Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 2, 2017

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

March 1, 2017

A Data Model for Network Topologies
draft-ietf-i2rs-yang-network-topo-12.txt

Abstract

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

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 September 2, 2017.

Copyright Notice

Table of Contents

1. Introduction

This document introduces an abstract (base) YANG [RFC7950] [RFC6991] data model to represent networks and topologies. The data model is divided into two parts. The first part of the model defines a network model that allows to define network hierarchies (i.e. network stacks) and to maintain an inventory of nodes contained in a network. The second part of the model augments the basic network 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 model introduces vertical layering relationships between networks that can be augmented to cover both network inventories and network/service topologies.

While it would be possible to combine both parts into a single model, the separation facilitates integration of network topology and network inventory models, by allowing to augment network inventory information separately and without concern for topology into the network model.

The model can be augmented to describe specifics of particular types of networks and topologies. For example, an augmenting model can provide network node information with attributes that are specific to a particular network type. Examples of augmenting models include 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 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 models allow applications to operate on an inventory or topology of any network at a generic level, where specifics of particular inventory/topology types are not required. At the same time, where data specific to a network type does comes into play and the model is augmented, the instantiated data still adheres to the same structure and is represented in 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 "network.yang", 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 models with inventory and topology specific attributes. Network hierarchy (stack) allows any given network to have one or more "supporting networks". The relationship of the base network model, the inventory models and the topology models is shown in the following figure (dotted lines in the figure denote possible augmentations to models defined in this document).

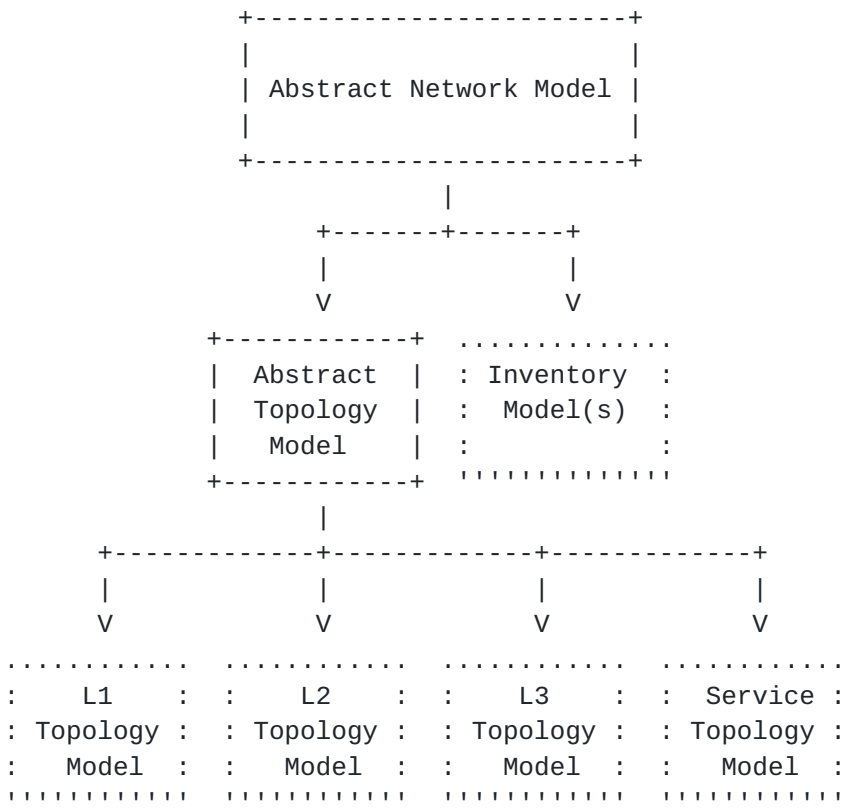

Figure 1: The network model structure

The network-topology YANG module introduced in this document, entitled "network-topology.yang", defines a generic topology 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 network.yang 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 model therefore allows to capture relationships between topologies, as well as dependencies between nodes and termination points across topologies. An example of a topology stack is shown in the following figure.

