Functional Requirements for Transport API

Version Number - 0.~~12~~13
~~Dec 21~~Jan 21, 2015

ONF Document Type: Technical Recommendation
ONF Document Name: Functional Requirements for Transport API

Disclaimer

List of Figures

List of Tables

No table of figures entries found.

1 Introduction

Software defined networking (SDN) paradigm revolves around separation of forwarding/data plane and control plane, logically centralized control and application-focused programmable interfaces. In transport networks where logically-centralized control/management and control-data separation are not new concepts and the network-control function and behavior are well-understood and established, standardizing application programmer's interfaces (APIs) to the network control functions become important.

1.1 Purpose

The purpose of this document is to specify the information that is relevant to an application programmer's interface (API) to transport network-control functions and serve as a "Functional Requirements Specification" (FRS) document for the transport API work in ONF.

Since the APIs are defined at interface boundaries and are intended to mask the need to understand the internal architectures on either side of the interface boundary, the focus has been to define purpose-specific use case scenarios from an application point of view treating the API provider as a "black box". These application use cases have been used to drive the API requirements as well as to harmonize, generalize and normalize the API specifications. To facilitate the understanding of these requirements, some of the use cases are described in the appendices.

Although these requirements are based off few use cases, a key purpose is to ensure that the APIs do not limit valid application use cases not considered here.

1.2 Scope

Based on the use case contributions and early discussions in the ONF/OTWG "Transport API" project discussion group, this work is initially scoped to specifying APIs for the following transport network controller services:

- Topology Service
- Connectivity Service
- Path Computation Service
- Virtual Network Service
- Notification Service

For the purposes of this document, it is assumed that access control and policy details are conveyed via a sideways/orthogonal interface. It is understood that all API requests would be subject to filtering and scoping based on the privileges assigned to the calling entity and these would be based on business contracts as well as security and organizational roles. Application of such policy constraints and filtering to the API requests and responses is out of scope for this document. In other words, the API considerations in this document are from the perspective of the most privileged super-user.

1.3 References

ONF TR-512	Core Information Model
ONF TR-502	SDN Architecture
ONF TR-516	Framework for SDN: Scope and Requirements
ONF2015.276.xx	SDN Notifications Framework (draft)
ONF2015.320.xx	Transport API IM Concepts
ONF2015.381.xx	Transport API Examples
ONF2015.338.xx	State Information Model
ONF2015.323.xx	LTP/End-Point Directionality

1.4 Abbreviations

API	Application Programmer's Interface
IM	Information Model
OTWG	Optical Transport Working Group
OTN	Optical Transport Network
OAM	Operations, Administration and Maintenance
MPLS-TP	MPLS-Transport Profile
EMS/NMS	Element/Network Management System
ASON/GMPLS	Automatic Switched Optical Network/Generalized MultiProtocol Label Switching
SLA	Service Level Agreement
NE	Network Element
FD	Forwarding Domain
NCD	Network Control Domain
EP	End Point
LTP	Logical Termination Point
T-API/TAPI	Transport API
TED	Traffic Engineering Database

1.5 Terms and Definitions

This section defines some key terms that aid in understating the requirements. More information is provided in the appendices and it is recommended that the reader familiarize themselves with the basic concepts, constructs and use cases described in those sections.

In general, the T-API uses terminology that is familiar to the transport network management industry, but maps to constructs defined in the ONF Core Information Model in form of purpose-specific realizations. So it must be noted that these definitions are neither authoritative nor exhaustive, and the reader should refer to the realized/mapped entities defined in ONF Core Information Model document.

Also it should be noted that API IM relates to information exchanged over an interface and the entity definitions are intended to provide a logical structure for encapsulating information that is exchanged, and not intended to specify the information model patterns for implementations on either side of the interface.

API Context (Context)

The T-API defines the scope of control and naming that a particular T-API provider or client application has with respect to the information exchanged over the interface. This *Context* is shared between the API provider and its client and determines the makeup of the network resource abstractions over which the API operates.

