Internet Engineering Task Force (IETF)
Request for Comments: 8345
Category: Standards Track
ISSN: 2070-1721

A. Clemm
Huawei
J. Medved
Cisco
R. Varga
Pantheon Technologies SRO
N. Bahadur
Bracket Computing
H. Ananthakrishnan
Packet Design
X. Liu
Jabil

March 2018

A YANG Data Model for Network Topologies

Abstract

This document defines an abstract (generic, or base) YANG data model for network/service topologies and inventories. The data model serves as a base model that is augmented with technology-specific details in other, more specific topology and inventory data models.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8345.

Copyright Notice

Table of Contents

1. Introduction

This document introduces an abstract (base) YANG [RFC7950] data model [RFC3444] to represent networks and topologies. The data model is divided into two parts: The first part of the data model defines a network data model that enables the definition of network hierarchies, or network stacks (i.e., networks that are layered on top of each other) and maintenance of an inventory of nodes contained in a network. The second part of the data model augments the basic network data model with information to describe topology information. Specifically, it adds the concepts of "links" and "termination points" to describe how nodes in a network are connected to each other. Moreover, the data model introduces vertical layering relationships between networks that can be augmented to cover both network inventories and network/service topologies.

Although it would be possible to combine both parts into a single data model, the separation facilitates integration of network topology and network inventory data models, because it allows network inventory information to be augmented separately, and without concern for topology, into the network data model.

The data model can be augmented to describe the specifics of particular types of networks and topologies. For example, an augmenting data model can provide network node information with attributes that are specific to a particular network type. Examples of augmenting models include data models for Layer 2 network topologies; Layer 3 network topologies such as unicast IGP, IS-IS [RFC1195], and OSPF [RFC2328]; traffic engineering (TE) data [RFC3209]; or any of the variety of transport and service topologies. Information specific to particular network types will be captured in separate, technology-specific data models.

The basic data models introduced in this document are generic in nature and can be applied to many network and service topologies and inventories. The data models allow applications to operate on an inventory or topology of any network at a generic level, where the specifics of particular inventory/topology types are not required. At the same time, where data specific to a network type comes into play and the data model is augmented, the instantiated data still adheres to the same structure and is represented in a consistent fashion. This also facilitates the representation of network hierarchies and dependencies between different network components and network types.

The abstract (base) network YANG module introduced in this document, entitled "ietf-network" (Section 6.1), contains a list of abstract network nodes and defines the concept of "network hierarchy" (network

stack). The abstract network node can be augmented in inventory and topology data models with inventory-specific and topology-specific attributes. The network hierarchy (stack) allows any given network to have one or more "supporting networks". The relationship between the base network data model, the inventory data models, and the topology data models is shown in Figure 1 (dotted lines in the figure denote possible augmentations to models defined in this document).

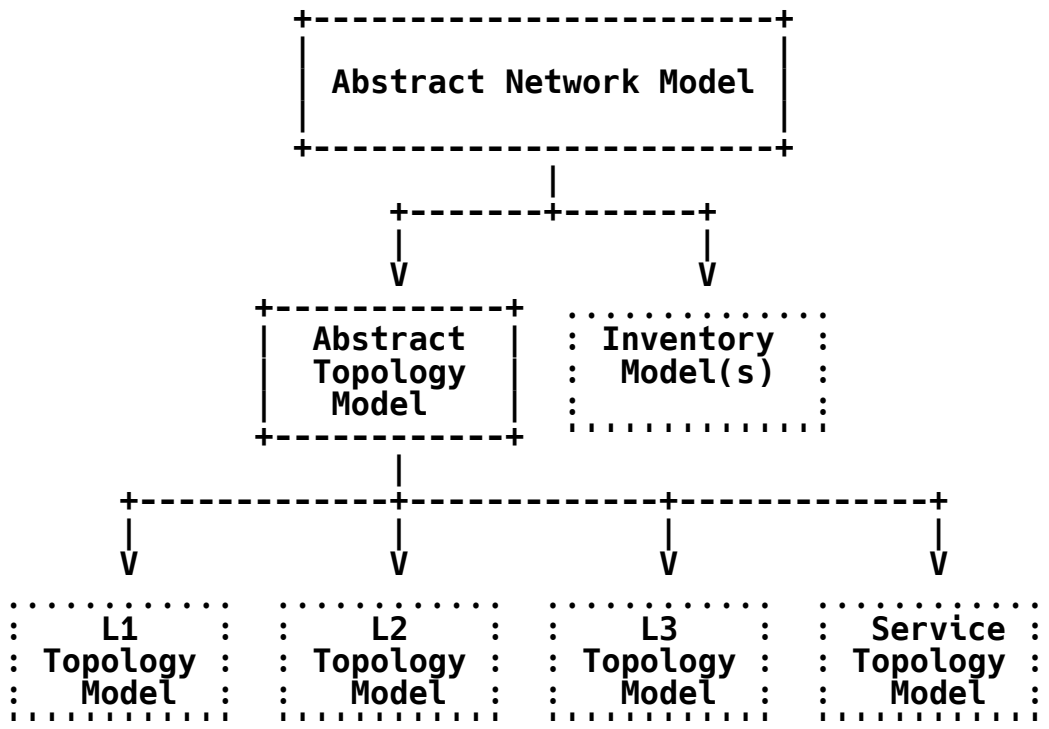

Figure 1: The Network Data Model Structure

The network-topology YANG module introduced in this document, entitled "ietf-network-topology" (Section 6.2), defines a generic topology data model at its most general level of abstraction. The module defines a topology graph and components from which it is composed: nodes, edges, and termination points. Nodes (from the "ietf-network" module) represent graph vertices and links represent graph edges. Nodes also contain termination points that anchor the links. A network can contain multiple topologies -- for example, topologies at different layers and overlay topologies. The data model therefore allows relationships between topologies, as well as dependencies between nodes and termination points across topologies, to be captured. An example of a topology stack is shown in Figure 2.

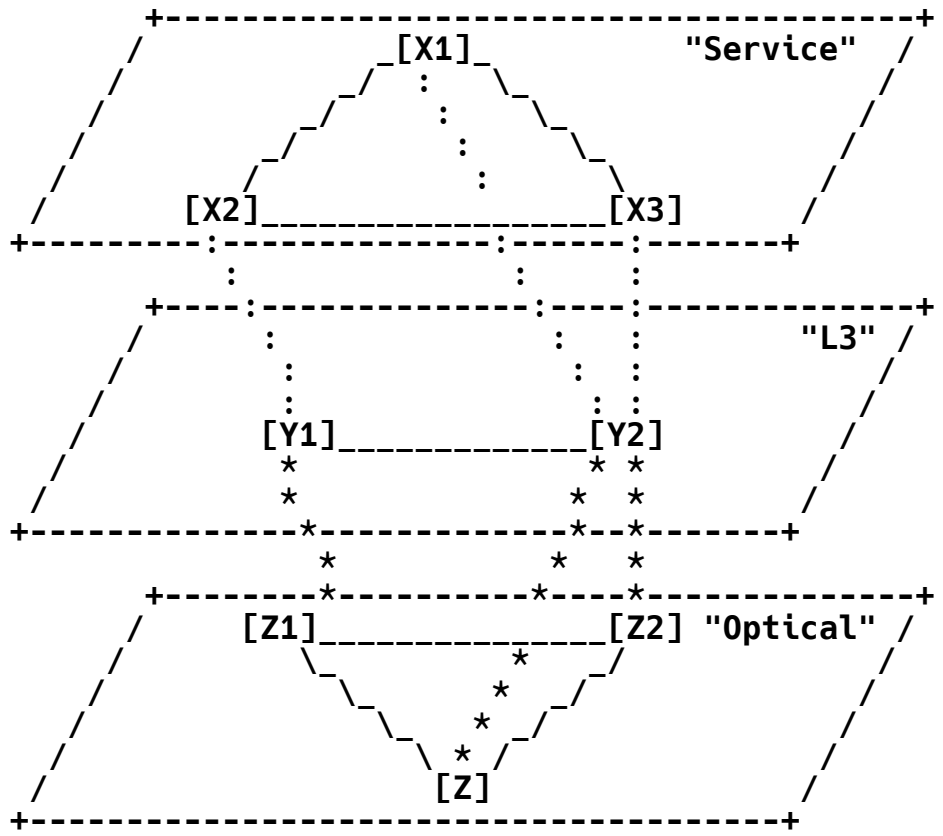

Figure 2: Topology Hierarchy (Stack) Example

Figure 2 shows three topology levels. At the top, the "Service" topology shows relationships between service entities, such as service functions in a service chain. The "L3" topology shows network elements at Layer 3 (IP), and the "Optical" topology shows network elements at Layer 1. Service functions in the "Service" topology are mapped onto network elements in the "L3" topology, which in turn are mapped onto network elements in the "Optical" topology. Two service functions (X1 and X3) are mapped onto a single L3 network element (Y2); this could happen, for example, if two service functions reside in the same Virtual Machine (VM) (or server) and share the same set of network interfaces. A single "L3" network element (Y2) is mapped onto two "Optical" network elements (Z2 and Z). This could happen, for example, if a single IP router attaches to multiple Reconfigurable Optical Add/Drop Multiplexers (ROADMs) in the optical domain.

