Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 25, 2016

A. Lindem, Ed.
Cisco Systems
L. Berger, Ed.
LabN Consulting, L.L.C.
D. Bogdanovic

C. Hopps
Deutsche Telekom

January 22, 2016

Network Device YANG Organizational Models
draft-rtgyangdt-rtgwg-device-model-02

Abstract

This document presents an approach for organizing YANG models in a comprehensive structure that may be used to configure and operate network devices. The structure is itself represented as a YANG model, with all of the related component models logically organized in a way that is operationally intuitive, but this model is not expected to be implemented. The identified component modules are expected to be defined and implemented on common network devices.

This document also defines two modules that can be used to model the logical and virtual resource representations that may be present on a network device. Examples of common industry terms for logical resource representations are Logical Systems or Routers. Examples of of common industry terms for virtual resource representations are Virtual Routing and Forwarding (VRF) instances and Virtual Switch Instances (VSIs).

This document is derived from work submitted to the IETF by members of the informal OpenConfig working group of network operators and is a product of the Routing Area YANG Architecture design team.

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 http://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 July 25, 2016.

Table of Contents

1. Introduction

"Operational Structure and Organization of YANG Models" [OC-STRUCT], highlights the value of organizing individual, self-standing YANG [RFC6020] models into a more comprehensive structure. This document builds on that work and presents a derivative structure for use in representing the networking infrastructure aspects of physical and virtual devices. [OC-STRUCT] and earlier versions of this document presented a single device-centric model root, this document no longer contains this element. Such an element would have translated to a single device management model that would be the root of all other models and was judged to be overly restrictive in terms of definition, implementation, and operation.

The document first presents a notional network device YANG organizational structure that provides a conceptual framework for the models that may be used to configure and operate network devices. The structure is itself presented as a YANG module, with all of the related component modules logically organized in a way that is operationally intuitive. This network device model is not expected to be implemented, but rather provide as context for the identified representative component modules with are expected to be defined, and supported on typical network devices.

This document also defines two new modules that are expected to be implemented. These models are defined to support the configuration and operation of network-devices that allow for the partitioning of resources from both, or either, management and networking perspectives. Both make use of emerging YANG functionality supported by either structural mount [STRUCTURAL-MOUNT] or YANG Schema Dispatching Language (YSDL) [YANG-YSDL]. The module definitions in this version of the document match [STRUCTURAL-MOUNT] but the solution adopted by the Netmod Working Group will be used.

Two forms of resource partitioning are supported:

The first form provides a logical partitioning of a network device where each partition is separately managed as essentially an independent network element which is 'hosted' by the base network device. These hosted network elements are referred to as logical network elements, or LNEs, and are supported by the logical-network-element module defined below. The module is used to identify LNEs and associate resources from the network-device with each LNE. LNEs themselves are represented in YANG as independent network devices;

each accessed independently. Optionally, and when supported by the implementation, they may also be accessed from the host system. Examples of vendor terminology for an LNE include logical system or router, and virtual switch, chassis, or fabric.

The second form provides support what is commonly referred to as Virtual Routing and Forwarding (VRF) instances as well as Virtual Switch Instances (VSI), see [RFC4026]. In this form of resource partitioning multiple control plane and forwarding/bridging instances are provided by and managed via a single (physical or logical) network device. This form of resource partitioning is referred to as Network Instances and are supported by the networking-instance module defined below. Configuration and operation of each network-instance is always via the network device and the networking-instance module.

This document was motivated by, and derived from, [OC-STRUCT]. The requirements from that document have been combined with the requirements from "Consistent Modeling of Operational State Data in YANG", [OC-OPSTATE], into "NETMOD Operational State Requirements", [NETMOD-OPSTATE]. This document is aimed at the requirement related to a common model-structure, currently Requirement 7, and also aims to provide a modeling base for Operational State representation.

The approach taken in this (and the original) document is to organize the models describing various aspects of network infrastructure, focusing on devices, their subsystems, and relevant protocols operating at the link and network layers. The proposal does not consider a common model for higher level network services. We focus on the set of models that are commonly used by network operators, and suggest a corresponding organization.

A significant portion of the text and model contained in this document was taken from the -00 of [OC-STRUCT].

1.1. Status of Work and Open Issues

This version of the document and structure are a product of the Routing Area YANG Architecture design team and is very much a work in progress rather than a final proposal. This version is a major change from the prior version and this change was enabled by the work on the previously mentioned Structural Mount/YSDL.

Structural Mount/YSDL enables a dramatic simplification of the presented device model, particularly for "lower-end" devices which are unlikely to support multiple network instances or logical network elements. Should structural-mount/YSDL not be available, the more explicit tree structure presented in earlier versions of this document will need to be utilized.

The top open issues are:

1. The use of YSDL vs Structural Mount needs to be resolved as does ensuring that the selected approach has the needed capabilities.

2. This document will need to match the evolution and standardization of [OC-OPSTATE] or [NETMOD-OPSTATE] by the Netmod WG.

3. Interpretation of different policy containers requires clarification.

4. It may make sense to use the identityref structuring with hardware and QoS model.

5. Which document(s) define the base System management, networking services, and oam protocols modules is TBD. This includes the possibility of simply using RFC7317 in place of the presented System management module.

6. The model will be updated once the "opstate" requirements are addressed.

7. It may make sense to publish the networking-instance and logical-network-element modules separately, as different devices may not implement both, and network providers may only require one or the other.