Topology The T-API *Topology* is a representation of the transparent topological-aspects of a *Forwarding-Domain (FD).*It describes the underlying topological network of *Nodes* and *Links* that enable the forwarding function provided by the *FD*.

Node

The T-API *Node* is a representation of the opaque forwarding-aspects of a *Forwarding-Domain (FD).*It describes the edge ports of the *FD* (*Node-Edge-Point*) and the forwarding capabilities between those edge ports.

Link

The T-API *Link* is a representation of the effective adjacency between two or more associated *Nodes* in a *Topoplogy*. It is terminated by *Node-Edge-Points* of the associated *Nodes*.

Node-Edge-Point

The T-API *Node-Edge-Point* represents the node-facing aspects of the edge-port functions that access the forwarding capabilities provided by the *Node*. Hence it provides an encapsulation of addressing, mapping, termination, adaptation and OAM functions of one or more transport layers (including circuit and packet forms) performed at the entry and exit points of the *Node*. The *Node-Edge-Points* have a specific *role* and *directionality* with respect to the *Link*

Service-End-Point

The T-API *Service-End-Point* represents the outward-facing aspects of the edge-port functions that access the forwarding capabilities provided by the *Node*. Hence it provides a limited, simplified view of interest to external clients (e.g. client addressing, capacity, resource availability, etc), that enable the clients to make a connectivity service request without the need to understand the provider network internals. *Service-End-Point* have a mapping relationship (*one-to-one, one-to-many, many-to-many*) to *Node-Edge-Points*.[1] The *Service-End-Points* have a specific *role* and directionality with respect to the *Service* request.

Connection-End-Point

The T-API *Connection-End-Point* represents the ingress/egress port aspects that access the forwarding function provided by the *Connection*. The *Connection-End-Points* have a client-server relationship with the Node-Edge-Points. The *Connection-End-Points* have a specific *role* and directionality with respect to the *Connection*.

Connectivity-Service

The T-API *Connectivity-Service* represents an "intent-like" *request* for connectivity between two or more *Service-End-Points*. As such, *Connectivity-Service* is a container for connectivity request details and is distinct from the *Connection* that realizes the request. But it must be noted

[1] Criteria for assigning/mapping ServiceEndPoints to NodeEdgePoints are out of scope of this FRS, but are typically part of implementation agreement (IAs) and some examples are provided by the use cases in the appendices.

these entities are defined for API purposes to refer to different aspects of information exchanged over T-API interface.

Connection

The T-API *Connection* represents an *enabled* (provisioned) potential for forwarding (including all circuit and packet forms) between two or more *Node-Edge-Points* of a *Node*. The T-API *Connection* is terminated by *Connection-End-Points* which are clients of the associated *Node-Edge-Points*. As such, the Connection is a container for provisioned connectivity that tracks the state of the allocated resources and is distinct from the *Connectivity-Service* request. But it must be noted these entities are defined for API purposes to refer to different aspects of information exchanged over T-API interface.

Connection Route (Route)

The T-API *Route* represents the route of a *Connection* through the lower-level *Nodes* in the underlying *Topology*. It is described as a list of references to the lower-level *Connections*.

Path

A *Path* is considered as" ordered list of TE *Links*[2]" (pair of {*Node* IDs and/or *NodeEdgePoint* ID} describing the Link ports). An Connection is composed by a concatenation of link resources (associated with a Link) and XCs in the different nodes.

1.6 Conventions

This document uses the keywords "may" and "must" to qualify optional and mandatory requirements.

[2] The TE Link here refers to a P2P Link or a single pair of Link ports in case of a Multipoint Link

2 Functional Architecture

The of Transport APIs are defined in the background context of network programmability and applies SDN principles to enable cost reduction, innovation and reduced time to market of new services by exposing standard SDN Controller control plane functions API. The Transport API abstracts a common set of control plane functions, such as Network Topology, Connectivity Requests, Path Computation and Network Virtualization to a set of Service interfaces.