Another example of a service topology stack is shown in Figure 3.

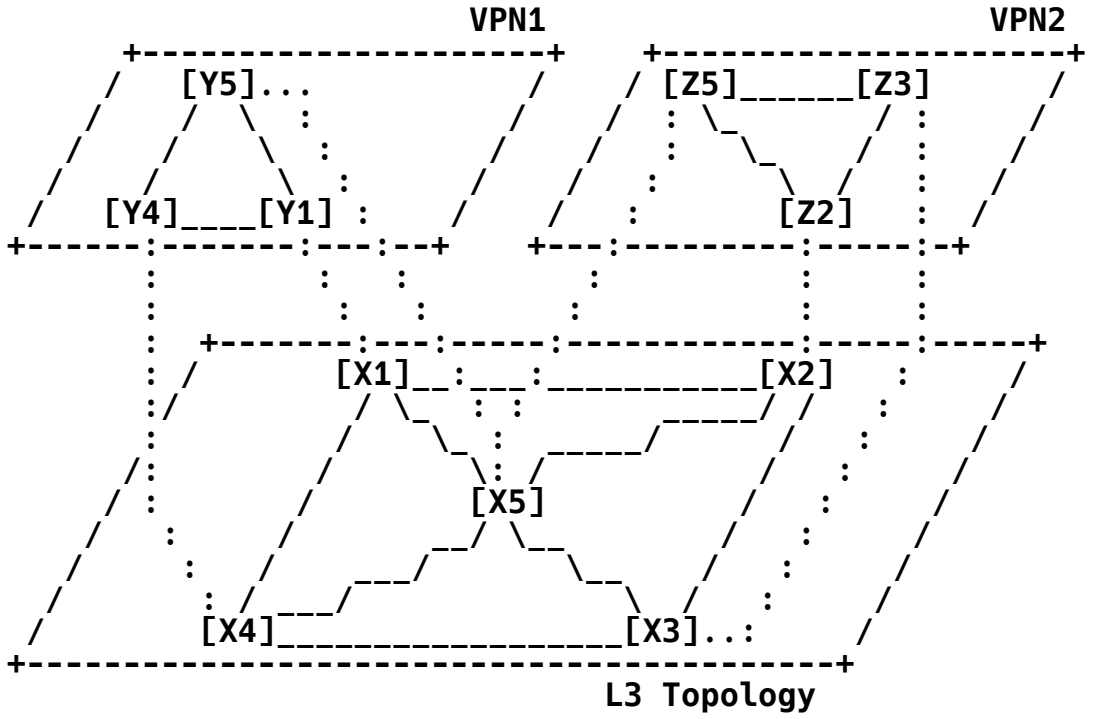

Figure 3: Topology Hierarchy (Stack) Example

Figure 3 shows two VPN service topologies (VPN1 and VPN2) instantiated over a common L3 topology. Each VPN service topology is mapped onto a subset of nodes from the common L3 topology.

There are multiple applications for such a data model. For example, within the context of Interface to the Routing System (I2RS), nodes within the network can use the data model to capture their understanding of the overall network topology and expose it to a network controller. A network controller can then use the instantiated topology data to compare and reconcile its own view of the network topology with that of the network elements that it controls. Alternatively, nodes within the network could propagate this understanding to compare and reconcile this understanding either among themselves or with the help of a controller. Beyond the network element and the immediate context of I2RS itself, a network controller might even use the data model to represent its view of the topology that it controls and expose it to applications north of itself. Further use cases where the data model can be applied are described in [USECASE-REQS].

In this data model, a network is categorized as either system controlled or not. If a network is system controlled, then it is automatically populated by the server and represents dynamically learned information that can be read from the operational state datastore. The data model can also be used to create or modify network topologies that might be associated with an inventory model or with an overlay network. Such a network is not system controlled; rather, it is configured by a client.

The data model allows a network to refer to a supporting network, supporting nodes, supporting links, etc. The data model also allows the layering of a network that is configured on top of a network that is system controlled. This permits the configuration of overlay networks on top of networks that are discovered. Specifically, this data model is structured to support being implemented as part of the ephemeral datastore [RFC8342], the requirements for which are defined in Section 3 of [RFC8242]. This allows network topology data that is written, i.e., configured by a client and not system controlled, to refer to dynamically learned data that is controlled by the system, not configured by a client. A simple use case might involve creating an overlay network that is supported by the dynamically discovered IP-routed network topology. When an implementation places written data for this data model in the ephemeral datastore, such a network MAY refer to another network that is system controlled.

2. Key Words

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

3. Definitions and Abbreviations

Datastore: A conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations, or combinations thereof. A datastore maps to an instantiated YANG data tree (definition from [RFC8342]).

Data subtree: An instantiated data node and the data nodes that are hierarchically contained within it.

IGP: Interior Gateway Protocol.

IS-IS: Intermediate System to Intermediate System.

OSPF: Open Shortest Path First (a link-state routing protocol).

SDN: Software-Defined Networking.

URI: Uniform Resource Identifier.

VM: Virtual Machine.

4. Model Structure Details

4.1. Base Network Model

The abstract (base) network data model is defined in the "ietf-network" module. Its structure is shown in Figure 4. The notation syntax follows the syntax used in [RFC8340].

```
module: ietf-network
  +--rw networks
     +--rw network* [network-id]
        +--rw network-id            network-id
        +--rw network-types
        +--rw supporting-network* [network-ref]
        |  +--rw network-ref    -> /networks/network/network-id
        +--rw node* [node-id]
           +--rw node-id            node-id
           +--rw supporting-node* [network-ref node-ref]
              +--rw network-ref
              |       -> ../../../supporting-network/network-ref
              +--rw node-ref       -> /networks/network/node/node-id
```

Figure 4: The Structure of the Abstract (Base) Network Data Model

The data model contains a container with a list of networks. Each network is captured in its own list entry, distinguished via a network-id.

A network has a certain type, such as L2, L3, OSPF, or IS-IS. A network can even have multiple types simultaneously. The type or types are captured underneath the container "network-types". In this model, it serves merely as an augmentation target; network-specific modules will later introduce new data nodes to represent new network types below this target, i.e., will insert them below "network-types" via YANG augmentation.

When a network is of a certain type, it will contain a corresponding data node. Network types SHOULD always be represented using presence containers, not leafs of type "empty". This allows the representation of hierarchies of network subtypes within the instance information. For example, an instance of an OSPF network (which, at the same time, is a Layer 3 unicast IGP network) would contain underneath "network-types" another presence container "l3-unicast-igp-network", which in turn would contain a presence container "ospf-network". Actual examples of this pattern can be found in [RFC8346].

A network can in turn be part of a hierarchy of networks, building on top of other networks. Any such networks are captured in the list "supporting-network". A supporting network is, in effect, an underlay network.

Furthermore, a network contains an inventory of nodes that are part of the network. The nodes of a network are captured in their own list. Each node is identified relative to its containing network by a node-id.

It should be noted that a node does not exist independently of a network; instead, it is a part of the network that contains it. In cases where the same device or entity takes part in multiple networks, or at multiple layers of a networking stack, the same device or entity will be represented by multiple nodes, one for each network. In other words, the node represents an abstraction of the device for the particular network of which it is a part. To indicate that the same entity or device is part of multiple topologies or networks, it is possible to create one "physical" network with a list of nodes for each of the devices or entities. This (physical) network -- the nodes (entities) in that network -- can then be referred to as an underlay network and as nodes from the other (logical) networks and nodes, respectively. Note that the data model

allows for the definition of more than one underlay network (and node), allowing for simultaneous representation of layered network topologies and service topologies, and their physical instantiation.

Similar to a network, a node can be supported by other nodes and map onto one or more other nodes in an underlay network. This is captured in the list "supporting-node". The resulting hierarchy of nodes also allows for the representation of device stacks, where a node at one level is supported by a set of nodes at an underlying level. For example:

- a "router" node might be supported by a node representing a route processor and separate nodes for various line cards and service modules,
- a virtual router might be supported or hosted on a physical device represented by a separate node,

and so on.

Network data of a network at a particular layer can come into being in one of two ways: (1) the network data is configured by client applications -- for example, in the case of overlay networks that are configured by an SDN Controller application, or (2) the network data is automatically controlled by the system, in the case of networks that can be discovered. It is possible for a configured (overlay) network to refer to a (discovered) underlay network.

