which are universally true, it can make sense to emit Gauge metrics for such thresholds. For example, a data center HVAC system knows the current measurements, the setpoints, and the alert setpoints. It has a globally valid and correct view of the desired system state. As a counter example, some thresholds can change with scale, deployment model, or over time. A certain amount of CPU usage may be acceptable in one setting and undesirable in another. Aggregation of values can further change acceptable values. In such a system, exposing bounds could be counter-productive.

For example a the maximum size of a queue may be exposed alongside the number of items currently in the queue like: ~~~~ # HELP
 acme_notifications_queue_capacity The capacity of the notifications queue. # TYPE acme_notifications_queue_capacity gauge
 acme_notifications_queue_capacity 10000 # HELP
 acme_notifications_queue_length The number of notifications in the queue. # TYPE acme_notifications_queue_length gauge
 acme_notifications_queue_length 42 ~~~~

5.18. Size Limits

This standard does not prescribe any particular limits on the number of samples exposed by a single exposition, the number of labels that may be present, the number of states a stateset may have, the number of labels in an info value, or metric name/label name/label value/help character limits.

Specific limits run the risk of preventing reasonable use cases, for example while a given exposition may have an appropriate number of labels after passing through a general purpose monitoring system a few target labels may have been added that would push it over the limit. Specific limits on numbers such as these would also not capture where the real costs are for general purpose monitoring systems. These guidelines are thus both to aid exposers and ingestors in understanding what is reasonable.

On the other hand, an exposition which is too large in some dimension could cause significant performance problems compared to the benefit of the metrics exposed. Thus some guidelines on the size of any single exposition would be useful.

ingestors may choose to impose limits themselves, for in particular to prevent attacks or outages. Still, ingestors need to consider reasonable use cases and try not to disproportionately impact them. If any single value/metric/exposition exceeds such limits then the whole exposition must be rejected.

In general there are three things which impact the performance of a general purpose monitoring system ingestion time series data: the number of unique time series, the number of samples over time in those series, and the number of unique strings such as metric names, label names, label values, and HELP. ingestors can control how often they ingest, so that aspect does not need further consideration.

The number of unique time series is roughly equivalent to the number of non-comment lines in the text format. As of 2020, 10 million time series in total is considered a large amount and is commonly the order of magnitude of the upper bound of any single-instance ingestor. Any single exposition should not go above 10k time series without due diligence. One common consideration is horizontal scaling: What happens if you scale your instance count by 1-2 orders of magnitude? Having a thousand top-of-rack switches in a single deployment would have been hard to imagine 30 years ago. If a target was a singleton (e.g. exposing metrics relating to an entire cluster) then several hundred thousand time series may be reasonable. It is not the number of unique MetricFamilies or the cardinality of individual labels/buckets/statesets that matters, it is the total

order of magnitude of the time series. 1,000 gauges with one Metric each are as costly as a single gauge with 1,000 Metrics.

If all targets of a particular type are exposing the same set of time series, then each additional targets' strings poses no incremental cost to most reasonably modern monitoring systems. If however each target has unique strings, there is such a cost. As an extreme example, a single 10k character metric name used by many targets is on its own very unlikely to be a problem in practice. To the contrary, a thousand targets each exposing a unique 36 character UUID is over three times as expensive as that single 10k character metric name in terms of strings to be stored assuming modern approaches. In addition, if these strings change over time older strings will still need to be stored for at least some time, incurring extra cost. Assuming the 10 million time series from the last paragraph, 100MB of unique strings per hour might indicate a use case for then the use case may be more like event logging, not metric time series.

There is a hard 128 UTF-8 character limit on exemplar length, to prevent misuse of the feature for tracing span data and other event logging.

6. Security Considerations

Implementors MAY choose to offer authentication, authorization, and accounting; if they so choose, this SHOULD be handled outside of OpenMetrics.

All exposers implementations SHOULD be able to secure their HTTP traffic with TLS 1.2 or later. If an exposers implementation does not support encryption, operators SHOULD use reverse proxies, firewalling, and/or ACLs where feasible.

Metric exposition should be independent of production services exposed to end users; as such, having a /metrics endpoint on ports like TCP/80, TCP/443, TCP/8080, and TCP/8443 is generally discouraged for publicly exposed services using OpenMetrics.

7. IANA Considerations

While currently most implementations of the Prometheus exposition format are using non-IANA-registered ports from an informal registry at [PrometheusPorts], OpenMetrics can be found on a well-defined port.

The port assigned by IANA for clients exposing data is <9099 requested for historical consistency>.

If more than one metric endpoint needs to be reachable at a common IP address and port, operators might consider using a reverse proxy that communicates with exposers over localhost addresses. To ease multiplexing, endpoints SHOULD carry their own name in their path, i.e. `"/node_exporter/metrics"`. Expositions SHOULD NOT be combined into one exposition, for the reasons covered under "Supporting target metadata in both push-based and pull-based systems" and to allow for independent ingestion without a single point of failure.