These APIs are defined to be applicable on the interface between a Transport SDN controller “Black Box” and its client application. The actors involved in the information exchange over this interface include transport network provider domain controllers in the role of producers and the transport network application systems in the role of the consumers. The transport network application systems could be either a business client system (which itself may include some control functions) or the network operator’s upper level control, orchestration and/or operations systems. This includes privileged application systems that would expect access to internal views of the network model and states using these same set of APIs - for example, usage of topology APIs to access abstract/virtual network topologies provided to business clients as well the underlying actual network topology and entities to which the abstract /virtual entities are mapped. The T-APIs are also intended to be equally applicable between the controllers within a transport controller recursive hierarchy.

It is understood that the APIs are executed within a shared *Context* between the API provider and its client application. The negotiation and setup of the shared *Context* is outside the T-API scope, but is expected to minimally involve agreement on Service End Points, its naming (TRI) and its capabilities. This *Context* determines the makeup of the network resource abstractions over which the API operates. For example, the API client could

- Request retrieval of the *Service End Points* in the shared context
- Request creation of a *Connectivity Service* between the *Service End Points* – these operations can be performed without the knowledge of Network *Topology* or with the knowledge of the Topology (using Topology retrieval APIs)
- Request creation/modification of *Virtual Network* topology within the shared context
- Request retrieval of the (Virtual) Network *Topology* - either provider-assigned (by offline/external means) or client-created- (using VN Service API) within the shared context

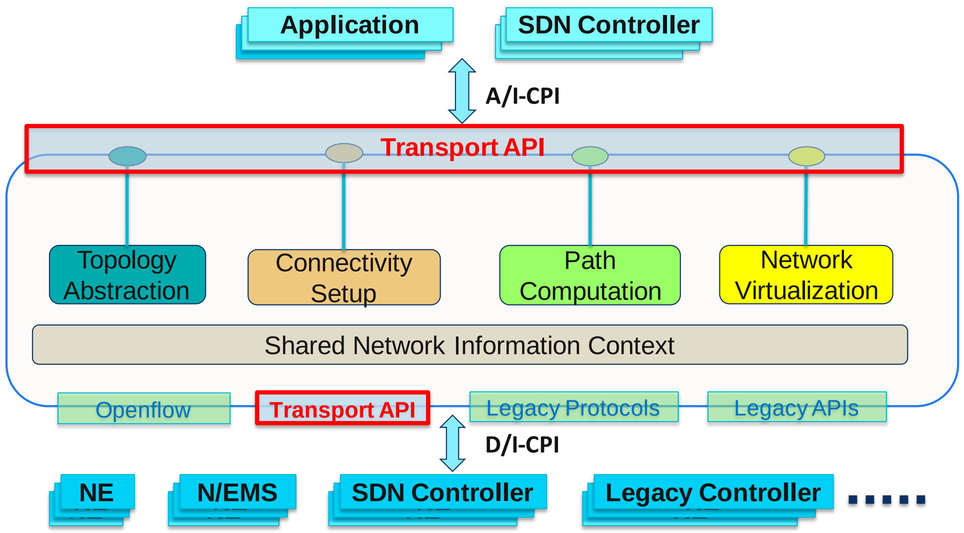


Figure 1: Transport API Functional Architecture

3 Functional Requirements

3.1 Topology Service

3.1.1 Topology Retrieval APIs

TAPI_FR_000x	**Get Topology Details**
Description	• Returns attributes of the *Topology* identified by the provided inputs. • This includes references to lower-level *Nodes* and *Links* encompassed by the *Topology*
Pre-conditions	
Inputs	• *Topology* ID or Name : String - A NULL input value is expected to return the top-most *Topology* that corresponds to the scope of the entire *Context* including any *Off-Network-Links*. • Scope Filter: *Layer-Protocol Name* List : Enumeration value - If set/non-empty, the API call will return references to only those encompassed *Nodes* and *Links* that support at least one of the specified layer protocols
Outputs	• List of IDs, Names, Labels and Extensions (if any) • List of encompassed *Nodes* indexed by Layer including Node details • List of encompassed *Links* indexed by Layer including Link details
Notifications	
Error-conditions	
Post-conditions	
Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get Topology) OIF Topology API Draft Spec oif-2014.91.xx (List Vertices)