2. Module Overview

In this document, we consider network devices that support protocols and functions defined within the IETF Routing Area, e.g, routers, firewalls and hosts. Such devices may be physical or virtual, e.g., a classic router with custom hardware or one residing within a server-based virtual machine implementing a virtual network function (VNF). Each device may sub-divide their resources into logical network elements (LNEs) each of which provides a managed logical device. Examples of vendor terminology for an LNE include logical system or router, and virtual switch, chassis, or fabric. Each LNE may also support virtual routing and forwarding (VRF) and virtual switching instance (VSI) functions, which are referred to below as a networking instances (NIs). This breakdown is represented in Figure 1.

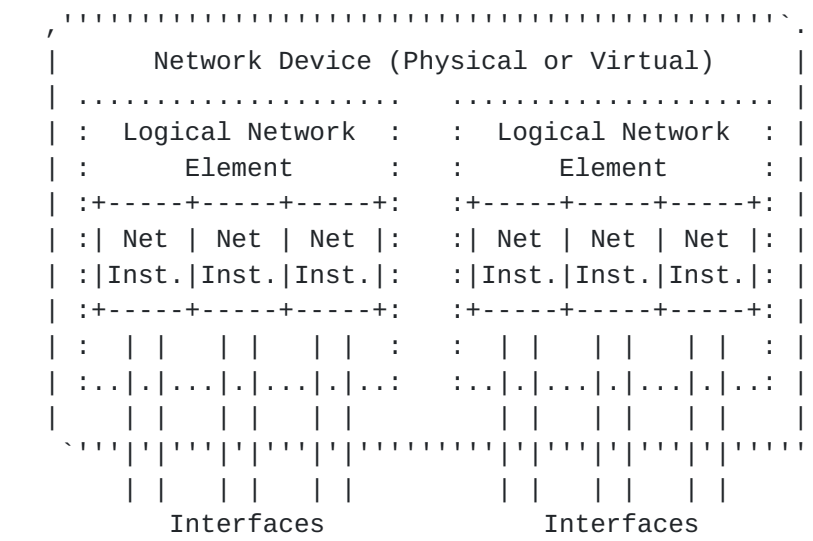

Figure 1: Module Element Relationships

A model for LNEs is described in Section 3 and the model for networking instances is covered in Section 4.

The presented notional network device module can itself be thought of as a "meta-model" as it describes the relationships between individual models. We choose to represent it also as a simple YANG module consisting of other models, which are in fact independent top level individual models. Although it is never expected to be implemented.

The presented modules do not follow the hierarchy of any particular implementation, and hence is vendor-neutral. Nevertheless, the structure should be familiar to network operators and also readily mapped to vendor implementations.

The overall structure is:

```
module: network-device
   +--rw ietf-yang-library
   |
   +--rw interfaces
   +--rw hardware
   +--rw qos
   |
   +--rw system-management
   +--rw networking-services
   +--rw oam-protocols
   |
   +--rw routing
   +--rw mpls
   +--rw ieee-dot1Q
   |
   +--rw ietf-acl
   +--rw ietf-key-chain
   |
   +--rw logical-network-element
   +--rw networking-instance
```

The network device is composed of top level modules that can be used to configure and operate a network device. (This is a significant difference from earlier versions of this document where there was a strict model hierarchy.) Importantly the network device structure is the same for a physical network device or a logical network device, such as those instantiated via the logical-network-element model. Extra spacing is included to denote different types of modules included.

YANG library [YANG-LIBRARY] is included as it used to identify details of the top level modules supported by the (physical or logical) network device. Th ability to identify supported modules is particularly important for LNEs which may have a set of supported modules which differs from the set supported by the host network device.

The interface management model [RFC7223] is included at the top level. The hardware module is a placeholder for a future device-specific configuration and operational state data model. For example, a common structure for the hardware model might include chassis, line cards, and ports, but we leave this unspecified. The quality of service (QoS) section is also a placeholder module for device configuration and operational state data which relates to the treatment of traffic across the device. This document defines augmentations to the interface module to support LNEs and NIs. Similar elements, although perhaps only for LNEs, may also need to be

included as part of the definition of the future hardware and QoS modules.

System management, networking services, and oam protocols represent new top level modules that are used to organize data models of similar functions. Additional information on each is provided below.

The routing and MPLS modules provide core support for the configuration and operation of a devices control plane and data plane functions. IEEE dot1Q [IEEE-8021Q] is an example of another module that provides similar functions for VLAN bridging, and other similar modules are also possible. Each of these modules is expected to be LNE and NI unaware, and to be instantiated as needed as part of the LNE and NI configuration and operation supported by the logical-network-element and networking-instance modules. (Note that this is a change from [RTG-CFG] which is currently defined with VRF/NI semantics.)

The access control list (ACL) and key chain modules are included as examples of other top level modules that may be supported by a network device.

The logical network element and networking instance modules enable LNEs and NIs respectively and are defined below.

2.1. Interface Model Components

Interfaces are a crucial part of any network device's configuration and operational state. They generally include a combination of raw physical interfaces, link-layer interfaces, addressing configuration, and logical interfaces that may not be tied to any physical interface. Several system services, and layer 2 and layer 3 protocols may also associate configuration or operational state data with different types of interfaces (these relationships are not shown for simplicity). The interface management model is defined by [RFC7223].