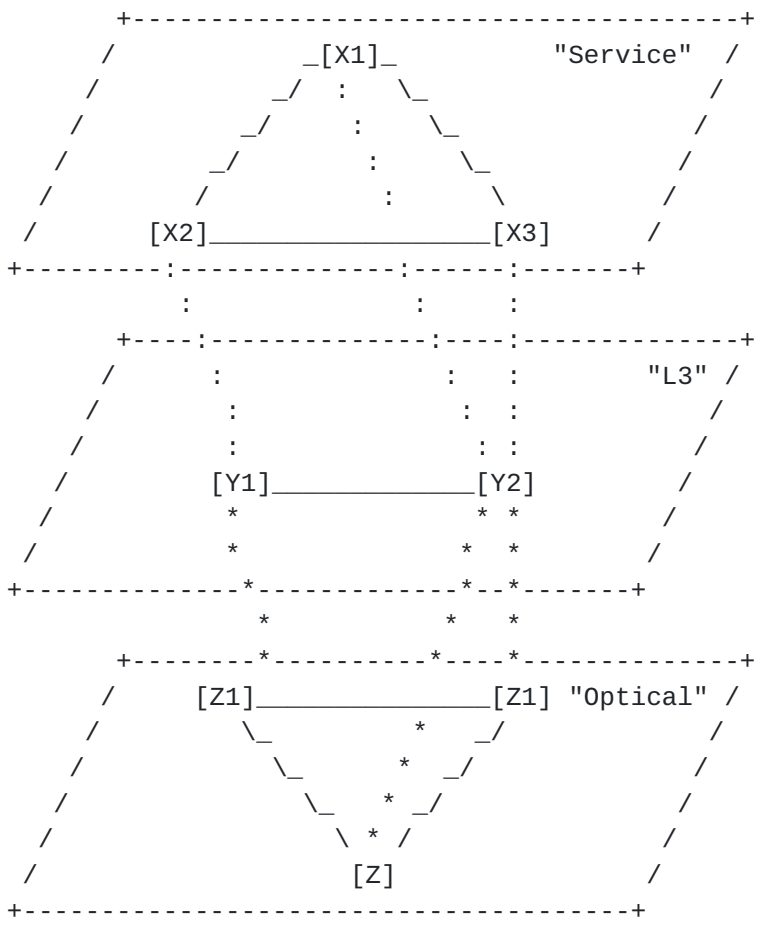

Figure 2: Topology hierarchy (stack) example

The figure shows three topology levels. At 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. The figure shows two Service Functions - X1 and X2 - mapping onto a single L3 network element; this could happen, for example, if two service functions reside in the same VM (or server) and share the same set of network interfaces. The figure shows a single "L3" network element mapped onto multiple "Optical" network elements. This could happen, for example, if a single IP router attaches to multiple ROADMs in the optical domain.

Another example of a service topology stack is shown in the following figure.

