Network Working Group
Internet-Draft
Intended status: Experimental
Expires: September 10, 2015

A. Clemm
J. Medved
R. Varga
T. Tkacik
Cisco
N. Bahadur
Bracket Computing
H. Ananthakrishnan
Packet Design
March 9, 2015

A Data Model for Network Topologies
draft-clemm-i2rs-yang-network-topo-04.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

Copyright Notice

Table of Contents

1. Introduction

This document introduces an abstract (base) YANG [RFC6020] [RFC6021] 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.

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).

```
                  +------------------------+
                  |                        |
                  | Abstract Network Model |
                  |                        |
                  +------------------------+
                               |
                       +-------+-------+
                       |               |
                       V               V
                +------------+  ..............
                |  Abstract  |  : Inventory  :
                |  Topology  |  :  Model(s)  :
                |   Model    |  :            :
                +------------+  ''''''''''''''
                       |
         +-------------+-------------+-------------+
         |             |             |             |
         V             V             V             V
   ............  ............  ............  ............
   :    L1    :  :    L2    :  :    L3    :  :  Service :
   : Topology :  : Topology :  : Topology :  : Topology :
   :   Model  :  :   Model  :  :   Model  :  :   Model  :
   ''''''''''''  ''''''''''''  ''''''''''''  ''''''''''''
```

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.

```
          +---------------------------------------+
         /            _[X1]_          "Service"  /
        /           _/  :   \_                  /
       /          _/     :    \_               /
      /         _/        :     \_            /
     /         /           :      \          /
    /       [X2]__________________[X3]      /
   +---------:--------------:------:-------+
              :              :     :
          +----:--------------:----:--------------+
         /      :              :   :        "L3" /
        /        :              :  :            /
       /         :               : :           /
      /         [Y1]_____________[Y2]         /
     /           *               * *         /
    /            *              *  *        /
   +--------------*-------------*--*-------+
                   *           *   *
          +--------*----------*----*--------------+
         /     [Z1]_______________[Z1] "Optical" /
        /         \_         *   _/             /
       /            \_      *  _/              /
      /               \_   * _/               /
     /                  \ * /                /
    /                    [Z]                /
   +---------------------------------------+
```

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.

```
                           VPN1                       VPN2
         +---------------------+    +---------------------+
        /   [Y5]...           /    / [Z5]______[Z3]      /
       /    /  \  :          /    /  : \_       / :     /
      /    /    \  :        /    /   :   \_    /  :    /
     /    /      \  :      /    /    :      \  /  :   /
    /   [Y4]____[Y1] :    /    /   :       [Z2]   :  /
   +------:-------:---:--+    +---:---------:-----:-+
          :        :   :          :          :     :
          :         :   :       :            :     :
          :  +-------:---:-----:------------:-----:-----+
          : /       [X1]__:___:___________[X2]   :     /
          :/       / \_  : :       _____/ /   :     /
          :       /    \_ :  _____/      /   :     /
         /:      /       \: /           /   :     /
        / :     /       [X5]          /   :     /
       /   :   /       __/ \__       /   :     /
      /     :   /    ___/       \__    /   :     /
     /       : / ___/              \  /   :     /
    /        [X4]__________________[X3]..:     /
   +------------------------------------------+
                                  L3 Topology
```

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 [topology-use-cases].

2. 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

3. Model Structure Details

3.1. 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.

```
module: network
   +--rw network* [network-id]
      +--rw network-id            network-id
      +--ro server-provided?      boolean
      +--rw network-types
      +--rw supporting-network* [network-ref]
      |  +--rw network-ref    leafref
      +--rw node* [node-id]
         +--rw node-id            node-id
         +--rw supporting-node* [network-ref node-ref]
            +--rw network-ref    leafref
            +--rw node-ref       leafref
```

Figure 4: The structure of the abstract (base) network model

The model contains a list of networks, contained underneath a root container for this module, "network". 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.