The logical-network-element and networking-instance modules defined in this document augment the existing interface management model in two ways: The first, by the logical-network-element module, adds an identifier which is used on physical interface types to identify an associated LNE. The second, by the networking-instance module, adds a name which is used on sub-interface types to identify an associated networking instance. Similarly, this name is also added for IPv4 and IPv6 types, as defined in [RFC7277].

The interface related augmentations are as follows:

```
augment /if:interfaces/if:interface:
   +--rw bind-lne-name?   string

augment /if:interfaces/if:interface:
   +--rw bind-networking-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv4:
   +--rw bind-networking-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv6:
   +--rw bind-networking-instance-name?   string
```

The following is an example of envisioned combined usage. The interfaces container includes a number of commonly used components as examples:

```
          +--rw interfaces
          |  +--rw interface* [name]
          |     +--rw name                        string
          |     +--rw bind-lne-name?              string
          |     +--rw ethernet
          |     |  +--rw bind-networking-instance-name?   string
          |     |  +--rw aggregates
          |     |  +--rw rstp
          |     |  +--rw lldp
          |     |  +--rw ptp
          |     +--rw vlans
          |     +--rw tunnels
          |     +--rw ipv4
          |     |  +--rw bind-networking-instance-name?   string
          |     |  +--rw arp
          |     |  +--rw icmp
          |     |  +--rw vrrp
          |     |  +--rw dhcp-client
          |     +--rw ipv6
          |        +--rw bind-networking-instance-name?   string
          |        +--rw vrrp
          |        +--rw icmpv6
          |        +--rw nd
          |        +--rw dhcpv6-client
```

The [RFC7223] defined interface model is structured to include all interfaces in a flat list, without regard to logical or virtual instances (e.g., VRFs) supported on the device. The bind-lne-name and bind-networking-instance-name leaves provide the association between an interface and its associated LNE and NI (e.g., VRF or VSI).

2.2. System Management

[Editor's note: need to discuss and resolve relationship between this structure and RFC7317 and determine if 7317 is close enough to simply use as is.]

System management is expected to reuse definitions contained in [RFC7317]. It is expected to be instantiated per device and LNE. Its structure is shown below:

```
module: network-device
   +--rw system-management
   |  +--rw system-management-global
   |  +--rw system-management-protocol* [type]
   |     +--rw type    identityref
```

System-management-global is used for configuration information and state that is independent of a particular management protocol. System-management-protocol is a list of management protocol specific elements. The type-specific sub-modules are expected to be defined.

The following is an example of envisioned usage:

```
module: network-device
   +--rw system-management
      +--rw system-management-global
      |  +--rw statistics-collection
      |  ...
      +--rw system-management-protocol* [type]
      |  +--rw type=syslog
      |  +--rw type=dns
      |  +--rw type=ntp
      |  +--rw type=ssh
      |  +--rw type=tacacs
      |  +--rw type=snmp
      |  +--rw type=netconf
```

2.3. Networking Services

A device may provide different network services to other devices, for example a device my act as a DHCP server. The model may be instantiated per device, LNE, and NI. An identityref is used to identify the type of specific service being provided and its associated configuration and state information. The defined structure is as follows:

```
module: network-device
   +--rw networking-services
   |  +--rw networking-service* [type]
   |     +--rw type    identityref
```

The following is an example of envisioned usage: Examples shown below include a device-based Network Time Protocol (NTP) server, a Domain Name System (DNS) server, and a Dynamic Host Configuration Protocol (DHCP) server:

```
module: network-device
   +--rw networking-services
      +--rw networking-service* [type]
         +--rw type=ntp-server
         +--rw type=dns-server
         +--rw type=dhcp-server
```

2.4. OAM Protocols

OAM protocols that may run within the context of a device are grouped within the oam-protocols model. The model may be instantiated per device, LNE, and NI. An identifyref is used to identify the information and state that may relate to a specific OAM protocol. The defined structure is as follows:

```
module: network-device
   +--rw oam-protocols
      +--rw oam-protocol* [type]
         +--rw type    identityref
```

The following is an example of envisioned usage. Examples shown below include Bi-directional Forwarding Detection (BFD), Ethernet Connectivity Fault Management (CFM), and Two-Way Active Measurement Protocol (TWAMP):

```
module: network-device
   +--rw oam-protocols
      +--rw oam-protocol* [type]
         +--rw type=bfd
         +--rw type=cfm
         +--rw type=twamp
```

2.5. Routing

Routing protocol and IP forwarding configuration and operation information is modeled via a routing model, such as the one defined in [RTG-CFG]. Although, the defined routing module includes support

for NIs, which it refers to as Routing Instances, while the approach presented in this document presumes that the routing module is unaware of LNEs and NIs.

The routing module is expected to include all IETF defined control plane protocols, such as BGP, OSPF, LDP and RSVP-TE. It is also expected to support configuration and operation of or more routing information bases (RIB). A RIB is a list of routes complemented with administrative data. Finally, policy is expected to be represented within each control plane protocol and RIB.