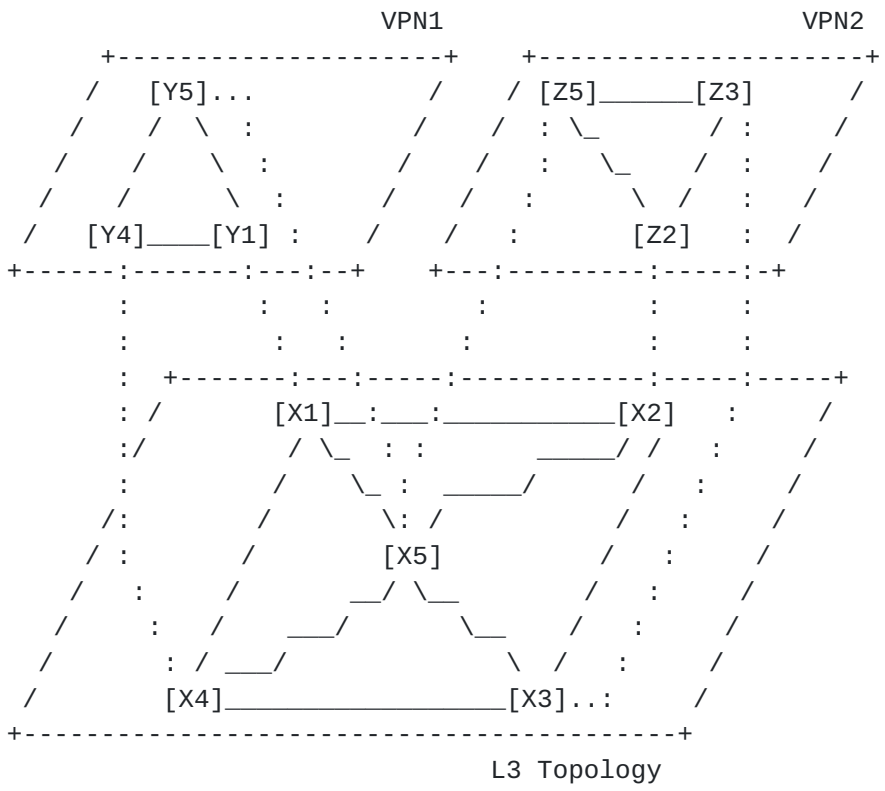

Figure 3: Topology hierarchy (stack) example

The figure 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 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 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 that the data model can be applied to are described in [I-D.draft-ietf-i2rs-usecase-reqs-summary].

In this data model, a network is categorized as either server-provided or not. If a network is server-provided, then it is dynamically learned information that can be read from the operational data-store. For example, as mentioned above, when a network

controller reads a router's topology, that network is server-provided. This data model can also be used to create or modify network topologies such as might be associated with an inventory or with an overlay network. Such a network is not server-provided but configured. This data model allows a network to refer to a supporting-network, supporting-nodes, supporting-links, etc.

The model does allow to layer a network that is configured on top of one that is server-provided. 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 data-store [I-D.draft-ietf-netmod-revised-datastores], defined as requirement Ephemeral-REQ-03 in [I-D.draft-ietf-i2rs-ephemeral-state]. This allows a written (e.g. not server-provided) network topology to refer to a dynamically learned server-provided network. 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 data store, then such a network MAY refer to another network that is server-provided.

An implementation's security policy MAY further restrict what information the server-provided model is allowed to access in standard configuration data-stores, or what server-provided network an ephemeral data store may access. These security policies are outside the scope of the standardization of this model.

Finally, it should be noted that the model is still subject to update per ongoing discussions that are related to design decisions regarding the fact that some layers of the network topology may be server provided while others may be configured. These issues are outlined in section 4.4.10.

2. Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Definitions and Acronyms

Datastore: A conceptual store of instantiated management information, with individual data items represented by data nodes which are arranged in hierarchical manner.

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

HTTP: Hyper-Text Transfer Protocol

IGP: Interior Gateway Protocol

IS-IS: Intermediate System to Intermediate System protocol

NETCONF: Network Configuration Protocol

OSPF: Open Shortest Path First, a link state routing protocol

URI: Uniform Resource Identifier

ReST: Representational State Transfer, a style of stateless interface and protocol that is generally carried over HTTP