OpenMetrics would like to register two MIME types, `"application/openmetrics-text"` and `"application/openmetrics-proto"`.

EDITOR'S NOTE: `"application/openmetrics-text"` is in active use since 2018, `"application/openmetrics-proto"` is not yet in active use.

EDITOR'S NOTE: We would like to thank Sumeer Bhola, but kramdown 2.x does not support "Contributor:" any more so we will add this by hand once consensus has been achieved.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [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>>.

8.2. Informative References

- [normalization] "Database Normalization", n.d., <https://en.wikipedia.org/wiki/Database_normalization>.
- [PrometheusPorts] "Prometheus informal port allocation", n.d., <<https://github.com/prometheus/prometheus/wiki/Default-port-allocations>>.

[timestamp]
"Go Timestamp ProtoBuf", n.d., <<https://github.com/protocolbuffers/protobuf/blob/2f6a7546e4539499bc08abc6900dc929782f5dcd/src/google/protobuf/timestamp.proto>>.

Authors' Addresses

Richard Hartmann (editor)
Grafana Labs

Email: richih@richih.org

Ben Kochie
GitLab

Email: superq@gmail.com

Brian Brazil
Robust Perception

Email: brian.brazil@gmail.com

Rob Skillington
Chronosphere

Email: rob.skillington@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 25, 2021

T. Graf
Swisscom
March 24, 2021

Export of MPLS Segment Routing Label Type Information in
IP Flow Information Export (IPFIX)
draft-tgraf-ipfix-mpls-sr-label-type-07

Abstract

This document introduces additional code points in the mplsTopLabelType Information Element for IS-IS, OSPFv2, OSPFv3 and BGP MPLS Segment Routing (SR) extensions to enable Segment Routing label protocol type information in IP Flow Information Export (IPFIX).

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 September 25, 2021.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
2. MPLS Segment Routing Top Label Type	2
3. IANA Considerations	3
4. Security Considerations	4
5. Acknowledgements	4
6. References	4
Author's Address	6

1. Introduction

Besides BGP-4 [RFC8277], LDP [RFC5036] and BGP VPN [RFC4364], four new routing-protocols, OSPFv2 Extensions [RFC8665], OSPFv3 Extensions [RFC8666], IS-IS Extensions [RFC8667] and BGP Prefix-SID [RFC8669] have been added to the list of routing-protocols able to propagate Segment Routing labels for the MPLS data plane [RFC8660].

Traffic Accounting in Segment Routing Networks [I-D.ali-spring-sr-traffic-accounting] describes how IPFIX can be leveraged to account traffic to MPLS Segment Routing label dimensions within a Segment Routing domain.

In the Information Model for IP Flow Information Export IPFIX [RFC7012], the information element `mplsTopLabelType(46)` describes which MPLS control plane protocol allocated the top-of-stack label in the MPLS label stack. RFC 7012 section 7.2 [RFC7012] describes the "IPFIX MPLS label type (Value 46)" sub-registry [IANA-IPFIX-IE46] where new code points should be added.

2. MPLS Segment Routing Top Label Type

By introducing four new code points to information element `mplsTopLabelType(46)` for IS-IS, OSPFv2, OSPFv3 and BGP Prefix-SID, when Segment Routing with one of these four routing protocols is deployed, we get insight into which traffic is being forwarded based on which MPLS control plane protocol.

A typical use case scenario is to monitor MPLS control plane migrations from LDP to IS-IS or OSPF Segment Routing. Such a migration can be done node by node as described in RFC8661 [RFC8661]

Another use case is the monitoring of a migration to a Seamless MPLS SR [I-D.hegde-spring-mpls-seamless-sr] architecture where prefixes are propagated with dynamic BGP labels according to RFC8277

[RFC8277], BGP Prefix-SID according to RFC8669 [RFC8669] and used for the forwarding between IGP domains. Adding an additional layer into the MPLS data plane to above described use case.

Both use cases can be verified by using `mplsTopLabelType(46)`, `mplsTopLabelIPv4Address(47)`, `mplsTopLabelStackSection(70)` and `forwardingStatus(89)` dimensions to get insights into

- o how many packets are forwarded or dropped
- o if dropped, for which reasons
- o the MPLS provider edge loopback address and label protocol

By looking at the MPLS label value itself, it is not always clear as to which label protocol it belongs, since they could potentially share the same label allocation range. This is the case for IGP-Adjacency SID's, LDP and dynamic BGP labels as an example.

3. IANA Considerations

This document specifies four additional code points for IS-IS, OSPFv2, OSPFv3 and BGP Prefix-SID Segment Routing extension in the existing sub-registry "IPFIX MPLS label type (Value 46)" of the "IPFIX Information Elements" and one new "IPFIX Information Element" with a new sub-registry in the "IP Flow Information Export (IPFIX) Entities" name space.