TAPI_FR_000x	**Get Node Details**
Description	• Returns attributes of the *Node* identified by the provided inputs. • This includes references to *NodeEdgePoints* aggregated by the *Node* • This also includes attributes representing the identification/naming, states and capabilities of the *Node*.
Pre-conditions	
Inputs	• *Topology* ID or Name : String - ID/name of the containing *Topology* that owns this *Node* - A NULL input value is expected to return the top-most *Topology* that corresponds to the scope of the entire *Context* • *Node* ID or Name : String - When NULL is provided, this API call should return an error condition • Scope Filter: *Layer-Protocol Name* List : Enumeration value - If set/non-empty, the API call will return references to only those aggregated *NodeEdgePoints* that support at least one of the specified layer protocols

Outputs	• List of IDs, Names, Labels and Extensions (if any) • List of supported *Layer-Protocol* Names • Administrative, and Operational, • Transfer characteristics such as Cost, Timing, Integrity and Capacity • List of references to aggregated *NodeEdgePoints* indexed by Layer
Notifications	
Error-conditions	
Post-conditions	
Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get Vertex) OIF Topology API Draft Spec oif-2014.91.xx (Get Vertex)

TAPI_FR_000x	**Get Link Details**
Description	• Returns attributes of the *Link* identified by the provided inputs. • This includes references to *NodeEdgePoints* terminating the *Link*. • This includes references to the *Nodes* associated by the *Link*. • This refers to an abstract/logical entity and could represent virtual links and/or compound links (internally aggregate lower-level serial/parallel links)
Pre-conditions	
Inputs	• *Topology* ID or Name : String - ID/name of the containing *Topology* that owns this *Link* - A NULL input value is expected to return the top-most *Topology* that corresponds to the scope of the entire *Context* including any *Off-Network-Links*. • *Link* ID or Name : String - When NULL is provided, this API call should return an error • Scope Filter: *Layer-Protocol Name* List : Enumeration value - If set/non-empty, the API call will return references to only those terminating *NodeEdgePoints* that support at least one of the specified layer protocols
Outputs	• List of IDs, Names, Labels and Extensions (if any) • Administrative, Operational, Control and Usage States • List of supported *Layer-Protocol* Names • Transfer characteristics such as Cost, Timing, Integrity and Capacity • Risk characteristics including shared-risk • Validation characteristics - Validation describes the various adjacenct discovery and reachability verification protocols. Also may describe Information source and degree of integrity. • List of following details for every *NodeEdgePoint* terminating the Link – Role of the terminating *NodeEdgePoint* in the context of the Link – Direction of the terminating *NodeEdgePoint* in the context of the Link – Reference to terminating *NodeEdgePoint* – List of references to associated *Nodes*
Notifications	
Error-conditions	
Post-conditions	
Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get Edge) OIF Topology API Draft Spec oif-2014.91.xx (Get Edge)

TAPI_FR_000x	**Get Node Edge Point Details**
Description	• Returns attributes of the *NodeEdgePoint* identified by the provided inputs.
Pre-conditions	
Inputs	• *Topology* ID or Name : String - ID/name of the containing *Topology* that owns this *Link* - A NULL input value is expected to return the top-most Topology that corresponds to the scope of the entire Context. • *Node* ID or Name : String - ID/name of the containing *Node* that owns or references this *NodeEdgePoint* - When NULL is provided, this API call should return an error condition • *NodeEdgePoint* ID or Name : String - When NULL is provided, this API call should return an error • Scope Filter: *Layer-Protocol Name* List : Enumeration value - If set/non-empty, the API call will return only the specified *Layer-Protocol* attribute-details indexed by Layer
Outputs	• List of IDs, Names, Labels and Extensions (if any) • Administrative, Operational, Control and Usage States • List of supported *Layer-Protocols* including attribute-details indexed by Layer
Notifications	
Error-conditions	
Post-conditions	
Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get End-point, Edge-End, Edge-End-Resource) OIF Topology API Draft Spec oif-2014.91.xx (Get End-point, Edge-End, Edge-End-Resource)