YANG: A data definition language for NETCONF

4. Model Structure Details

4.1. Base Network Model

The abstract (base) network model is defined in the network.yang module. Its structure is shown in the following figure. Brackets enclose list keys, "rw" means configuration data, "ro" means operational state data, and "?" designates optional nodes. A "+" indicates a line break.

```
         module: ietf-network
   +--rw networks
      +--rw network* [network-id]
         +--rw network-types
         +--rw network-id            network-id
         +--ro server-provided?      boolean
         +--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 model

The 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 module 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. insert them below "network-types" by ways of 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 empty type. This allows to represent 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 container "l3-unicast-igp-network", which in turn would contain a container "ospf-network".

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 it is contained in. In cases where the same entity takes part in multiple networks, or at multiple layers of a networking stack, the same 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 that it a is part of. To represent that the same entity or same 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, respectively the (entities) nodes in that network, can then be referred to as underlay network and nodes from the other (logical) networks and nodes, respectively. Note that the model allows to define more than one underlay network (and node), allowing for simultaneous representation of layered network- and service topologies and 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 allows also to represent 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.

Finally, there is an object "server-provided". This object is state that indicates how the network came into being. Network data can come into being in one of two ways. In one way, network data is configured by client applications, for example in case of overlay networks that are configured by an SDN Controller application. In annother way, it is populated by the server, in case of networks that can be discovered.

If server-provided is set to false, the network was configured by a client application, for example in the case of an overlay network that is configured by a controller application. If server-provided is set to true, the network was populated by the server itself, respectively an application on the server that is able to discover the network. Client applications SHOULD NOT modify configurations of networks for which "server-provided" is true. When they do, they need to be aware that any modifications they make are subject to be reverted by the server. For servers that support NACM (Netconf Access Control Model), data node rules should ideally prevent write access by other clients to network instances for which server-provided is set to true.

4.2. Base Network Topology Model

The abstract (base) network topology model is defined in the "network-topology.yang" module. It builds on the network model defined in the "network.yang" 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 the following figure. Brackets enclose list keys, "rw" means configuration data, "ro" means operational state data, and "?" designates optional nodes. A "+" indicates a line break.

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

Figure 5: The structure of the abstract (base) network topology 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 in an underlay topology (which are terminated by the corresponding underlay termination points). This is captured in the list "supporting-link".

4.3. Extending the model

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

We start with augmentations against the network.yang module. First, a new network type needs to be defined. For this, a presence container that resembles the new network type is defined. It is inserted by means of augmentation below the network-types container. Subsequently, data nodes for any network-type specific node parameters 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 network-topology.yang. Data nodes are defined both for link parameters, as well as termination point parameters, that are specific to the new network type. Those data nodes are inserted by way of 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 further structure, but lengthens data node path names.

In cases where a hierarchy of network types is defined, augmentations can in turn against augmenting modules, with the module of a network "sub-type" augmenting the module of a network "super-type".

4.4. Discussion and selected design decisions

4.4.1. Container structure

Rather than maintaining lists in separate containers, the 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 specifiers used to refer to specific nodes, be it in management operations or in specifications of constraints, can remain relatively compact. Of course, this means there is no separate structure in instance information that separates elements of different lists from one another. Such structure is semantically not required, although it might enhance human readability in some cases.

4.4.2. Underlay hierarchies and mappings

To minimize assumptions of 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 whether a termination point actually refers to an interface, or whether a node refers to a specific "system" or device; the model at this generic level makes no provisions for that.

Where additional specifics about mappings between upper and lower layers are required, those 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 which introduces a leaf of type ifref that references the corresponding interface [RFC7223]. 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 model makes use of the "require-instance" construct of YANG 1.1 [RFC7950].

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/tps refer to nodes in the underlay are actually in existence. Configuration requests in

which supporting nodes/links/tps 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/tp 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 [I-D.draft-ietf-netconf-yang-push].

4.4.4. Use of groupings