3.2. 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.

```
   module: network
      +--rw network* [network-id]
         +--rw network-id            network-id
         +--ro server-provided?      boolean
         +--rw network-types
         +--rw supporting-network* [network-ref]
         |  +--rw network-ref    leafref
         +--rw node* [node-id]
         |  +--rw node-id                  node-id
         |  +--rw supporting-node* [network-ref node-ref]
         |  |  +--rw network-ref    leafref
         |  |  +--rw node-ref       leafref
         |  +--rw lnk:termination-point* [tp-id]
         |     +--rw lnk:tp-id                           tp-id
         |     +--rw lnk:supporting-termination-point*
                               [network-ref node-ref tp-ref]
         |        +--rw lnk:network-ref    leafref
         |        +--rw lnk:node-ref       leafref
         |        +--rw lnk:tp-ref         leafref
         +--rw lnk:link* [link-id]
            +--rw lnk:link-id            link-id
            +--rw lnk:source
            |  +--rw lnk:source-node    leafref
            |  +--rw lnk:source-tp?     leafref
            +--rw lnk:destination
            |  +--rw lnk:dest-node    leafref
            |  +--rw lnk:dest-tp?     leafref
            +--rw lnk:supporting-link* [network-ref link-ref]
               +--rw lnk:network-ref    leafref
               +--rw lnk:link-ref       leafref
```

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".

3.3. 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".

3.4. Discussion and selected design decisions

3.4.1. 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.

3.4.2. 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.

3.4.3. 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.

3.4.4. 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.

3.4.5. 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.

3.4.6. 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.

3.4.7. 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.

3.4.8. 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.

```
                         +---------------------+
                        /   [X1]____[X2]      /  X(Service Overlay)
                       +----:--:----:--------+
                         ..:    :..: :
                ........:     ....: : :....
         +-----:-------------:--+    :     :...
        /   [Y1]____[Y2]....:  /      :..      :
       +------|-------|-------+          :..    :...
        Y(L3) |       +---------------------:-----+ :
              |                         +----:----|-:----------+
              +------------------------/---[D1]  [D2]         /
                                      +----------------------+
                                        P (Physical network)
```

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.

3.5. Items for further discussion

YANG requires data needs 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.

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 flag that indicates for each network whether it should be considered as read-only or configurable by applications.

An alternative would be to designate network list elements as read only. The read-only network list includes each network; it is the complete reference. In parallel a second network list is introduced. This list serves the purpose of being able to configure networks which are then mirrored in the read-only list. The configurable network list adheres to the same structure and uses the same groupings as its read-only counterpart. As most data is defined in those groupings, the amount of additional definitions required will be limited. A configurable network will thus be represented twice: once in the read-only list of all networks, a second time in a configuration sandbox.

Similar considerations apply to scenarios in which data is subject to configuration, but implementations want to be smart enough require only some mapping information, such as which link is supported by which other links, while automatically deriving other mapping information where possible, such as which termination points are supported by which underlay termination points. To accommodate such cases, separate provision may again be made by including another "server-provided" option.

4. YANG Modules

4.1. Defining the Abstract Network: network.yang

```
<CODE BEGINS> file "network.yang"
module network {
  yang-version 1;
  namespace "urn:TBD:params:xml:ns:yang:nodes";
  prefix nd;

  import ietf-inet-types { prefix inet; }

  organization "TBD";
  contact
    "WILL-BE-DEFINED-LATER";
  description
    "This module defines a common base model for a collection
     of nodes in a network. Node definitions s are further used
     in network topologies and inventories.";

  revision 2014-3-9 {
    description
      "Initial revision.";
    reference "draft-clemm-i2rs-yang-network-topo-04";
  }

  typedef node-id {
    type inet:uri;
  }

  typedef network-id {
    type inet:uri;
  }

  grouping network-ref {
    leaf network-ref {
      type leafref {
        path "/network/topology-id";
      }
    }
  }

  grouping node-ref {
    uses network-ref;
    leaf node-ref {
      type leafref {
        path "/network[network-id=current()" +
             "/../network-ref]/node/node-id";
      }
    }
  }

  list network {
    key "network-id";

    leaf network-id {
      type network-id;
    }

    leaf server-provided {
      type boolean;
      config false;
    }

    container network-types {
    }

    list supporting-network {
      key "network-ref";
      leaf network-ref {
        type leafref {
          path "/network/network-id";
        }
      }
    }

    list node {
      key "node-id";
      leaf node-id {
        type node-id;
      }
      list supporting-node {
        key "network-ref node-ref";
        leaf network-ref {
          type leafref {
            path "../../../supporting-network/network-ref";
          }
        }
        leaf node-ref {
          type leafref {
          // path "/network[network-id=current()" +
          // "/../network-ref]/node/node-id";
          path "/network/node/node-id";
          }
        }
      }
    }
  }

}
<CODE ENDS>
```