The revised datastore architecture [RFC8342] is used to account for those possibilities. Specifically, for each network, the origin of its data is indicated per the "origin" metadata [RFC7952] annotation (as defined in [RFC8342]) -- "intended" for data that was configured by a client application and "learned" for data that is discovered. Network data that is discovered is automatically populated as part of the operational state datastore. Network data that is configured is part of the configuration and intended datastores, respectively. Configured network data that is actually in effect is, in addition, reflected in the operational state datastore. Data in the operational state datastore will always have complete referential integrity. Should a configured data item (such as a node) have a dangling reference that refers to a non-existing data item (such as a supporting node), the configured data item will automatically be removed from the operational state datastore and thus only appear in the intended datastore. It will be up to the client application (such as an SDN Controller) to resolve the situation and ensure that the reference to the supporting resources is configured properly.

4.2. Base Network Topology Data Model

The abstract (base) network topology data model is defined in the "ietf-network-topology" module. It builds on the network data model defined in the "ietf-network" module, augmenting it with links (defining how nodes are connected) and termination points (which anchor the links and are contained in nodes). The structure of the network topology module is shown in Figure 5. The notation syntax follows the syntax used in [RFC8340].

```
module: ietf-network-topology
  augment /nw:networks/nw:network:
    +--rw link* [link-id]
       +--rw link-id            link-id
       +--rw source
       |  +--rw source-node?   -> ../../../nw:node/node-id
       |  +--rw source-tp?     leafref
       +--rw destination
       |  +--rw dest-node?   -> ../../../nw:node/node-id
       |  +--rw dest-tp?     leafref
       +--rw supporting-link* [network-ref link-ref]
          +--rw network-ref
          |       -> ../../../nw:supporting-network/network-ref
          +--rw link-ref       leafref
  augment /nw:networks/nw:network/nw:node:
    +--rw termination-point* [tp-id]
       +--rw tp-id                              tp-id
       +--rw supporting-termination-point*
               [network-ref node-ref tp-ref]
          +--rw network-ref
          |       -> ../../../nw:supporting-node/network-ref
          +--rw node-ref
          |       -> ../../../nw:supporting-node/node-ref
          +--rw tp-ref         leafref
```

Figure 5: The Structure of the Abstract (Base) Network Topology Data Model

A node has a list of termination points that are used to terminate links. An example of a termination point might be a physical or logical port or, more generally, an interface.

Like a node, a termination point can in turn be supported by an underlying termination point, contained in the supporting node of the underlay network.

A link is identified by a link-id that uniquely identifies the link within a given topology. Links are point-to-point and unidirectional. Accordingly, a link contains a source and a destination. Both source and destination reference a corresponding node, as well as a termination point on that node. Similar to a node, a link can map onto one or more links (which are terminated by the corresponding underlay termination points) in an underlay topology. This is captured in the list "supporting-link".

4.3. Extending the Data Model

In order to derive a data model for a specific type of network, the base data model can be extended. This can be done roughly as follows: a new YANG module for the new network type is introduced. In this module, a number of augmentations are defined against the "ietf-network" and "ietf-network-topology" modules.

We start with augmentations against the "ietf-network" module. First, a new network type needs to be defined; this is done by defining a presence container that represents the new network type. The new network type is inserted, by means of augmentation, below the network-types container. Subsequently, data nodes for any node parameters that are specific to a network type are defined and augmented into the node list. The new data nodes can be defined as conditional ("when") on the presence of the corresponding network type in the containing network. In cases where there are any requirements or restrictions in terms of network hierarchies, such as when a network of a new network type requires a specific type of underlay network, it is possible to define corresponding constraints as well and augment the supporting-network list accordingly. However, care should be taken to avoid excessive definitions of constraints.

Subsequently, augmentations are defined against the "ietf-network-topology" module. Data nodes are defined for link parameters, as well as termination point parameters, that are specific to the new network type. Those data nodes are inserted via augmentation into the link and termination-point lists, respectively. Again, data nodes can be defined as conditional on the presence of the corresponding network type in the containing network, by adding a corresponding "when" statement.

It is possible, but not required, to group data nodes for a given network type under a dedicated container. Doing so introduces additional structure but lengthens data node path names.

In cases where a hierarchy of network types is defined, augmentations can in turn be applied against augmenting modules, with the module of a network whose type is more specific augmenting the module of a network whose type is more general.

4.4. Discussion and Selected Design Decisions

4.4.1. Container Structure

Rather than maintaining lists in separate containers, the data model is kept relatively flat in terms of its containment structure. Lists of nodes, links, termination points, and supporting nodes; supporting links; and supporting termination points are not kept in separate containers. Therefore, path identifiers that are used to refer to specific nodes -- in management operations or in specifications of constraints -- can remain relatively compact. Of course, this means that there is no separate structure in instance information that separates elements of different lists from one another. Such a structure is semantically not required, but it might provide enhanced "human readability" in some cases.

4.4.2. Underlay Hierarchies and Mappings

To minimize assumptions regarding what a particular entity might actually represent, mappings between networks, nodes, links, and termination points are kept strictly generic. For example, no assumptions are made regarding whether a termination point actually refers to an interface or whether a node refers to a specific "system" or device; the data model at this generic level makes no provisions for these.

Where additional specifics about mappings between upper and lower layers are required, the information can be captured in augmenting modules. For example, to express that a termination point in a particular network type maps to an interface, an augmenting module can introduce an augmentation to the termination point. The augmentation introduces a leaf of type "interface-ref". That leaf references the corresponding interface [RFC8343]. Similarly, if a node maps to a particular device or network element, an augmenting module can augment the node data with a leaf that references the network element.

It is possible for links at one level of a hierarchy to map to multiple links at another level of the hierarchy. For example, a VPN topology might model VPN tunnels as links. Where a VPN tunnel maps to a path that is composed of a chain of several links, the link will contain a list of those supporting links. Likewise, it is possible for a link at one level of a hierarchy to aggregate a bundle of links at another level of the hierarchy.

4.4.3. Dealing with Changes in Underlay Networks

It is possible for a network to undergo churn even as other networks are layered on top of it. When a supporting node, link, or termination point is deleted, the supporting leafrefs in the overlay will be left dangling. To allow for this possibility, the data model makes use of the "require-instance" construct of YANG 1.1 [RFC7950].

A dangling leafref of a configured object leaves the corresponding instance in a state in which it lacks referential integrity, effectively rendering it nonoperational. Any corresponding object instance is therefore removed from the operational state datastore until the situation has been resolved, i.e., until either (1) the supporting object is added to the operational state datastore or (2) the instance is reconfigured to refer to another object that is actually reflected in the operational state datastore. It will remain part of the intended datastore.

It is the responsibility of the application maintaining the overlay to deal with the possibility of churn in the underlay network. When a server receives a request to configure an overlay network, it SHOULD validate whether supporting nodes / links / termination points refer to nodes in the underlay that actually exist, i.e., verify that the nodes are reflected in the operational state datastore. Configuration requests in which supporting nodes / links / termination points refer to objects currently not in existence SHOULD be rejected. It is the responsibility of the application to update the overlay when a supporting node / link / termination point is deleted at a later point in time. For this purpose, an application might subscribe to updates when changes to the underlay occur -- for example, using mechanisms defined in [YANG-Push].

4.4.4. Use of Groupings

The data model makes use of groupings instead of simply defining data nodes "inline". This makes it easier to include the corresponding data nodes in notifications, which then do not need to respecify each data node that is to be included. The trade-off is that it makes the specification of constraints more complex, because constraints involving data nodes outside the grouping need to be specified in

conjunction with a "uses" statement where the grouping is applied. This also means that constraints and XML Path Language (XPath) statements need to be specified in such a way that they navigate "down" first and select entire sets of nodes, as opposed to being able to simply specify them against individual data nodes.

4.4.5. Cardinality and Directionality of Links

The topology data model includes links that are point-to-point and unidirectional. It does not directly support multipoint and bidirectional links. Although this may appear as a limitation, the decision to do so keeps the data model simple and generic, and it allows it to be very easily subjected to applications that make use of graph algorithms. Bidirectional connections can be represented through pairs of unidirectional links. Multipoint networks can be represented through pseudonodes (similar to IS-IS, for example). By introducing hierarchies of nodes with nodes at one level mapping onto a set of other nodes at another level and by introducing new links for nodes at that level, topologies with connections representing non-point-to-point communication patterns can be represented.

4.4.6. Multihoming and Link Aggregation

Links are terminated by a single termination point, not sets of termination points. Connections involving multihoming or link aggregation schemes need to be represented using multiple point-to-point links and then defining a link at a higher layer that is supported by those individual links.

4.4.7. Mapping Redundancy

In a hierarchy of networks, there are nodes mapping to nodes, links mapping to links, and termination points mapping to termination points. Some of this information is redundant. Specifically, if the mapping of a link to one or more other links is known and the termination points of each link are known, the mapping information for the termination points can be derived via transitive closure and does not have to be explicitly configured. Nonetheless, in order to not constrain applications regarding which mappings they want to configure and which should be derived, the data model provides the option to configure this information explicitly. The data model includes integrity constraints to allow for validating for consistency.