The anticipated structure is as follows:

```
   module: network-device
       +--rw routing
          +--rw control-plane-protocols
          |  +--rw control-plane-protocol* [type]
          |     +--rw type      identityref
          |     +--rw policy
          +--rw ribs
             +--rw rib* [name]
                +--rw name           string
                +--rw description?   string
                +--rw policy
```

2.6. MPLS

MPLS data plane related information is grouped together, as with the previously discussed modules, is unaware of VRFs/NIs. The model may be instantiated per device, LNE, and NI. MPLS control plane protocols are expected to be included in Section 2.5. MPLS may reuse and build on [OC-MPLS] or other emerging models and has an anticipated structure as follows:

```
  module: network-device
       +--rw mpls
          +--rw global
          +--rw lsps* [type]
             +--rw type    identityref
```

Type refers to LSP type, such as static, traffic engineered or routing congruent. The following is an example of such usage:

```
module: network-device
     +--rw mpls
        +--rw global
        +--rw lsps* [type]
            +--rw type=static
            +--rw type=constrained-paths
            +--rw type=igp-congruent
```

3. Logical Network Elements

A logical network element is a network-device which is contained within another network-device. Using host-virtualization terminology one could refer to an LNE as a "Guest", and the containing network-device as the "Host". While LNEs may be implemented via host-virtualization technologies this is not a requirement.

Logical network elements represent the capability of some devices to partition resources into independent logical routers and/or switches. Device support for multiple logical network elements is implementation specific. Systems without such capabilities need not include support for the logical-network-element module. In physical devices, some hardware features are shared across partitions, but control plane (e.g., routing) protocol instances, tables, and configuration are managed separately. For example, in virtual routers or VNFs, this may correspond to establishing multiple logical instances using a single software installation. The model supports configuration of multiple instances on a single device by creating a list of logical network elements, each with their own configuration and operational state related to routing and switching protocols, as shown below:

```
module: logical-network-element
   +--rw logical-network-inventory
      +--rw logical-network-element* [name]
         +--rw name?   string
         +--rw description? string
         +--rw managed?     boolean
         +--rw root?        structural-mount
augment /if:interfaces/if:interface:
   +--rw bind-lne-name?     string
```

`name` identifies the logical network element. `managed` indicates if the host network device is able to manage the LNE via the `root` structure.

3.1. LNE Management - Host Network Device View

There are multiple implementation approaches possible to enable a network device to support the logical-network-element module and multiple LNEs. Some approaches will allow the management functions operating at network device level to access LNE configuration and operation information, while others will not. Similarly, even when LNE management from the network device is supported by the implementation, it may be prohibited by user policy.

The `managed` boolean mentioned above is used to indicate when LNE management from the network device context is possible. When the `managed` is `false`, the LNE cannot be managed by the host system and can only be managed from within the context of the LNE as described in the next section, Section 3.2.

When `managed` is `true`, the LNE can be managed from both the context of the LNE and the host network device. In this case, the same information that is available from within the LNE context is made available via the `root` element, with paths modified as described in [STRUCTURAL-MOUNT].

As an example, consider the case where an LNE with a `name` of "one" is defined on a network device. In this case the following structure might be made available:

```
 ..................................................................
                                              [ network-device state ]
    module: logical-network-element
       +--rw logical-network-inventory
          +--rw logical-network-element* [name]
             +--rw name="one"          string
             +--rw manged=true         boolean
             +--rw root                structural-mount

 ..................................................................
                                 [ LNE state exposed to network-device]

                +--rw ietf-yang-library
                +--rw interfaces
                +--rw hardware
                +--rw qos
                +--rw system-management
                +--rw networking-services
                +--rw oam-protocols
                +--rw routing
                +--rw mpls
                +--rw ieee-dot1Q
                +--rw networking-instance
 ..................................................................
```

As an LNE is a network device itself, all modules that may be present at the top level network device may also be present for the LNE, be made available under `root`, and be accessible via paths modified per [STRUCTURAL-MOUNT]. The list of available modules is expected to be implementation dependent. As is the method used by an implementation to support LNEs.

Resources assigned to the LNE will be represented in that LNE's resource modules. e.g., an LNE's interfaces module will contain the interfaces assigned to that LNE from the containing network-device.

3.2. LNE Management - LNE View

Management functions operating with the context of an LNE are accessed through standard LNE's management interfaces, e.g., NETCONF and SNMP. When accessing an LNE via an LNE's management interface, a network-device representation will be presented, but its scope will be limited to the specific LNE. Normal YANG/NETCONF mechanisms, together with ietf-yang-library, can be used to identify the available modules. Each supported module will be presented as a top level module. Only LNE associated resources will be reflected in resource related modules, e.g., interfaces, hardware and perhaps QoS. From the management perspective, there will be no difference between

the available LNE view (information) and an a physical network device.

Multiple implementation approaches are possible to provide LNE views, and these are outside the scope of this document.

3.3. LNE Instantiation

TBD -- need to resolve if instantiation is based on (a) new list entry creation, (2) YSDL, or (3) Structural Mount.

4. Network Instances

The network instance container is used to represent virtual routing and forwarding instances (VRFs) and virtual switching instances (VSIs), [RFC4026]. VRFs and VSIs are commonly used to isolate routing and switching domains, for example to create virtual private networks, each with their own active protocols and routing/switching policies. The model represents both core/provider and virtual instances. Networking instances reuse and build on [RTG-CFG] and are shown below:

```
module: networking-instance
   +--rw networking-instances
      +--rw networking-instance* [name]
         +--rw name                          string
         +--rw type?                         identityref
         +--rw enabled?                      boolean
         +--rw description?                  string
         +--rw networking-instance-policy
         |  ...
         +--rw root?     structural-mount
         |  ...
augment /if:interfaces/if:interface:
   +--rw bind-networking-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv4:
   +--rw bind-networking-instance-name?   string
augment /if:interfaces/if:interface/ip:ipv6:
   +--rw bind-networking-instance-name?   string
```

A networking instance is identified by a `name` string. This string is used both as an index within the networking-instance module and to associate resources with a networking instance is shown above in the interface augmentation. Type is used to indicate the type NI, such as L3-VRF, VPLS, L2-VSI, etc. Networking instance policy and root are discussed in greater detail below.