The model makes use of groupings, instead of simply defining data nodes "in-line". This allows to more easily include the corresponding data nodes in notifications, which then do not need to respecify each data node that is to be included. The tradeoff for this 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 XPath-statements need to 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 model includes links that are point-to-point and unidirectional. It does not directly support multipoint and bidirectional links. While this may appear as a limitation, it does keep the model simple, generic, and allows it to very easily be subjected to applications that make use of graph algorithms. Bi-directional connections can be represented through pairs of unidirectional links. Multipoint networks can be represented through pseudo-nodes (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 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, 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 link-to-links mapping known, and the termination points of each link known, termination point mapping information 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 model does provide for the option to configure this information explicitly. The model includes integrity constraints to allow for validating for consistency.

4.4.8. Typing

A network's network types are represented using a container which contains a data node for each of its network types. A network can encompass several types of network simultaneously, hence a container is used instead of a case construct, with each network type in turn represented by a dedicated presence container itself. The reason for not simply using an empty leaf, or even simpler, do away even with the network container and just use 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. Network-type specific containers 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 represent that the same device can be part of multiple networks and topologies. However, the model defines a node as relative to the network that it is contained in. 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 the following figure. It depicts three networks with two nodes each. A physical network P 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 underlay. X1 has both Y1 and D1 as underlay nodes, while Y1 has D1 as underlay node.

Likewise, X2 has both Y2 and D2 as underlay nodes, while Y2 has D2 as underlay node. The fact that X1 and Y1 are both instantiated on the same physical node D1 can be easily derived.

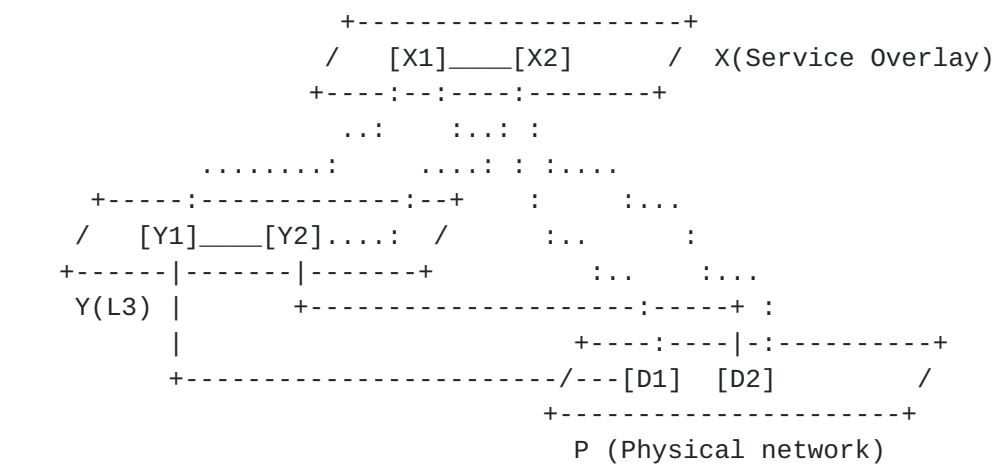

Figure 6: Topology hierarchy example - multiple underlays

In the case of a physical network, nodes represent physical devices and termination points physical ports. It should be noted that it is also conceivable to augment the 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 server-provided network topology

YANG requires data nodes to be designated as either configuration or operational data, but not both, yet it is important to have all network information, including vertical cross-network dependencies, captured in one coherent model. In most cases, network topology information is discovered about a network; the topology is considered a property of the network that is reflected in the model. That said, it is conceivable that certain types of topology need to also be configurable by an application. The model needs to support both cases.

There are several alternatives in which this can be addressed. The alternative chosen in this draft does not restrict network topology information as read-only, but includes a state "server-provided" that indicates for each network whether it is populated by the server or by a client application. Client applications that do attempt to modify network topology may simply see their actions reverted, not unlike other client applications that compete with one another, each

wanting to "own" the same data. When Netconf Access Control Model [RFC6536] is supported, node access rules SHOULD be automatically maintained by a server to deny client write access to network and topology instances for which "server-provided" is true.

It should be noted that this solution stretches its use of the configuration concept slightly. Configuration information in general is subject to backup and restore, which is not applicable to server-provided information. Perhaps more noteworthy is the potential ability of a client to lock a configuration and thus prevent changes to server-provided network topology while the lock is in effect. As a result it would potentially incur a time lag until topology changes that occur in the meantime are reflected, unless implementations choose to provide special treatment for network topology information.

Other alternatives had been considered. In one alternative, all information about network topology is in effect is represented as network state, i.e. as read-only information, regardless of how it came into being. For cases where network topology needs to be configured, a second branch for configurable topology information is introduced. Any network topology configuration is mirrored by network state information. A configurable network will thus be represented twice: once in the read-only list of all networks, a second time in a configuration sandbox. One implication of this solution would have been significantly increased complexity of augmentations due to multiple target branches.

Another alternative would make use of a YANG extension to tag specific network instances as "server-provided" instead of defining a leaf object, or rely on the concept of YANG metadata [RFC7952] for the same effect. The tag would be automatically applied to any topology data that comes into being (respectively is configured) by an embedded application on the network, as opposed to e.g. a controller application.

4.4.11. Identifiers of string or URI type

The current model defines identifiers of nodes, networks, links, and termination points as URIs. An alternative would define them as string.

The case for strings is that they will be easier to implement. The reason for choosing URIs is that the topology/node/tp exists in a larger context, hence it is useful to be able to correlate identifiers across systems. While strings, being the universal data type, are easier for human beings (a string is a string is a string), they also muddle things. What typically happens is that strings have some structure which 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, more complex system.

5. Interactions with Other YANG Modules

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

This is a protocol independent yang 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 e.g. model "ietf-routing" [RFC8022].

The model obeys the requirements for the ephemeral state found in the document [I-D.draft-ietf-i2rs-ephemeral-state]. For ephemeral topology data that is server provided, the process tasked with maintaining topology information will load information from the routing process (such as OSPF) into the data model without relying on a configuration datastore.

6. YANG Modules

6.1. Defining the Abstract Network: network.yang