4.4.8. Typing

A network's network types are represented using a container that contains a data node for each of its network types. A network can encompass several types of networks simultaneously; hence, a container is used instead of a case construct, with each network type in turn represented by a dedicated presence container. The reason for not simply using an empty leaf, or (even more simply) even doing away with the network container and just using a leaf-list of "network-type" instead, is to be able to represent "class hierarchies" of network types, with one network type "refining" the other. Containers specific to a network type are to be defined in the network-specific modules, augmenting the network-types container.

4.4.9. Representing the Same Device in Multiple Networks

One common requirement concerns the ability to indicate that the same device can be part of multiple networks and topologies. However, the data model defines a node as relative to the network that contains it. The same node cannot be part of multiple topologies. In many cases, a node will be the abstraction of a particular device in a network. To reflect that the same device is part of multiple topologies, the following approach might be chosen: a new type of network to represent a "physical" (or "device") network is introduced, with nodes representing devices. This network forms an underlay network for logical networks above it, with nodes of the logical network mapping onto nodes in the physical network.

This scenario is depicted in Figure 6. This figure depicts three networks with two nodes each. A physical network ("P" in the figure) consists of an inventory of two nodes (D1 and D2), each representing a device. A second network, X, has a third network, Y, as its underlay. Both X and Y also have the physical network (P) as their underlay. X1 has both Y1 and D1 as underlay nodes, while Y1 has D1 as its underlay node. Likewise, X2 has both Y2 and D2 as underlay nodes, while Y2 has D2 as its underlay node. The fact that X1 and Y1 are both instantiated on the same physical node (D1) can be easily seen.

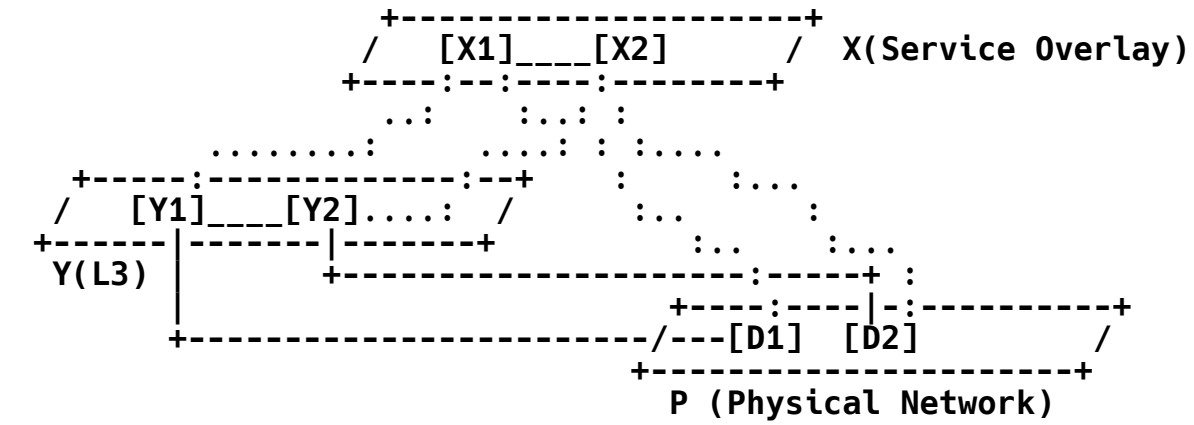

Figure 6: Topology Hierarchy Example - Multiple Underlays

In the case of a physical network, nodes represent physical devices and termination points represent physical ports. It should be noted that it is also possible to augment the data model for a physical network type, defining augmentations that have nodes reference system information and termination points reference physical interfaces, in order to provide a bridge between network and device models.

4.4.10. Supporting Client-Configured and System-Controlled Network Topologies

YANG requires data nodes to be designated as either configuration data ("config true") or operational data ("config false"), but not both, yet it is important to have all network information, including vertical cross-network dependencies, captured in one coherent data model. In most cases, network topology information about a network is discovered; the topology is considered a property of the network that is reflected in the data model. That said, certain types of topologies need to also be configurable by an application, e.g., in the case of overlay topologies.

The YANG data model for network topologies designates all data as "config true". The distinction between data that is actually configured and data that is in effect, including network data that is discovered, is provided through the datastores introduced as part of the Network Management Datastore Architecture (NMDA) [RFC8342]. Network topology data that is discovered is automatically populated as part of the operational state datastore, i.e., <operational>. It is "system controlled". Network topology that is configured is instantiated as part of a configuration datastore, e.g., <intended>. Only when it has actually taken effect will it also be instantiated as part of the operational state datastore, i.e., <operational>.

In general, a configured network topology will refer to an underlay topology and include layering information, such as the supporting node(s) underlying a node, supporting link(s) underlying a link, and supporting termination point(s) underlying a termination point. The supporting objects must be instantiated in the operational state datastore in order for the dependent overlay object to be reflected in the operational state datastore. Should a configured data item (such as a node) have a dangling reference that refers to a nonexistent data item (such as a supporting node), the configured data item will automatically be removed from <operational> and show up only in <intended>. It will be up to the client application to resolve the situation and ensure that the reference to the supporting resources is configured properly.

For each network, the origin of its data is indicated per the "origin" metadata [RFC7952] annotation defined in [RFC8342]. In general, the origin of discovered network data is "learned"; the origin of configured network data is "intended".

4.4.11. Identifiers of String or URI Type

The current data model defines identifiers of nodes, networks, links, and termination points as URIs. Alternatively, they could have been defined as strings.

The case for strings is that they will be easier to implement. The reason for choosing URIs is that the topology / node / termination point exists in a larger context; hence, it is useful to be able to correlate identifiers across systems. Although strings -- being the universal data type -- are easier for human beings, they also muddle things. What typically happens is that strings have some structure that is magically assigned, and the knowledge of this structure has to be communicated to each system working with the data. A URI makes the structure explicit and also attaches additional semantics: the URI, unlike a free-form string, can be fed into a URI resolver, which can point to additional resources associated with the URI. This property is important when the topology data is integrated into a larger and more complex system.

5. Interactions with Other YANG Modules

The data model makes use of data types that have been defined in [RFC6991].

This is a protocol-independent YANG data model with topology information. It is separate from, and not linked with, data models that are used to configure routing protocols or routing information. This includes, for example, the "ietf-routing" YANG module [RFC8022].

The data model obeys the requirements for the ephemeral state as specified in [RFC8242]. For ephemeral topology data that is system controlled, the process tasked with maintaining topology information will load information from the routing process (such as OSPF) into the operational state datastore without relying on a configuration datastore.

6. YANG Modules

6.1. Defining the Abstract Network: ietf-network

<CODE BEGINS> file "ietf-network@2018-02-26.yang"

```
module ietf-network {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network";
  prefix nw;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  organization
    "IETF I2RS (Interface to the Routing System) Working Group";

  contact
    "WG Web:    <https://datatracker.ietf.org/wg/i2rs/>
     WG List:   <mailto:i2rs@ietf.org>

     Editor:    Alexander Clemm
                <mailto:ludwig@clemm.org>

     Editor:    Jan Medved
                <mailto:jmedved@cisco.com>

     Editor:    Robert Varga
                <mailto:robert.varga@pantheon.tech>

     Editor:    Nitin Bahadur
                <mailto:nitin_bahadur@yahoo.com>

     Editor:    Hariharan Ananthakrishnan
                <mailto:hari@packetdesign.com>

     Editor:    Xufeng Liu
                <mailto:xufeng.liu.ietf@gmail.com>";
```

```
  description
    "This module defines a common base data model for a collection
     of nodes in a network.  Node definitions are further used
     in network topologies and inventories.

     Copyright (c) 2018 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 8345;
     see the RFC itself for full legal notices.";

  revision 2018-02-26 {
    description
      "Initial revision.";
    reference
      "RFC 8345: A YANG Data Model for Network Topologies";
  }

  typedef node-id {
    type inet:uri;
    description
      "Identifier for a node.  The precise structure of the node-id
       will be up to the implementation.  For example, some
       implementations MAY pick a URI that includes the network-id
       as part of the path.  The identifier SHOULD be chosen
       such that the same node in a real network topology will
       always be identified through the same identifier, even if
       the data model is instantiated in separate datastores.  An
       implementation MAY choose to capture semantics in the
       identifier -- for example, to indicate the type of node.";
  }
```

```
  typedef network-id {
    type inet:uri;
    description
      "Identifier for a network.  The precise structure of the
       network-id will be up to the implementation.  The identifier
       SHOULD be chosen such that the same network will always be
       identified through the same identifier, even if the data model
       is instantiated in separate datastores.  An implementation MAY
       choose to capture semantics in the identifier -- for example,
       to indicate the type of network.";
  }

  grouping network-ref {
    description
      "Contains the information necessary to reference a network --
       for example, an underlay network.";
    leaf network-ref {
      type leafref {
        path "/nw:networks/nw:network/nw:network-id";
      require-instance false;
      }
      description
        "Used to reference a network -- for example, an underlay
         network.";
    }
  }

  grouping node-ref {
    description
      "Contains the information necessary to reference a node.";
    leaf node-ref {
      type leafref {
        path "/nw:networks/nw:network[nw:network-id=current()/../"+
          "network-ref]/nw:node/nw:node-id";
        require-instance false;
      }
      description
        "Used to reference a node.
         Nodes are identified relative to the network that
         contains them.";
    }
    uses network-ref;
  }
```

```
container networks {
  description
    "Serves as a top-level container for a list of networks.";
  list network {
    key "network-id";
    description
      "Describes a network.
       A network typically contains an inventory of nodes,
       topological information (augmented through the
       network-topology data model), and layering information.";
    leaf network-id {
      type network-id;
      description
        "Identifies a network.";
    }
    container network-types {
      description
        "Serves as an augmentation target.
         The network type is indicated through corresponding
         presence containers augmented into this container.";
    }
    list supporting-network {
      key "network-ref";
      description
        "An underlay network, used to represent layered network
         topologies.";
      leaf network-ref {
        type leafref {
          path "/nw:networks/nw:network/nw:network-id";
        require-instance false;
        }
        description
          "References the underlay network.";
      }
    }
```

```
        list node {
          key "node-id";
          description
            "The inventory of nodes of this network.";
          leaf node-id {
            type node-id;
            description
              "Uniquely identifies a node within the containing
               network.";
          }
          list supporting-node {
            key "network-ref node-ref";
            description
              "Represents another node that is in an underlay network
               and that supports this node.  Used to represent layering
               structure.";
            leaf network-ref {
              type leafref {
                path "../../../nw:supporting-network/nw:network-ref";
              require-instance false;
              }
              description
                "References the underlay network of which the
                 underlay node is a part.";
            }
            leaf node-ref {
              type leafref {
                path "/nw:networks/nw:network/nw:node/nw:node-id";
              require-instance false;
              }
              description
                "References the underlay node itself.";
            }
          }
        }
      }
    }
  }

  <CODE ENDS>
```