4.1. Network Instance Policy

Networking instance policies are used to control how NI information is represented at the device level, VRF routing policies, and VRF/VSI identifiers. Examples include BGP route targets (RTs) and route distinguishers (RDs), virtual network identifiers (VN-IDs), VPLS neighbors, etc. The structure is expected to be:

```
    module: networking-instance
       +--rw networking-instances
          +--rw networking-instance* [name]
             +--rw networking-instance-policy
                (TBD)
```

4.2. Network Instance Management

Modules that may be used to represent networking instance specific information will be available under `root`. As with LNEs, actual module availability is expected to be implementation dependent. The ietf-yang-library module is expected to be the primary method used to identify supported modules. Resource related control and assignment is expected to be managed at the network-device level, not the networking instance level, based on the `bind-networking-instance-name` augmentation mentioned above.

As an example, consider the case where a networking instance with a `name` of "green" is defined on a network device. In this case the following structure might be made available:

```
  module: networking-instance
     +--rw ietf-yang-library
     +--rw interfaces
     |  +--rw bind-networking-instance-name="green"  string
     +--rw system-management
     +--rw networking-instances
        +--rw networking-instance* [name]
           +--rw name="green"    string
           +--rw type?                              identityref
           +--rw enabled=true                       boolean
           +--rw description="The Green VRF"        string
           +--rw networking-instance-policy
           |  ... (RT=1000:1, RD=1.2.3.4)
           +--rw root?                              structural-mount
              +--rw ietf-yang-library
              +--rw networking-services
              +--rw oam-protocols
              +--rw routing
              +--rw mpls
```

All modules that represent control-plane and data-plane information may be present at the `root`, and be accessible via paths modified per [STRUCTURAL-MOUNT]. The list of available modules is expected to be implementation dependent. As is the method used by an implementation to support NIs.

4.3. Network Instance Instantiation

TBD -- need to resolve if instantiation is based on (a) new list entry creation, (2) YSDL, or (3) Structural Mount.

5. Security Considerations

The network-device model structure described in this document does not define actual configuration and state data, hence it is not directly responsible for security risks.

Each of the component models that provide the corresponding configuration and state data should be considered sensitive from a security standpoint since they generally manipulate aspects of network configurations. Each component model should be carefully evaluated to determine its security risks, along with mitigations to reduce such risks.

LNE portion is TBD

NI portion is TBD

6. IANA Considerations

This YANG model currently uses a temporary ad-hoc namespace. If it is placed or redirected for the standards track, an appropriate namespace URI will be registered in the "IETF XML Registry" [RFC3688]. The YANG structure modules will be registered in the "YANG Module Names" registry [RFC6020].

7. YANG Modules

The structure of the models defined in this document are described by the YANG module below.