```
<CODE BEGINS> file "ietf-network@2017-03-01.yang"
module ietf-network {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network";
  prefix nd;

  import ietf-inet-types {
    prefix inet;
  }

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

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

     WG Chair:  Susan Hares
                <mailto:shares@ndzh.com>
```

```
        WG Chair:  Russ White
                   <mailto:russ@riw.us>

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

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

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

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

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

        Editor:    Xufeng Liu
                   <mailto:Xufeng_Liu@jabil.com>";

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

       Copyright (c) 2017 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

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

       This version of this YANG module is part of
       draft-ietf-i2rs-yang-network-topo-12;
       see the RFC itself for full legal notices.

       NOTE TO RFC EDITOR: Please replace above reference to
       draft-ietf-i2rs-yang-network-topo-12 with RFC
       number when published (i.e. RFC xxxx).";

    revision 2017-03-01 {
      description
        "Initial revision.
         NOTE TO RFC EDITOR: Please replace the following reference
```

```
          to draft-ietf-i2rs-yang-network-topo-12 with
          RFC number when published (i.e. RFC xxxx).";
      reference
        "draft-ietf-i2rs-yang-network-topo-12";
    }

    typedef node-id {
      type inet:uri;
      description
        "Identifier for a node.  The precise structure of the node-id
         will be up to the implementation.  Some implementations MAY
         for example, 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 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 an implementation.
        The identifier SHOULD be chosen such that the same network
        will always be identified through the same identifier,
        even if the 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 "/nd:networks/nd:network/nd: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 "/nd:networks/nd:network[nd:network-id=current()/../"+
            "network-ref]/nd:node/nd:node-id";
          require-instance false;
        }
        description
          "Used to reference a node.
           Nodes are identified relative to the network they are
           contained in.";
      }
      uses network-ref;
    }

    container networks {
      description
        "Serves as 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
           network-topology model), as well as 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 network-id;
          description
            "Identifies a network.";
        }
        leaf server-provided {
          type boolean;
          config false;
          description
            "Indicates whether the information concerning this
             particular network is populated by the server
             (server-provided true, the general case for network
             information discovered from the server),
             or whether it is configured by a client
             (server-provided true, possible e.g. for
```

```
            service overlays managed through a controller).
            Clients should not attempt to make modifications
            to network instances with server-provided set to
            true; when they do, they need to be aware that
            any modifications they make are subject to be
            reverted by the server.
            For servers that support NACM (Netconf Access Control
            Model), data node rules should ideally prevent
            write access by other clients to the network instance
            when server-provided is set to true.";
        }
        list supporting-network {
          key "network-ref";
          description
            "An underlay network, used to represent layered network
             topologies.";
          leaf network-ref {
            type leafref {
              path "/networks/network/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
              "Identifies a node uniquely within the containing
               network.";
          }
          list supporting-node {
            key "network-ref node-ref";
            description
              "Represents another node, in an underlay network, that
               this node is supported by.  Used to represent layering
               structure.";
            leaf network-ref {
              type leafref {
                path "../../../supporting-network/network-ref";
              require-instance false;
              }
              description
                "References the underlay network that the
```

```
                 underlay node is part of.";
             }
             leaf node-ref {
               type leafref {
                 path "/networks/network/node/node-id";
               require-instance false;
               }
               description
                 "References the underlay node itself.";
             }
           }
         }
       }
     }
   }

   <CODE ENDS>
```

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

```
 <CODE BEGINS> file "ietf-network-topology@2017-03-01.yang"
 module ietf-network-topology {
   yang-version 1.1;
   namespace "urn:ietf:params:xml:ns:yang:ietf-network-topology";
   prefix lnk;

   import ietf-inet-types {
     prefix inet;
   }
   import ietf-network {
     prefix nd;
   }

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

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

      WG Chair:  Susan Hares
                 <mailto:shares@ndzh.com>

      WG Chair:  Russ White
                 <mailto:russ@riw.us>

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

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

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

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

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

      Editor:     Xufeng Liu
                  <mailto:Xufeng_Liu@jabil.com>";

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

      Copyright (c) 2017 IETF Trust and the persons identified as
      authors of the code.  All rights reserved.

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

      This version of this YANG module is part of
      draft-ietf-i2rs-yang-network-topo-12;
      see the RFC itself for full legal notices.