6.2. Creating Abstract Network Topology: ietf-network-topology

<CODE BEGINS> file "ietf-network-topology@2018-02-26.yang"

```
module ietf-network-topology {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-topology";
  prefix nt;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-network {
    prefix nw;
    reference
      "RFC 8345: A YANG Data Model for Network Topologies";
  }

  organization
    "IETF I2RS (Interface to the Routing System) Working Group";

  contact
    "WG Web:    <https://datatracker.ietf.org/wg/i2rs/>
     WG List:   <mailto:i2rs@ietf.org>

     Editor:    Alexander Clemm
                <mailto:ludwig@clemm.org>

     Editor:    Jan Medved
                <mailto:jmedved@cisco.com>

     Editor:    Robert Varga
                <mailto:robert.varga@pantheon.tech>

     Editor:    Nitin Bahadur
                <mailto:nitin_bahadur@yahoo.com>

     Editor:    Hariharan Ananthakrishnan
                <mailto:hari@packetdesign.com>

     Editor:    Xufeng Liu
                <mailto:xufeng.liu.ietf@gmail.com>";
```

```
  description
    "This module defines a common base model for a network topology,
     augmenting the base network data model with links to connect
     nodes, as well as termination points to terminate links
     on nodes.

     Copyright (c) 2018 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 8345;
     see the RFC itself for full legal notices.";

  revision 2018-02-26 {
    description
      "Initial revision.";
    reference
      "RFC 8345: A YANG Data Model for Network Topologies";
  }

  typedef link-id {
    type inet:uri;
    description
      "An identifier for a link in a topology.  The precise
       structure of the link-id will be up to the implementation.
       The identifier SHOULD be chosen such that the same link in a
       real network topology will always be identified through the
       same identifier, even if the data model is instantiated in
       separate datastores.  An implementation MAY choose to capture
       semantics in the identifier -- for example, to indicate the
       type of link and/or the type of topology of which the link is
       a part.";
  }

  typedef tp-id {
    type inet:uri;
    description
      "An identifier for termination points on a node.  The precise
       structure of the tp-id will be up to the implementation.
       The identifier SHOULD be chosen such that the same termination
       point in a real network topology will always be identified
       through the same identifier, even if the data model is
```

```
        instantiated in separate datastores.  An implementation MAY
        choose to capture semantics in the identifier -- for example,
        to indicate the type of termination point and/or the type of
        node that contains the termination point.";
    }

    grouping link-ref {
      description
        "This grouping can be used to reference a link in a specific
         network.  Although it is not used in this module, it is
         defined here for the convenience of augmenting modules.";
      leaf link-ref {
        type leafref {
          path "/nw:networks/nw:network[nw:network-id=current()/../"+
            "network-ref]/nt:link/nt:link-id";
          require-instance false;
        }
        description
          "A type for an absolute reference to a link instance.
           (This type should not be used for relative references.
           In such a case, a relative path should be used instead.)";
      }
      uses nw:network-ref;
    }

    grouping tp-ref {
      description
        "This grouping can be used to reference a termination point
         in a specific node.  Although it is not used in this module,
         it is defined here for the convenience of augmenting
         modules.";
      leaf tp-ref {
        type leafref {
          path "/nw:networks/nw:network[nw:network-id=current()/../"+
            "network-ref]/nw:node[nw:node-id=current()/../"+
            "node-ref]/nt:termination-point/nt:tp-id";
          require-instance false;
        }
        description
          "A type for an absolute reference to a termination point.
           (This type should not be used for relative references.
           In such a case, a relative path should be used instead.)";
      }
      uses nw:node-ref;
    }
```

```
     augment "/nw:networks/nw:network" {
       description
         "Add links to the network data model.";
       list link {
         key "link-id";
         description
           "A network link connects a local (source) node and
            a remote (destination) node via a set of the respective
            node's termination points.  It is possible to have several
            links between the same source and destination nodes.
            Likewise, a link could potentially be re-homed between
            termination points.  Therefore, in order to ensure that we
            would always know to distinguish between links, every link
            is identified by a dedicated link identifier.  Note that a
            link models a point-to-point link, not a multipoint link.";
         leaf link-id {
           type link-id;
           description
             "The identifier of a link in the topology.
              A link is specific to a topology to which it belongs.";
         }
         container source {
           description
             "This container holds the logical source of a particular
              link.";
           leaf source-node {
             type leafref {
               path "../../../nw:node/nw:node-id";
               require-instance false;
             }
             description
               "Source node identifier.  Must be in the same topology.";
           }
           leaf source-tp {
             type leafref {
               path "../../../nw:node[nw:node-id=current()/../"+
                 "source-node]/termination-point/tp-id";
               require-instance false;
             }
             description
               "This termination point is located within the source node
                and terminates the link.";
           }
         }
```

```
        container destination {
          description
            "This container holds the logical destination of a
             particular link.";
          leaf dest-node {
            type leafref {
              path "../../../nw:node/nw:node-id";
            require-instance false;
            }
            description
              "Destination node identifier.  Must be in the same
               network.";
          }
          leaf dest-tp {
            type leafref {
              path "../../../nw:node[nw:node-id=current()/../"+
                "dest-node]/termination-point/tp-id";
              require-instance false;
            }
            description
              "This termination point is located within the
               destination node and terminates the link.";
          }
        }
        list supporting-link {
          key "network-ref link-ref";
          description
            "Identifies the link or links on which this link depends.";
          leaf network-ref {
            type leafref {
              path "../../../nw:supporting-network/nw:network-ref";
            require-instance false;
            }
            description
              "This leaf identifies in which underlay topology
               the supporting link is present.";
          }
```

```
        leaf link-ref {
          type leafref {
            path "/nw:networks/nw:network[nw:network-id=current()/"+
              "../network-ref]/link/link-id";
            require-instance false;
          }
          description
            "This leaf identifies a link that is a part
             of this link's underlay.  Reference loops in which
             a link identifies itself as its underlay, either
             directly or transitively, are not allowed.";
        }
      }
    }
  }
  augment "/nw:networks/nw:network/nw:node" {
    description
      "Augments termination points that terminate links.
       Termination points can ultimately be mapped to interfaces.";
    list termination-point {
      key "tp-id";
      description
        "A termination point can terminate a link.
         Depending on the type of topology, a termination point
         could, for example, refer to a port or an interface.";
      leaf tp-id {
        type tp-id;
        description
          "Termination point identifier.";
      }
      list supporting-termination-point {
        key "network-ref node-ref tp-ref";
        description
          "This list identifies any termination points on which a
           given termination point depends or onto which it maps.
           Those termination points will themselves be contained
           in a supporting node.  This dependency information can be
           inferred from the dependencies between links.  Therefore,
           this item is not separately configurable.  Hence, no
           corresponding constraint needs to be articulated.
           The corresponding information is simply provided by the
           implementing system.";
```

```
            leaf network-ref {
              type leafref {
                path "../../../nw:supporting-node/nw:network-ref";
              require-instance false;
              }
              description
                "This leaf identifies in which topology the
                 supporting termination point is present.";
            }
            leaf node-ref {
              type leafref {
                path "../../../nw:supporting-node/nw:node-ref";
              require-instance false;
              }
              description
                "This leaf identifies in which node the supporting
                 termination point is present.";
            }
            leaf tp-ref {
              type leafref {
                path "/nw:networks/nw:network[nw:network-id=current()/"+
                  "../network-ref]/nw:node[nw:node-id=current()/../"+
                  "node-ref]/termination-point/tp-id";
              require-instance false;
              }
              description
                "Reference to the underlay node (the underlay node must
                 be in a different topology).";
            }
          }
        }
      }
    }
```

<CODE ENDS>

7. IANA Considerations

This document registers the following namespace URIs in the "IETF XML Registry" [RFC3688]:

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

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

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

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

This document registers the following YANG modules in the "YANG Module Names" registry [RFC6020]:

Name: ietf-network
Namespace: urn:ietf:params:xml:ns:yang:ietf-network
Prefix: nw
Reference: RFC 8345

Name: ietf-network-topology
Namespace: urn:ietf:params:xml:ns:yang:ietf-network-topology
Prefix: nt
Reference: RFC 8345

Name: ietf-network-state
Namespace: urn:ietf:params:xml:ns:yang:ietf-network-state
Prefix: nw-s
Reference: RFC 8345

Name: ietf-network-topology-state
Namespace: urn:ietf:params:xml:ns:yang:ietf-network-topology-state
Prefix: nt-s
Reference: RFC 8345

8. Security Considerations

The YANG modules specified in this document define 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 [RFC5246].