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

```
  <CODE BEGINS> file "network-topology.yang"
  module network-topology {
    yang-version 1;
    namespace "urn:TBD:params:xml:ns:yang:links";
    prefix lnk;

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

    organization "TBD";
    contact
      "WILL-BE-DEFINED-LATER";
    description
      "This module defines a common base model for a collection of links
       connecting nodes.";

    revision 2014-3-9 {
      description
        "Initial revision.";
      reference "draft-clemm-i2rs-yang-network-topo-04";
    }

    typedef link-id {
      type inet:uri;
      description
        "An identifier for a link in a topology.
         The identifier may be opaque.
         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 on a node.
         The identifier may be opaque.
         The identifier SHOULD be chosen such that the same TP 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 TP
         and/or the type of node and topology that the TP is a part
         of.";
    }

    grouping link-ref {
      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;
      leaf link-ref {
        type leafref {
          path "/nd:network[nd:network-id=current()" +
               "/../nd:network-ref]/link/link-id";
        }
      }
    }

    grouping tp-ref {
      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;
      leaf tp-ref {
        type leafref {
          path "/nd:network[nd:network-id=current()" +
               "/../nd:network-ref]/nd:node[nd:node-id=current()" +
               "/../nd:node-ref]/termination-point/tp-id";
        }
      }
    }

    augment "/nd:network" {
      list link {
        key "link-id";
        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.";
        }
        description
          "A Network Link connects a by Local (Source) node and
          a Remote (Destination) Network Nodes via a set of the
          nodes’ termination points.
          As it is possible to have several links between the same
          source and destination nodes, and as a link could
          potentially be re-homed between termination points, 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.
          Layering dependencies on links in underlay topologies are
          not represented as the layering information of nodes and of
          termination points is sufficient.";
        container source {
          description
            "This container holds the logical source of a particular
            link.";
          leaf source-node {
            type leafref {
              path "../../../nd:node/nd:node-id";
            }
            mandatory true;
            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";
            }
            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";
            }
            mandatory true;
            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";
            }
            description
              "Termination point within destination node that
              terminates the link.";
          }
        }
        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";
            }
            description
              "This leaf identifies in which underlay topology
              supporting link is present.";
          }
          leaf link-ref {
            type leafref {
              path "/nd:network[nd:network-id=" +
                   "current()/../network-ref]/link/link-id";
            }
            description
              "This leaf identifies a link which is forms a part
              of this link’s underlay. Reference loops, where
              a link identifies itself as its underlay, either
              directly or transitively, are not allowed.";
          }
        }
      }
    }

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

  <CODE ENDS>
```

5. Security Considerations

The transport protocol used for sending the topology data MUST support authentication and SHOULD support encryption. The data-model by itself does not create any security implications.

6. Contributors

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

o Ken Gray, Cisco Systems

o Tom Nadeau, Brocade

o Aleksandr Zhdankin, Cisco

7. Acknowledgements

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

8. References

8.1. Normative References

[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", STD 54, 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.

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

[RFC6021] Schoenwaelder, J., "Common YANG Data Types", RFC 6021, October 2010.

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

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

8.2. Informative References

[restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", I-D draft-ietf-netconf-restconf-04, January 2015.

[topology-use-cases]
Medved, J., Previdi, S., Lopez, V., and S. Amante, "Topology API Use Cases", I-D draft-amante-i2rs-topology-use-cases-01, October 2013.

[yang-json]
Lhotka, L., "JSON Encoding of Data Modeled with YANG", I-D draft-ietf-netmod-yang-json-03, February 2015.

Authors’ Addresses

Alexander Clemm
Cisco

EMail: alex@cisco.com

Jan Medved
Cisco

EMail: jmedved@cisco.com

Robert Varga
Cisco

EMail: rovarga@cisco.com

Tony Tkacik
Cisco

EMail: ttkacik@cisco.com

Nitin Bahadur
Bracket Computing

EMail: nitin_bahadur@yahoo.com

Hariharan Ananthakrishnan
Packet Design

EMail: hari@packetdesign.com