      NOTE TO RFC EDITOR: Please replace above reference to
      draft-ietf-i2rs-yang-network-topo-12 with RFC
      number when published (i.e. RFC xxxx).";

   revision 2017-03-01 {
     description
       "Initial revision.
        NOTE TO RFC EDITOR: Please replace the following reference
        to draft-ietf-i2rs-yang-network-topo-12 with
        RFC number when published (i.e. RFC xxxx).";
     reference
       "draft-ietf-i2rs-yang-network-topo-12";
   }
```

```
   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 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 that the link is a part
        of.";
   }

   typedef tp-id {
     type inet:uri;
     description
       "An identifier for termination points (TPs) 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 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
       "References a link in a specific network.";
     leaf link-ref {
       type leafref {
         path "/nd:networks/nd:network[nd:network-id=current()/../"+
           "network-ref]/lnk:link/lnk:link-id";
         require-instance false;
       }
       description
         "A type for an absolute reference a link instance.
          (This type should not be used for relative references.
          In such a case, a relative path should be used instead.)";
     }
     uses nd:network-ref;
   }

   grouping tp-ref {
```

```
    description
      "References a termination point in a specific node.";
    leaf tp-ref {
      type leafref {
        path "/nd:networks/nd:network[nd:network-id=current()/../"+
          "network-ref]/nd:node[nd:node-id=current()/../"+
          "node-ref]/lnk:termination-point/lnk: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 nd:node-ref;
  }

  augment "/nd:networks/nd:network" {
    description
      "Add links to the network 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 "../../../nd:node/nd:node-id";
            require-instance false;
          }
          description
            "Source node identifier, must be in same topology.";
        }
        leaf source-tp {
          type leafref {
            path "../../../nd:node[nd:node-id=current()/../"+
```

```
            "source-node]/termination-point/tp-id";
          require-instance false;
        }
        description
          "Termination point within source node that terminates
           the link.";
      }
    }
    container destination {
      description
        "This container holds the logical destination of a
         particular link.";
      leaf dest-node {
        type leafref {
          path "../../../nd:node/nd:node-id";
        require-instance false;
        }
        description
          "Destination node identifier, must be in the same
           network.";
      }
      leaf dest-tp {
        type leafref {
          path "../../../nd:node[nd:node-id=current()/../"+
            "dest-node]/termination-point/tp-id";
          require-instance false;
        }
        description
          "Termination point within destination node that
           terminates the 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.";
    }
    list supporting-link {
      key "network-ref link-ref";
      description
        "Identifies the link, or links, that this link
         is dependent on.";
      leaf network-ref {
        type leafref {
          path "../../../nd:supporting-network/nd:network-ref";
        require-instance false;
        }
```

```
          description
            "This leaf identifies in which underlay topology
             the supporting link is present.";
        }
        leaf link-ref {
          type leafref {
            path "/nd:networks/nd:network[nd:network-id=current()/"+
              "../network-ref]/link/link-id";
            require-instance false;
          }
          description
            "This leaf identifies a link which 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 "/nd:networks/nd:network/nd:node" {
    description
      "Augment termination points which 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 that
           the termination point is dependent on, or maps onto.
           Those termination points will themselves be contained
           in a supporting node.
           This dependency information can be inferred from
           the dependencies between links.  For this reason,
           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 "../../../nd:supporting-node/nd:network-ref";
            require-instance false;
            }
            description
              "This leaf identifies in which topology the
               supporting termination point is present.";
          }
          leaf node-ref {
            type leafref {
              path "../../../nd:supporting-node/nd:node-ref";
            require-instance false;
            }
            description
              "This leaf identifies in which node the supporting
               termination point is present.";
          }
          leaf tp-ref {
            type leafref {
              path "/nd:networks/nd:network[nd:network-id=current()/"+
                "../network-ref]/nd:node[nd:node-id=current()/../"+
                "node-ref]/termination-point/tp-id";
              require-instance false;
            }
            description
              "Reference to 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.

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: nd
Reference: draft-ietf-i2rs-yang-network-topo-12.txt (RFC form)

Name: ietf-network-topology
Namespace: urn:ietf:params:xml:ns:yang:ietf-network-topology
Prefix: lnk
Reference: draft-ietf-i2rs-yang-network-topo-12.txt (RFC form)

8. Security Considerations

The YANG module defined in this memo is independent of a particular protocol and can be accessed via a number of protocols that need to access YANG-defined data. This includes but is not limited to the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242].

The NETCONF access control model (NACM) [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content. However, NACM can be applied analogously also to other protocols that attempt access to YANG-defined data. In fact, it needs to be applied in the same way and should, like YANG, thus be considered independent of any particular protocol that is used to access YANG-defined data. Otherwise, access control rules defined by NACM could be very easily circumvented simply by using another access mechanism which does not enforce NACM. The alternative of mandating the introduction of mechanisms parallel to NACM that specify the same access control rules for other transports is clearly undesirable, as this would not only inhibit ease-of-use of systems that implement multiple protocols to access YANG data, but also open the specter of security holes due to inconsistencies in articulation and enforcement of rules across mechanisms that are essentially redundant.