The NETCONF access control model [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

The network topology and inventory created by these modules reveal information about the structure of networks that could be very helpful to an attacker. As a privacy consideration, although there is no personally identifiable information defined in these modules, it is possible that some node identifiers may be associated with devices that are in turn associated with specific users.

The YANG modules define information that can be configurable in certain instances -- for example, in the case of overlay topologies that can be created by client applications. In such cases, a malicious client could introduce topologies that are undesired. Specifically, a malicious client could attempt to remove or add a node, a link, or a termination point by creating or deleting corresponding elements in node, link, or termination point lists, respectively. In the case of a topology that is learned, the server will automatically prohibit such misconfiguration attempts. In the case of a topology that is configured, i.e., whose origin is "intended", the undesired configuration could become effective and be reflected in the operational state datastore, leading to disruption of services provided via this topology. For example, the topology could be "cut" or could be configured in a suboptimal way, leading to increased consumption of resources in the underlay network due to the routing and bandwidth utilization inefficiencies that would result. Likewise, it could lead to degradation of service levels as well as possible disruption of service. For those reasons, it is important that the NETCONF access control model be vigorously applied to prevent topology misconfiguration by unauthorized clients.

There are a number of data nodes defined in these 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)

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:

In the "ietf-network" module:

- network: A malicious client could attempt to remove or add a network in an effort to remove an overlay topology or to create an unauthorized overlay.
- supporting network: A malicious client could attempt to disrupt the logical structure of the model, resulting in a lack of overall data integrity and making it more difficult to, for example, troubleshoot problems rooted in the layering of network topologies.
- node: A malicious client could attempt to remove or add a node from the network -- for example, in order to sabotage the topology of a network overlay.
- supporting node: A malicious client could attempt to change the supporting node in order to sabotage the layering of an overlay.

In the "ietf-network-topology" module:

- link: A malicious client could attempt to remove a link from a topology, add a new link, manipulate the way the link is layered over supporting links, or modify the source or destination of the link. In each case, the structure of the topology would be sabotaged, and this scenario could, for example, result in an overlay topology that is less than optimal.
- termination point: A malicious client could attempt to remove termination points from a node, add "phantom" termination points to a node, or change the layering dependencies of termination points, again in an effort to sabotage the integrity of a topology and potentially disrupt orderly operations of an overlay.

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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[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>.

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

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

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

9.2. Informative References

[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>.

[RFC2328] Moy, J., "OSPF Version 2", STD 54, RFC 2328, DOI 10.17487/RFC2328, April 1998, <https://www.rfc-editor.org/info/rfc2328>.

[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>.

[RFC3444] Pras, A. and J. Schoenwaelder, "On the Difference between Information Models and Data Models", RFC 3444, DOI 10.17487/RFC3444, January 2003, <https://www.rfc-editor.org/info/rfc3444>.

[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>.

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, DOI 10.17487/RFC7952, August 2016, <https://www.rfc-editor.org/info/rfc7952>.

[RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, DOI 10.17487/RFC8022, November 2016, <https://www.rfc-editor.org/info/rfc8022>.

[RFC8242] Haas, J. and S. Hares, "Interface to the Routing System (I2RS) Ephemeral State Requirements", RFC 8242, DOI 10.17487/RFC8242, September 2017, <https://www.rfc-editor.org/info/rfc8242>.

[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>.

[RFC8346] Clemm, A., Medved, J., Varga, R., Liu, X., Ananthakrishnan, H., and N. Bahadur, "A YANG Data Model for Layer 3 Topologies", RFC 8346, DOI 10.17487/RFC8346, March 2018, <https://www.rfc-editor.org/info/rfc8346>.

[USECASE-REQS] Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", Work in Progress, draft-ietf-i2rs-usecase-reqs-summary-03, November 2016.

[YANG-Push] Clemm, A., Voit, E., Gonzalez Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", Work in Progress, draft-ietf-netconf-yang-push-15, February 2018.

Appendix A. Model Use Cases

A.1. Fetching Topology from a Network Element

In its simplest form, topology is learned by a network element (e.g., a router) through its participation in peering protocols (IS-IS, BGP, etc.). This learned topology can then be exported (e.g., to a Network Management System) for external utilization. Typically, any network element in a domain can be queried for its topology and be expected to return the same result.

In a slightly more complex form, the network element may be a controller. It could be a network element with satellite or subtended devices hanging off of it, or it could be a controller in the more classical sense -- that is, a special device designated to orchestrate the activities of a number of other devices (e.g., an Optical Controller). In this case, the controller device is logically a singleton and must be queried distinctly.

It is worth noting that controllers can be built on top of other controllers to establish a topology incorporating all of the domains within an entire network.

In all of the cases above, the topology learned by the network element is considered to be operational state data. That is, the data is accumulated purely by the network element's interactions with other systems and is subject to change dynamically without input or consent.

A.2. Modifying TE Topology Imported from an Optical Controller

Consider a scenario where an Optical Controller presents its topology, in abstract TE terms, to a client packet controller. This customized topology (which gets merged into the client's native topology) contains sufficient information for the path-computing client to select paths across the optical domain according to its policies. If the client determines (at any given point in time) that this imported topology does not cater exactly to its requirements, it may decide to request modifications to the topology. Such customization requests may include the addition or deletion of topological elements or the modification of attributes associated with existing topological elements. From the perspective of the Optical Controller, these requests translate into configuration changes to the exported abstract topology.

A.3. Annotating Topology for Local Computation

In certain scenarios, the topology learned by a controller needs to be augmented with additional attributes before running a computation algorithm on it. Consider the case where a path-computation application on the controller needs to take the geographic coordinates of the nodes into account while computing paths on the learned topology. If the learned topology does not contain these coordinates, then these additional attributes must be configured on the corresponding topological elements.

A.4. SDN Controller-Based Configuration of Overlays on Top of Underlays

In this scenario, an SDN Controller (for example, Open Daylight) maintains a view of the topology of the network that it controls based on information that it discovers from the network. In addition, it provides an application in which it configures and maintains an overlay topology.

The SDN Controller thus maintains two roles:

- It is a client to the network.
- It is a server to its own northbound applications and clients, e.g., an Operations Support System (OSS).

In other words, one system's client (or controller, in this case) may be another system's server (or managed system).

In this scenario, the SDN Controller maintains a consolidated data model of multiple layers of topology. This includes the lower layers of the network topology, built from information that is discovered from the network. It also includes upper layers of topology overlay, configurable by the controller's client, i.e., the OSS. To the OSS, the lower topology layers constitute "read-only" information. The upper topology layers need to be read-writable.

Appendix B. Companion YANG Data Models for Implementations Not Compliant with NMDA

The YANG modules defined in this document are designed to be used in conjunction with implementations that support the Network Management Datastore Architecture (NMDA) as defined in [RFC8342]. In order to allow implementations to use the data model even in cases when NMDA is not supported, the following two companion modules -- "ietf-network-state" and "ietf-network-topology-state" -- are defined; they represent the operational state of networks and network topologies, respectively. These modules mirror the "ietf-network"

and "ietf-network-topology" modules (defined in Sections 6.1 and 6.2 of this document); however, in the case of these modules, all data nodes are non-configurable. They represent state that comes into being by either (1) learning topology information from the network or (2) applying configuration from the mirrored modules.

The "ietf-network-state" and "ietf-network-topology-state" companion modules are redundant and SHOULD NOT be supported by implementations that support NMDA; therefore, we define these modules in Appendices B.1 and B.2 (below) instead of the main body of this document.

As the structure of both modules mirrors that of their underlying modules, the YANG tree is not depicted separately.