7.1. Network Device Model Structure

```
<CODE BEGINS> file "network-device@2016-01-19.yang"
module network-device {

  yang-version "1";
```

```
     // namespace
     namespace "urn:ietf:params:xml:ns:yang:network-device";

     prefix "struct";

     // import some basic types

     // meta
     organization "IETF RTG YANG Design Team Collaboration
                   with OpenConfig";

     contact
         "Routing Area YANG Architecture Design Team -
          <rtg-dt-yang-arch@ietf.org>";

     description
       "This module describes a model structure for YANG
        configuration and operational state data models. Its intent is
        to describe how individual device protocol and feature models
        fit together and interact.";

     revision "2016-01-19" {
       description
         "IETF Routing YANG Design Team Meta-Model";
       reference "TBD";
     }

     // extension statements

     // identity statements

     identity oam-protocol-type {
         description
             "Base identity for derivation of OAM protocols";
     }

     identity networking-service-type {
         description
             "Base identity for derivation of networking services";
     }

      identity system-management-protocol-type {
         description
             "Base identity for derivation of system management
              protocols";
      }

      identity oam-service-type {
```

```
         description
             "Base identity for derivation of Operations,
              Administration, and Maintenance (OAM) services.";
      }

      identity control-plane-protocol-type {
         description
             "Base identity for derivation of control-plane protocols";
      }

      identity mpls-lsp-type {
         description
             "Base identity for derivation of MPLS LSP typs";
      }

     // typedef statements

     // grouping statements

     grouping control-plane-protocols {
         description
             "Grouping for control plane protocols configured for
              a networking-instance";
         container control-plane-protocols {
             description
                 "Container for control plane protocols configured for
                  a networking instance.";
             list control-plane-protocol {
                 key "type";
                 description
                     "List of control plane protocols configured for
                      a networking instance.";
                 leaf type {
                     type identityref {
                         base control-plane-protocol-type;
                     }
                     mandatory true;
                     description
                         "The control plane protocol type, e.g., BGP,
                          OSPF IS-IS, etc";
                 }
                 container policy {
                     description
                         "Protocol specific policy,
                         reusing [RTG-POLICY]";
                 }
             }
         }
```

```
    }

    grouping ribs {
      description
        "Routing Information Bases (RIBs) supported by a
         networking-instance";
      container ribs {
          description
              "RIBs supported by a networking-instance";
          list rib {
              key "name";
              min-elements "1";
              description
                  "Each entry represents a RIB identified by the
                 'name' key. All routes in a RIB must belong to the
                  same address family.

                  For each routing instance, an implementation should
                  provide one system-controlled default RIB for each
                  supported address family.";
              leaf name {
                  type string;
                  description
                      "The name of the RIB.";
              }
              reference "draft-ietf-netmod-routing-cfg";
              leaf description {
                  type string;
                  description
                      "Description of the RIB";
              }
              // Note that there is no list of interfaces within
              container policy {
                  description "Policy specific to RIB";
              }
          }
      }
    }

    // top level device definition statements
    container ietf-yang-library {
      description
        "YANG Module Library as defined in
         draft-ietf-netconf-yang-library";
    }

    container interfaces {
      description
```

```
        "Interface list as defined by RFC7223/RFC7224";
    }

    container hardware {
      description
        "Hardware / vendor-specific data relevant to the platform.
        This container is an anchor point for platform-specific
        configuration and operational state data.  It may be further
        organized into chassis, line cards, ports, etc.  It is
        expected that vendor or platform-specific augmentations
        would be used to populate this part of the device model";
    }

    container qos {
      description "QoS features, for example policing, shaping, etc.";
    }

    container system-management {
        description
          "System management for physical or virtual device.";
        container system-management-global {
            description "System management - with reuse of RFC 7317";
        }
        list system-management-protocol {
            key "type";
            leaf type {
                type identityref {
                    base system-management-protocol-type;
                }
                mandatory true;
                description
                    "Syslog, ssh, TACAC+, SNMP, NETCONF, etc.";
            }
            description "List of system management protocol
                         configured for a logical networking
                         element.";
        }
    }

    container networking-services {
        description
            "Container for list of configured networking
             services.";
        list networking-service {
            key "type";
            description
                "List of networking services configured for a
                 networking instance.";
```

```
            leaf type {
                type identityref {
                    base networking-service-type;
                }
                mandatory true;
                description
                    "The networking services type supported within
                     a networking instance, e.g., NTP server, DNS
                     server, DHCP server, etc.";
            }
        }
    }

    container oam-protocols {
        description
            "Container for configured OAM protocols.";
        list oam-protocol {
            key "type";
            leaf type {
                type identityref {
                    base oam-protocol-type;
                }
                mandatory true;
                description
                    "The Operations, Administration, and
                     Maintenance (OAM) protocol type, e.g., BFD,
                     TWAMP, CFM, etc.";
            }
            description
                "List of configured OAM protocols.";
        }
    }

    container routing {
      description
        "The YANG Data Model for Routing Management revised to be
         Network Instance / VRF independent. ";
      // Note that there is no routing or network instance
      uses control-plane-protocols;
      uses ribs;
    }

    container mpls {
        description "MPLS and TE configuration";
        container global {
            description "Global MPLS configuration";
        }
        list lsps {
```

```
            key "type";
            description
                "List of LSP types.";
            leaf type {
                type identityref {
                    base mpls-lsp-type;
                }
                mandatory true;
                description
                    "MPLS and Traffic Engineering protocol LSP types,
                     static, LDP/SR (igp-congruent),
                     RSVP TE (constrained-paths) , etc.";
            }
        }
    }

    container ieee-dot1Q {
      description
        "The YANG Data Model for VLAN bridges as defined by the IEEE";
    }

    container ietf-acl {
      description "Packet Access Control Lists (ACLs) as specified
                   in draft-ietf-netmod-acl-model";
    }

    container ietf-key-chain {
      description "Key chains as specified in
                  draft-ietf-rtgwg-yang-key-chain;";
    }

    container logical-network-element {
      description
        "This module is used to support multiple logical network
         elements on a single physical or virtual system.";
    }

    container networking-instance {
      description
        "This module is used to support multiple network instances
         within a single physical or virtual device.  Network
         instances are commonly know as VRFs (virtual routing
         and forwarding) and VSIs (virtual switching instances).";
    }
    // rpc statements

    // notification statements
```

```
}
<CODE ENDS>
```

7.2. Logical Network Element Model

```
<CODE BEGINS> file "logical-network-element@2016-01-19.yang"
module logical-network-element {

  yang-version "1";

  // namespace
  namespace "urn:ietf:params:xml:ns:yang:logical-network-element";

  prefix "struct";

  // import some basic types
  import ietf-interfaces {
    prefix if;
  }

  // meta
  organization "IETF RTG YANG Design Team Collaboration
                with OpenConfig";

  contact
      "Routing Area YANG Architecture Design Team -
       <rtg-dt-yang-arch@ietf.org>";

  description
    "This module is used to support multiple logical network
     elements on a single physical or virtual system.";

  revision "2016-01-19" {
    description
      "IETF Routing YANG Design Team Meta-Model";
    reference "TBD";
  }

  // extension statements

  // feature statements
  feature bind-network-element-id {
    description
      "Logical network element ID to which an interface is bound";
  }

  // identity statements
```

```
identity logical-network-element-type {
  description
   "Identify type of logical-network-element";
}

identity lne-managed {
  base logical-network-element-type;
  description
    "A Logical Network Element that can
     be managed by the host system";
}

identity lne-unmanaged {
  base logical-network-element-type;
  description
    "A Logical Network Element that cannot
     be managed by the host system";
}

// typedef statements

// grouping statements

// top level device definition statements
container logical-network-inventory {
  description "Allows a network device to support multiple logical
               network element (device) instances";
  list logical-network-element {
    key lne-id;
    description "List of logical network elements";
    leaf lne-id {
      type uint16; // expect a small number of logical routers
      description "Device-wide unique identifier for the
                   logical network element";
    }
    leaf lne-name {
      type string;
      description "Descriptive name for the logical network
                   element";
    }
    leaf lne-type {
      type identityref {
        base logical-network-element-type;
      }
      description "Type of logical-network-element";
    }
    leaf lne-root {
      type structural-mount;
```

```
          description "Root for models supported per logical
                       network element";
        }
      }
    }

    // augment statements
    augment "/if:interfaces/if:interface" {
      description
          "Add a node for the identification of the logical network
          element associated with an interface. Applies to interfaces
          that can be assigned on a per logical network element basis.
          A <TBD> error is returned when the interface type cannot be
          assigned.";

      leaf bind-lne-id {
        type uint16;
        description
          "Logical network element ID to which interface is bound";
      }
    }

    // rpc statements

    // notification statements

  }
  <CODE ENDS>
```