Value	Description	Reference	Requester
TBD1	OSPFv2 Segment Routing	RFC8665	[RFC-to-be]
TBD2	OSPFv3 Segment Routing	RFC8666	[RFC-to-be]
TBD3	IS-IS Segment Routing	RFC8667	[RFC-to-be]
TBD4	BGP Segment Routing Prefix-SID	RFC8669	[RFC-to-be]

Figure 1: Updates to "IPFIX MPLS label type (Value 46)" SubRegistry

Note to IANA:

- o Please assign TBD1 to 4 to the next available numbers according to the "IPFIX MPLS label type (Value 46)" sub-registry [IANA-IPFIX-IE46] procedure.

- o Please replace the [RFC-to-be] with the RFC number assigned to this document.

Note to RFC-editor:

- o Please remove above two IANA notes.

4. Security Considerations

There exists no extra security considerations regarding the allocation of these new IPFIX information elements compared to RFC7012 [RFC7012].

5. Acknowledgements

I would like to thank to the IE doctors, Paul Aitken and Andrew Feren, as well Benoit Claise, Loa Andersson, Tianran Zhou, Pierre Francois, Bruno Decreane, Paolo Lucente, Hannes Gredler, Ketan Talaulikar, Sabrina Tanamal, Erik Auerswald and Sergey Fomin for their review and valuable comments.

6. References

6.1. Normative References

[RFC7012] Claise, B., Ed. and B. Trammell, Ed., "Information Model for IP Flow Information Export (IPFIX)", RFC 7012, DOI 10.17487/RFC7012, September 2013, <<https://www.rfc-editor.org/info/rfc7012>>.

6.2. Informative References

[I-D.ali-spring-sr-traffic-accounting]
Filsfils, C., Talaulikar, K., Sivabalan, S., Horneffer, M., Raszuk, R., Litkowski, S., Voyer, D., and R. Morton, "Traffic Accounting in Segment Routing Networks", draft-ali-spring-sr-traffic-accounting-04 (work in progress), February 2020.

[I-D.hegde-spring-mpls-seamless-sr]
Hegde, S., Bowers, C., Xu, X., Gulko, A., Bogdanov, A., Uttaro, J., Jalil, L., Khaddam, M., and A. Alston, "Seamless Segment Routing", draft-hegde-spring-mpls-seamless-sr-04 (work in progress), January 2021.

- [IANA-IPFIX-IE46] "IANA IP Flow Information Export (IPFIX) Information Element #46 SubRegistry",
 <<https://www.iana.org/assignments/ipfix/ipfix.xhtml#ipfix-mpls-label-type>>.
- [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>>.
- [RFC5036] Andersson, L., Ed., Minei, I., Ed., and B. Thomas, Ed., "LDP Specification", RFC 5036, DOI 10.17487/RFC5036, October 2007, <<https://www.rfc-editor.org/info/rfc5036>>.
- [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>>.
- [RFC8660] Bashandy, A., Ed., Filsfils, C., Ed., Previdi, S., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing with the MPLS Data Plane", RFC 8660, DOI 10.17487/RFC8660, December 2019, <<https://www.rfc-editor.org/info/rfc8660>>.
- [RFC8661] Bashandy, A., Ed., Filsfils, C., Ed., Previdi, S., Decraene, B., and S. Litkowski, "Segment Routing MPLS Interworking with LDP", RFC 8661, DOI 10.17487/RFC8661, December 2019, <<https://www.rfc-editor.org/info/rfc8661>>.
- [RFC8665] Psenak, P., Ed., Previdi, S., Ed., Filsfils, C., Gredler, H., Shakir, R., Henderickx, W., and J. Tantsura, "OSPF Extensions for Segment Routing", RFC 8665, DOI 10.17487/RFC8665, December 2019, <<https://www.rfc-editor.org/info/rfc8665>>.
- [RFC8666] Psenak, P., Ed. and S. Previdi, Ed., "OSPFv3 Extensions for Segment Routing", RFC 8666, DOI 10.17487/RFC8666, December 2019, <<https://www.rfc-editor.org/info/rfc8666>>.
- [RFC8667] Previdi, S., Ed., Ginsberg, L., Ed., Filsfils, C., Bashandy, A., Gredler, H., and B. Decraene, "IS-IS Extensions for Segment Routing", RFC 8667, DOI 10.17487/RFC8667, December 2019, <<https://www.rfc-editor.org/info/rfc8667>>.

[RFC8669] Previdi, S., Filsfils, C., Lindem, A., Ed., Sreekantiah,
 A., and H. Gredler, "Segment Routing Prefix Segment
 Identifier Extensions for BGP", RFC 8669,
 DOI 10.17487/RFC8669, December 2019,
 <<https://www.rfc-editor.org/info/rfc8669>>.

Author's Address

Thomas Graf
Swisscom
Binzring 17
Zurich 8045
Switzerland

Email: thomas.graf@swisscom.com