B.1. YANG Module for Network State

<CODE BEGINS> file "ietf-network-state@2018-02-26.yang"

```
module ietf-network-state {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-state";
  prefix nw-s;

  import ietf-network {
    prefix nw;
    reference
      "RFC 8345: A YANG Data Model for Network Topologies";
  }

  organization
    "IETF I2RS (Interface to the Routing System) Working Group";

  contact
    "WG Web:    <https://datatracker.ietf.org/wg/i2rs/>
     WG List:   <mailto:i2rs@ietf.org>

     Editor:    Alexander Clemm
                <mailto:ludwig@clemm.org>

     Editor:    Jan Medved
                <mailto:jmedved@cisco.com>

     Editor:    Robert Varga
                <mailto:robert.varga@pantheon.tech>

     Editor:    Nitin Bahadur
                <mailto:nitin_bahadur@yahoo.com>
```

```
     Editor:    Hariharan Ananthakrishnan
                <mailto:hari@packetdesign.com>

     Editor:    Xufeng Liu
                <mailto:xufeng.liu.ietf@gmail.com>";

  description
    "This module defines a common base data model for a collection
     of nodes in a network.  Node definitions are further used
     in network topologies and inventories.  It represents
     information that either (1) is learned and automatically
     populated or (2) results from applying network information
     that has been configured per the 'ietf-network' data model,
     mirroring the corresponding data nodes in this data model.

     The data model mirrors 'ietf-network' but contains only
     read-only state data.  The data model is not needed when the
     underlying implementation infrastructure supports the Network
     Management Datastore Architecture (NMDA).

     Copyright (c) 2018 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 8345;
     see the RFC itself for full legal notices.";

  revision 2018-02-26 {
    description
      "Initial revision.";
    reference
      "RFC 8345: A YANG Data Model for Network Topologies";
  }
```

```
  grouping network-ref {
    description
      "Contains the information necessary to reference a network --
       for example, an underlay network.";
    leaf network-ref {
      type leafref {
        path "/nw-s:networks/nw-s:network/nw-s:network-id";
      require-instance false;
      }
      description
        "Used to reference a network -- for example, an underlay
         network.";
    }
  }

  grouping node-ref {
    description
      "Contains the information necessary to reference a node.";
    leaf node-ref {
      type leafref {
        path "/nw-s:networks/nw-s:network[nw-s:network-id=current()"+
          "/../network-ref]/nw-s:node/nw-s:node-id";
        require-instance false;
      }
      description
        "Used to reference a node.
         Nodes are identified relative to the network that
         contains them.";
    }
    uses network-ref;
  }
```

```
  container networks {
    config false;
    description
      "Serves as a top-level container for a list of networks.";
    list network {
      key "network-id";
      description
        "Describes a network.
         A network typically contains an inventory of nodes,
         topological information (augmented through the
         network-topology data model), and layering information.";
      container network-types {
        description
          "Serves as an augmentation target.
           The network type is indicated through corresponding
           presence containers augmented into this container.";
      }
      leaf network-id {
        type nw:network-id;
        description
          "Identifies a network.";
      }
      list supporting-network {
        key "network-ref";
        description
          "An underlay network, used to represent layered network
           topologies.";
        leaf network-ref {
          type leafref {
            path "/nw-s:networks/nw-s:network/nw-s:network-id";
          require-instance false;
          }
          description
            "References the underlay network.";
        }
      }
```

```
      list node {
        key "node-id";
        description
          "The inventory of nodes of this network.";
        leaf node-id {
          type nw:node-id;
          description
            "Uniquely identifies a node within the containing
             network.";
        }
        list supporting-node {
          key "network-ref node-ref";
          description
            "Represents another node that is in an underlay network
             and that supports this node.  Used to represent layering
             structure.";
          leaf network-ref {
            type leafref {
              path "../../../nw-s:supporting-network/nw-s:network-ref";
            require-instance false;
            }
            description
              "References the underlay network of which the
               underlay node is a part.";
          }
          leaf node-ref {
            type leafref {
              path "/nw-s:networks/nw-s:network/nw-s:node/nw-s:node-id";
            require-instance false;
            }
            description
              "References the underlay node itself.";
          }
        }
      }
    }
  }
}
```

<CODE ENDS>

B.2. YANG Module for Network Topology State

```
  <CODE BEGINS> file "ietf-network-topology-state@2018-02-26.yang"

  module ietf-network-topology-state {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:ietf-network-topology-state";
    prefix nt-s;

    import ietf-network-state {
      prefix nw-s;
      reference
        "RFC 8345: A YANG Data Model for Network Topologies";
    }
    import ietf-network-topology {
      prefix nt;
      reference
        "RFC 8345: A YANG Data Model for Network Topologies";
    }

    organization
      "IETF I2RS (Interface to the Routing System) Working Group";

    contact
      "WG Web:    <https://datatracker.ietf.org/wg/i2rs/>
       WG List:   <mailto:i2rs@ietf.org>

       Editor:    Alexander Clemm
                  <mailto:ludwig@clemm.org>

       Editor:    Jan Medved
                  <mailto:jmedved@cisco.com>

       Editor:    Robert Varga
                  <mailto:robert.varga@pantheon.tech>

       Editor:    Nitin Bahadur
                  <mailto:nitin_bahadur@yahoo.com>

       Editor:    Hariharan Ananthakrishnan
                  <mailto:hari@packetdesign.com>

       Editor:    Xufeng Liu
                  <mailto:xufeng.liu.ietf@gmail.com>";
```

```
    description
      "This module defines a common base data model for network
       topology state, representing topology that either (1) is learned
       or (2) results from applying topology that has been configured
       per the 'ietf-network-topology' data model, mirroring the
       corresponding data nodes in this data model.  It augments the
       base network state data model with links to connect nodes, as
       well as termination points to terminate links on nodes.

       The data model mirrors 'ietf-network-topology' but contains only
       read-only state data.  The data model is not needed when the
       underlying implementation infrastructure supports the Network
       Management Datastore Architecture (NMDA).

       Copyright (c) 2018 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 8345;
       see the RFC itself for full legal notices.";

    revision 2018-02-26 {
      description
        "Initial revision.";
      reference
        "RFC 8345: A YANG Data Model for Network Topologies";
    }
```

```
    grouping link-ref {
      description
        "References a link in a specific network.  Although this
         grouping is not used in this module, it is defined here for
         the convenience of augmenting modules.";
      leaf link-ref {
        type leafref {
          path "/nw-s:networks/nw-s:network[nw-s:network-id=current()"+
            "/../network-ref]/nt-s:link/nt-s:link-id";
          require-instance false;
        }
        description
          "A type for an absolute reference to a link instance.
           (This type should not be used for relative references.
           In such a case, a relative path should be used instead.)";
      }
      uses nw-s:network-ref;
    }

    grouping tp-ref {
      description
        "References a termination point in a specific node.  Although
         this grouping is not used in this module, it is defined here
         for the convenience of augmenting modules.";
      leaf tp-ref {
        type leafref {
          path "/nw-s:networks/nw-s:network[nw-s:network-id=current()"+
            "/../network-ref]/nw-s:node[nw-s:node-id=current()/../"+
            "node-ref]/nt-s:termination-point/nt-s:tp-id";
          require-instance false;
        }
        description
          "A type for an absolute reference to a termination point.
           (This type should not be used for relative references.
           In such a case, a relative path should be used instead.)";
      }
      uses nw-s:node-ref;
    }

    augment "/nw-s:networks/nw-s:network" {
      description
        "Add links to the network data model.";
      list link {
        key "link-id";
        description
          "A network link connects a local (source) node and
           a remote (destination) node via a set of the respective
           node's termination points.  It is possible to have several
```

```
          links between the same source and destination nodes.
          Likewise, a link could potentially be re-homed between
          termination points.  Therefore, in order to ensure that we
          would always know to distinguish between links, every link
          is identified by a dedicated link identifier.  Note that a
          link models a point-to-point link, not a multipoint link.";
        container source {
          description
            "This container holds the logical source of a particular
             link.";
          leaf source-node {
            type leafref {
              path "../../../nw-s:node/nw-s:node-id";
              require-instance false;
            }
            description
              "Source node identifier.  Must be in the same topology.";
          }
          leaf source-tp {
            type leafref {
              path "../../../nw-s:node[nw-s:node-id=current()/../"+
                "source-node]/termination-point/tp-id";
              require-instance false;
            }
            description
              "This termination point is located within the source node
               and terminates the link.";
          }
        }
        container destination {
          description
            "This container holds the logical destination of a
             particular link.";
          leaf dest-node {
            type leafref {
              path "../../../nw-s:node/nw-s:node-id";
            require-instance false;
            }
            description
              "Destination node identifier.  Must be in the same
               network.";
          }
```

```
          leaf dest-tp {
            type leafref {
              path "../../../nw-s:node[nw-s:node-id=current()/../"+
                "dest-node]/termination-point/tp-id";
              require-instance false;
            }
            description
              "This termination point is located within the
               destination node and terminates the link.";
          }
        }
        leaf link-id {
          type nt:link-id;
          description
            "The identifier of a link in the topology.
             A link is specific to a topology to which it belongs.";
        }
        list supporting-link {
          key "network-ref link-ref";
          description
            "Identifies the link or links on which this link depends.";
          leaf network-ref {
            type leafref {
              path "../../../nw-s:supporting-network/nw-s:network-ref";
            require-instance false;
            }
            description
              "This leaf identifies in which underlay topology
               the supporting link is present.";
          }
          leaf link-ref {
            type leafref {
              path "/nw-s:networks/nw-s:network[nw-s:network-id="+
                "current()/../network-ref]/link/link-id";
              require-instance false;
            }
            description
              "This leaf identifies a link that is a part
               of this link's underlay.  Reference loops in which
               a link identifies itself as its underlay, either
               directly or transitively, are not allowed.";
          }
        }
      }
    }
```

```
    augment "/nw-s:networks/nw-s:network/nw-s:node" {
      description
        "Augments termination points that terminate links.
         Termination points can ultimately be mapped to interfaces.";
      list termination-point {
        key "tp-id";
        description
          "A termination point can terminate a link.
           Depending on the type of topology, a termination point
           could, for example, refer to a port or an interface.";
        leaf tp-id {
          type nt:tp-id;
          description
            "Termination point identifier.";
        }
        list supporting-termination-point {
          key "network-ref node-ref tp-ref";
          description
            "This list identifies any termination points on which a
             given termination point depends or onto which it maps.
             Those termination points will themselves be contained
             in a supporting node.  This dependency information can be
             inferred from the dependencies between links.  Therefore,
             this item is not separately configurable.  Hence, no
             corresponding constraint needs to be articulated.
             The corresponding information is simply provided by the
             implementing system.";
          leaf network-ref {
            type leafref {
              path "../../../nw-s:supporting-node/nw-s:network-ref";
            require-instance false;
            }
            description
              "This leaf identifies in which topology the
               supporting termination point is present.";
          }
          leaf node-ref {
            type leafref {
              path "../../../nw-s:supporting-node/nw-s:node-ref";
            require-instance false;
            }
            description
              "This leaf identifies in which node the supporting
               termination point is present.";
          }
```

```
          leaf tp-ref {
            type leafref {
              path "/nw-s:networks/nw-s:network[nw-s:network-id="+
                "current()/../network-ref]/nw-s:node[nw-s:node-id="+
                "current()/../node-ref]/termination-point/tp-id";
              require-instance false;
            }
            description
              "Reference to the underlay node (the underlay node must
               be in a different topology).";
          }
        }
      }
    }
  }
```