7.3. Networking Instance Model

```
  <CODE BEGINS> file "networking-instance@2016-01-20.yang"
  module networking-instance {

    yang-version "1";

    // namespace
    namespace "urn:ietf:params:xml:ns:yang:networking-instance";

    prefix "struct";

    // import some basic types
    import ietf-interfaces {
      prefix if;
    }

    import ietf-ip {
      prefix ip;
```

```
    }

    // meta
    organization "IETF RTG YANG Design Team Collaboration
                  with OpenConfig";

    contact
        "Routing Area YANG Architecture Design Team -
         <rtg-dt-yang-arch@ietf.org>";

    description
      "This module is used to support multiple network instances
       within a single physical or virtual device.  Network
       instances are commonly know as VRFs (virtual routing
       and forwarding) and VSIs (virtual switching instances).";

    revision "2016-01-20" {
      description
        "IETF Routing YANG Design Team Meta-Model";
      reference "TBD";
    }

    // extension statements

    feature bind-networking-instance-name {
      description
        "Networking Instance to which an interface instance is bound";
    }

    // identity statements

    identity networking-instance-type {
        description
           "Base identity from which identities describing
            networking instance types are derived.";
    }

     identity ipv4-interface-protocol-type {
        description
            "Base identity for derivation of IPv4 interface
             protocols";
     }

     identity ipv6-interface-protocol-type {
        description
            "Base identity for derivation of IPv6 interface
             protocols";
     }
```

```
    // typedef statements

    // grouping statements

    grouping interface-ip-common {
      description
        "interface-specific configuration for IP interfaces, IPv4 and
        IPv6";

    }

    grouping ipv4-interface-protocols {
        container ipv4-interface-protocols {
            list ipv4-interface-protocol {
                key "type";
                leaf type {
                    type identityref {
                        base ipv4-interface-protocol-type;
                    }
                    mandatory true;
                    description
                        "ARP, ICMP, VRRP, DHCP Client, etc.";
                }
                description
                    "List of IPv4 protocols configured
                     on an interface";
            }
            description
                "Container for list of IPv4 protocols configured
                  on an interface";
        }
        description
            "Grouping for IPv4 protocols configured on an interface";
    }

    grouping ipv6-interface-protocols {
        description
            "Grouping for IPv6 protocols configured on
             an interface.";
        container ipv6-interface-protocols {
            description
                "Container for list of IPv6 protocols configured
                  on an interface.";
            list ipv6-interface-protocol {
                key "type";
                description
                    "List of IPv6 protocols configured
                     on an interface";
```

```
                leaf type {
                    type identityref {
                        base ipv6-interface-protocol-type;
                    }
                    mandatory true;
                    description
                        "ND, ICMPv6, VRRP, DHCPv6 Client, etc.";
                }
            }
        }
    }

    grouping networking-instance-policy {
      description
          "Networking instance policies such as route
           distinguisher, route targets, VPLS ID and neighbor,
           Ethernet ID, etc. ";
      reference
          "RFC 4364 - BGP/MPLS Virtual Private Networks (VPNs)
           RFC 6074 - Provisioning, Auto-Discovery, and Signaling
                in Layer 2 Virtual Private Networks (L2VPNs)
           RFC 7432 - BGP MPLS-Based Ethernet VPN";
      container networking-instance-policy {
          description "Networking Instance Policy -- details TBD";
      }
    }

    // top level device definition statements
    container networking-instances {
        description "Networking instances each of which have
                     an independent IP/IPv6 addressing space
                     and protocol instantiations. For layer 3,
                     this consistent with the routing-instance
                     definition in ietf-routing";
        reference "draft-ietf-netmod-routing-cfg";
        list networking-instance {
            key name;
            description "List of networking-instances";
            leaf name {
                type string;
                description "device scoped
                             identifier for the networking
                             instance";
            }
            leaf type {
                type identityref {
                    base networking-instance-type;
                }
```

```
                description
                    "The networking instance type -- details TBD
                     Likely types include core, L3-VRF, VPLS,
                     L2-cross-connect, L2-VSI, etc.";
            }
            leaf enabled {
                type boolean;
                default "true";
                description
                  "Flag indicating whether or not the networking
                   instance is enabled.";
            }
            leaf description {
                type string;
                description
                  "Description of the networking instance
                  and its intended purpose";
            }
            uses networking-instance-policy;
            leaf root {
              type structural-mount;
              description "Root for models supported per
                           networking instance";
            }
        }
    }

    // augment statements

    augment "/if:interfaces/if:interface" {
      description
          "Add a node for the identification of the logical networking
          instance (which is within the interface's identified logical
          network element) associated with the IP information
          configured on an interface";

      leaf bind-networking-instance-name {
        type string;
        description
          "Networking Instance to which an interface is bound";
      }
    }

    augment "/if:interfaces/if:interface/ip:ipv4" {
      description
          "Add a node for the identification of the logical
          networking instance (which is within the interface's
          identified physical or virtual device) associated with
```

```
          the IP information configured on an interface";

      leaf bind-networking-instance-name {
        type string;
        description
          "Networking Instance to which IPv4 interface is bound";

      }
    }

    augment "/if:interfaces/if:interface/ip:ipv6" {
      description
          "Add a node for the identification of the logical
          networking instance (which is within the interface's
          identified physical or virtual device) associated with
          the IP information configured on an interface";

      leaf bind-networking-instance-name {
        type string;
        description
          "Networking Instance to which IPv6 interface is bound";

      }
    }

    // rpc statements

    // notification statements

  }
  <CODE ENDS>
```