The YANG module defines 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 do the following:

- Remove or add a node, a link, a termination point, by creating or deleting corresponding elements in the node, link, and termination point lists, respectively. In the case of a topology that is

server-provided, the server will automaticaly correct such misconfiguration attempts. In the case of a topology that is configured, the services provided via this topology might be disrupted. For example, the topology could be "cut" or be configured in a suboptimal way, leading to degradation of service levels and possibly disruption of service.

- Modify the underlay information, i.e. the configuration of supporting-node, supporting-link, and supporting-termination-point, respectively. Again, in the case of a topology that is server-provided, the server will automaticaly correct such misconfiguration attempts. However, in the case of a topology that is configured, this will affect the vertical layering and the way in which the overlay maps onto an overlay. This could be exploited to severely disrupt the overlay network by degrading service levels. In addition, it could be exploited to result in increased consumption of resources in the underlay network, for example by disrupting congruence between overlay and underlay nodes which would result in routing and bandwidth utilization inefficiencies.

For those reasons, it is important that the NETCONF access control model is vigorously applied to prevent topology misconfiguration by unauthorized clients.

Topologies that are server-provided and that provide ephemeral topology information are less vulnerable, as they provide read-only access to clients.

9. Contributors

The model presented in this paper was contributed to by more people than can be listed on the author list. Additional contributors include:

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

10. Acknowledgements

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, and Xian Zhang.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to indicate requirement levels", RFC 2119, March 1997.

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

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

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language", RFC 7950, August 2016.

11.2. Informative References

[I-D.draft-ietf-i2rs-ephemeral-state]
Haas, J. and S. Hares, "I2RS Ephemeral State Requirements", I-D draft-ietf-i2rs-ephemeral-state-23, November 2016.

[I-D.draft-ietf-i2rs-usecase-reqs-summary]
Hares, S. and M. Chen, "Summary of I2RS Use Case Requirements", I-D draft-ietf-i2rs-usecase-reqs-summary-30, November 2016.

[I-D.draft-ietf-netconf-yang-push]
Clemm, A., Voit, E., Gonzalez Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "Subscribing to YANG datastore push updates", I-D draft-ietf-netconf-yang-push-04, October 2016.

[I-D.draft-ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "A Revised Conceptual Model for YANG Datastores", I-D draft-ietf-netmod-revised-datastores-00, December 2016.

[RFC1195] Callon, R., "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments", RFC 1195, December 1990.

[RFC2328] Moy, J., "OSPF Version 2", RFC 2328, April 1998.

[RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, December 2001.

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

[RFC7952] Lhotka, L., "Defining and Using Metadata with YANG", RFC 7952, August 2016.

[RFC8022] Lhotka, L. and A. Lindem, "A YANG Data Model for Routing Management", RFC 8022, November 2016.

Appendix A. Appendix: 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 an NMS) for external utilization. Typically, any network element in a domain can be queried for its topology and expected to return the same result.

In a slightly more complex form, the network element may be a controller, either by nature of it having satellite or subtended devices hanging off of it, or in the more classical sense, such as 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 controllers to establish a topology incorporating of all 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 Transport Controller presents its topology in abstract TE Terms to a Client Packet Controller. This Customized Topology (that 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 exactly cater to its requirements, it may decide to request modifications to the topology. Such customization requests may include addition or deletion of topoogical elements or 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 OSS.

In other words, one system's client (or controller, in this case) may be another system's server (or "uber-network device").

In this scenario, the SDN controller maintains a consolidated 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.

Authors' Addresses

Alexander Clemm
Huawei

EMail: ludwig@clemm.org

Jan Medved
Cisco

EMail: jmedved@cisco.com

Robert Varga
Pantheon Technologies SRO

EMail: robert.varga@pantheon.sk

Nitin Bahadur
Bracket Computing

EMail: nitin_bahadur@yahoo.com

Hariharan Ananthakrishnan
Packet Design

EMail: hari@packetdesign.com

Xufeng Liu
Jabil

EMail: Xufeng_Liu@jabil.com