<CODE ENDS>

Appendix C. An Example

This section contains an example of an instance data tree in JSON encoding [RFC7951]. The example instantiates "ietf-network-topology" (and "ietf-network", which "ietf-network-topology" augments) for the topology depicted in Figure 7. There are three nodes: D1, D2, and D3. D1 has three termination points (1-0-1, 1-2-1, and 1-3-1). D2 has three termination points as well (2-1-1, 2-0-1, and 2-3-1). D3 has two termination points (3-1-1 and 3-2-1). In addition, there are six links, two between each pair of nodes with one going in each direction.

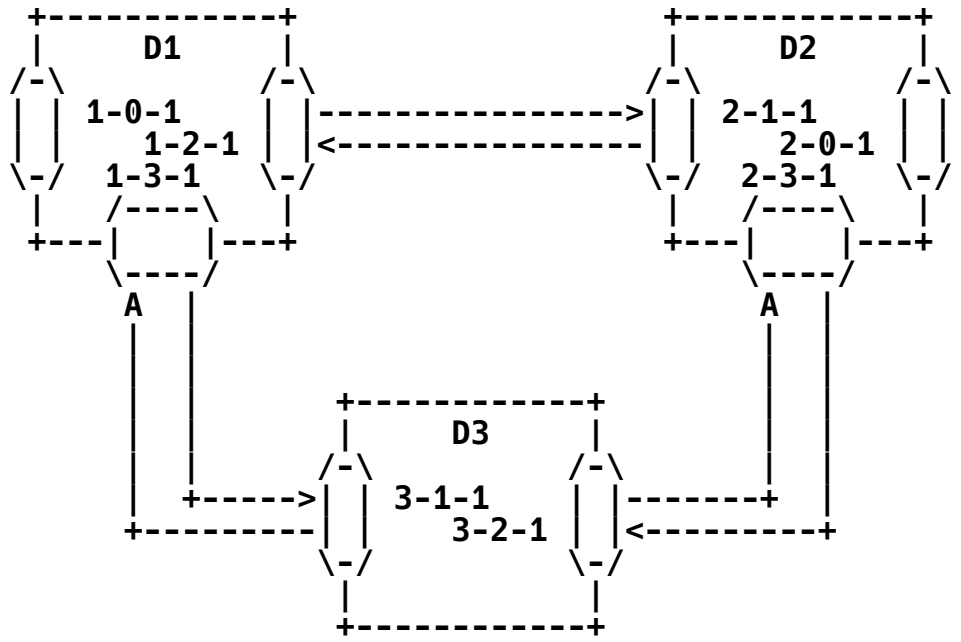

Figure 7: A Network Topology Example

The corresponding instance data tree is depicted in Figure 8:

```
{
  "ietf-network:networks": {
    "network": [
      {
        "network-types": {
        },
        "network-id": "otn-hc",
        "node": [
          {
            "node-id": "D1",
            "termination-point": [
              {
                "tp-id": "1-0-1"
              },
              {
                "tp-id": "1-2-1"
              },
              {
                "tp-id": "1-3-1"
              }
            ]
          },
          {
            "node-id": "D2",
            "termination-point": [
              {
                "tp-id": "2-0-1"
              },
              {
                "tp-id": "2-1-1"
              },
              {
                "tp-id": "2-3-1"
              }
            ]
          },
```

```
            {
              "node-id": "D3",
              "termination-point": [
                {
                  "tp-id": "3-1-1"
                },
                {
                  "tp-id": "3-2-1"
                }
              ]
            }
          ],
          "ietf-network-topology:link": [
            {
              "link-id": "D1,1-2-1,D2,2-1-1",
              "source": {
                "source-node": "D1",
                "source-tp": "1-2-1"
              }
              "destination": {
                "dest-node": "D2",
                "dest-tp": "2-1-1"
              }
            },
            {
              "link-id": "D2,2-1-1,D1,1-2-1",
              "source": {
                "source-node": "D2",
                "source-tp": "2-1-1"
              }
              "destination": {
                "dest-node": "D1",
                "dest-tp": "1-2-1"
              }
            },
            {
              "link-id": "D1,1-3-1,D3,3-1-1",
              "source": {
                "source-node": "D1",
                "source-tp": "1-3-1"
              }
              "destination": {
                "dest-node": "D3",
                "dest-tp": "3-1-1"
              }
            },
```

```
            {
              "link-id": "D3,3-1-1,D1,1-3-1",
              "source": {
                "source-node": "D3",
                "source-tp": "3-1-1"
              }
              "destination": {
                "dest-node": "D1",
                "dest-tp": "1-3-1"
              }
            },
            {
              "link-id": "D2,2-3-1,D3,3-2-1",
              "source": {
                "source-node": "D2",
                "source-tp": "2-3-1"
              }
              "destination": {
                "dest-node": "D3",
                "dest-tp": "3-2-1"
              }
            },
            {
              "link-id": "D3,3-2-1,D2,2-3-1",
              "source": {
                "source-node": "D3",
                "source-tp": "3-2-1"
              }
              "destination": {
                "dest-node": "D2",
                "dest-tp": "2-3-1"
              }
            }
          ]
        }
      ]
    }
  }
```

Figure 8: Instance Data Tree

Acknowledgments

We wish to acknowledge the helpful contributions, comments, and suggestions that were received from Alia Atlas, Andy Bierman, Martin Bjorklund, Igor Bryskin, Benoit Claise, Susan Hares, Ladislav Lhotka, Carlos Pignataro, Juergen Schoenwaelder, Robert Wilton, Qin Wu, and Xian Zhang.

Contributors

More people contributed to the data model presented in this paper than can be listed in the "Authors' Addresses" section. Additional contributors include:

- Vishnu Pavan Beeram, Juniper
- Ken Gray, Cisco
- Tom Nadeau, Brocade
- Tony Tkacik
- Kent Watsen, Juniper
- Aleksandr Zhdankin, Cisco

Authors' Addresses

Alexander Clemm
Huawei USA - Futurewei Technologies Inc.
Santa Clara, CA
United States of America

Email: ludwig@clemm.org, alexander.clemm@huawei.com

Jan Medved
Cisco

Email: jmedved@cisco.com

Robert Varga
Pantheon Technologies SRO

Email: robert.varga@pantheon.tech

Nitin Bahadur
Bracket Computing

Email: nitin_bahadur@yahoo.com

Hariharan Ananthakrishnan
Packet Design

Email: hari@packetdesign.com

Xufeng Liu
Jabil

Email: xufeng.liu.ietf@gmail.com