8. References

8.1. Normative References

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC4026] Andersson, L. and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology", RFC 4026, March 2005.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC7223] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 7223, May 2014.

[RFC7277] Bjorklund, M., "A YANG Data Model for IP Management", RFC 7277, June 2014.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, August 2014.

[STRUCTURAL-MOUNT]
Bjorklund, M., "YANG Structural Mount", draft-bjorklund-netmod-structural-mount-00.txt (work in progress), December 2015.

[YANG-YSDL]
Lhotha, L., "YANG Schema Dispatching Language", draft-lhotka-netmod-ysdl-00.txt (work in progress), November 2015.

8.2. Informative References

[IEEE-8021Q]
Holness, M., "IEEE 802.1Q YANG Module Specifications", IEEE-Draft http://www.ieee802.org/1/files/public/docs2015/new-mholness-yang-8021Q-0515-v04.pdf, May 2015.

[NETMOD-OPSTATE]
Watsen, K. and T. Nadeau, "NETMOD Operational State Requirements", draft-ietf-netmod-opstate-reqs-03.txt (work in progress), January 2016.

[OC-MPLS] George, J., Fang, L., Osborne, E., and R. Shakir, "MPLS / TE Model for Service Provider Networks", draft-openconfig-mpls-consolidated-model-02.txt (work in progress), October 2015.

[OC-OPSTATE]
Shakir, R., Shaikh, A., and M. Hines, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-opstate-01.txt (work in progress), July 2015.

[OC-STRUCT]
Shaikh, A., Shakir, R., D'Souza, K., and L. Fang, "Consistent Modeling of Operational State Data in YANG", draft-openconfig-netmod-model-structure-00.txt (work in progress), March 2015.

[RTG-CFG] Lhotha, L. and A. Lindem, "A YANG Data Model for Routing Management", draft-ietf-netmod-routing-cfg-20.txt (work in progress), October 2015.

[YANG-LIBRARY]
Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", draft-ietf-netconf-yang-library-03.txt (work in progress), December 2015.

Appendix A. Acknowledgments

This document is derived from draft-openconfig-netmod-model-structure-00. The Authors of that document who are not also authors of this document are listed as Contributors to this work.

The original stated: The authors are grateful for valuable contributions to this document and the associated models from: Deepak Bansal, Paul Borman, Chris Chase, Josh George, Marcus Hines, and Jim Uttaro.

The Routing Area Yang Architecture design team members included Acee Lindem, Anees Shaikh, Christian Hopps, Dean Bogdanovic, Lou Berger, Qin Wu, Rob Shakir, Stephane Litkowski, and Yan Gang.

The identityref approach was proposed by Mahesh Jethanandani.

The RFC text was produced using Marshall Rose's xml2rfc tool.

Appendix B. Contributors

Contributors' Addresses

Anees Shaikh
Google
1600 Amphitheatre Pkwy
Mountain View, CA 94043
United States
Email: aashaikh@google.com

Rob Shakir
Jive Communications, Inc.
1275 W 1600 N, Suite 100
Orem, UT 84057
United States
Email: rjs@rob.sh

Kevin D'Souza
AT&T
200 S. Laurel Ave
Middletown, NJ
United States
Email: kd6913@att.com

Luyuan Fang
Microsoft
205 108th Ave. NE, Suite 400
Bellevue, WA
United States
Email: lufang@microsoft.com

Qin Wu
Huawei Technologies
101 Software Avenue, Yuhua District
Nanjing, Jiangsu 210012
China

Email: bill.wu@huawei.com

Stephane Litkowski
Orange
9 rue du chene germain
Cesson Sevigne 35512
France

Email: stephane.litkowski@orange.com

Gang Yan
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: yangang@huawei.com

Authors' Addresses

Acee Lindem (editor)
Cisco Systems
301 Midenhall Way
Cary, NC 27513
USA

Email: acee@cisco.com

Lou Berger (editor)
LabN Consulting, L.L.C.

Email: lberger@labn.net

Dean Bogdanovic

Email: ivandean@gmail.com

Christan Hopps
Deutsche Telekom

Email: chopps@chopps.org