3.2 Connectivity Service

3.2.1 Connectivity Retrieval APIs

TAPI_FR_000x	**Get Service End Point List**
Description	• Returns list of *ServiceEndPoints* • This includes the *ServiceEndPoints* are being used in a *ConnectivityService* request as well as those that are not being used • This also includes the attribute details for each *ServiceEndPoint* - including references to the mapped *NodeEdgePoint*.
Pre-conditions	
Inputs	• Retrieve Scope Filter: *Layer-Protocol* List : Enumeration value - If set/non-empty, the API call will return references to only those encompassed *ServiceEndPoints* that support at least one of the specified layer protocols

Outputs	List of *ServiceEndPoints* indexed by *Layer* and details for each including: • List of IDs, Names, Labels and Extensions (if any) • Lifecycle State • List of supported *Layer-Protocols* including attribute-details indexed by Layer • Reference to the *NodeEdgePoints* mapped to this *ServiceEndPoint*
Notifications	
Error-conditions	
Post-conditions	
Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get End-point, Edge-End, Edge-End-Resource) OIF Topology API Draft Spec oif-2014.91.xx (Get End-point, Edge-End, Edge-End-Resource)

TAPI_FR_000x	**Get Service End Point Details**
Description	• Returns attributes of the *ServiceEndPoint* identified by the provided inputs. - including references to the mapped NodeEdgePoint.
Pre-conditions	
Inputs	• *ServiceEndPoint* ID or Name : String - When NULL is provided, this API call should return an error condition
Outputs	• List of IDs, Names, Labels and Extensions (if any) • Usage State • List of supported *Layer-Protocols* including attribute-details indexed by Layer • Reference to the *NodeEdgePoint* mapped to this *ServiceEndPoint*
Notifications	
Error-conditions	
Post-conditions	
Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get End-point, Edge-End, Edge-End-Resource) OIF Topology API Draft Spec oif-2014.91.xx (Get End-point, Edge-End, Edge-End-Resource)

TAPI_FR_000x	**Get Connectivity Service List**
Description	• Returns list of *ConnectivityService* entities that represent the connectivity requests that were received • This also includes attribute details for each *ConnectivityService* including – References to *ServiceEndPoints* terminating the *Service* – Optionally References to any *Connections* realizing the *ConnectivityService*
Pre-conditions	
Inputs	• Retrieve Scope Filter: *Layer-Protocol* List : Enumeration value - If set/non-empty, the API call will return references to only those encompassed *ConnectivityServices* that support at least one of the specified layer protocols • Include *Connections* : true or false

Outputs	List of *ConnectivityServices* indexed by *Layer* and details for each including: • List of IDs, Names, Labels and Extensions (if any) • Administrative, Operational, and Lifecycle States • Connectivity Constraints including – Required Constraints such as Capacity – Optional Constraints such as Layer, Latency, Cost, etc • List of following details for every *ServiceEndPoint* associated with the *ConnectivityService* – Role of the terminating *ServiceEndPoint* in the context of the *ConnectivityService* – Directionality of the terminating *ServiceEndPoint* in the context of the *ConnectivityService* – Reference to terminating *ServiceEndPoint* • Optionally List of *Connections* realizing the *ConnectivityService*
Notifications	
Error-conditions	
Post-conditions	
Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get Edge) OIF Topology API Draft Spec oif-2014.91.xx (Get Edge)

TAPI_FR_000x	**Get Connectivity Service Details**
Description	• Returns attributes of the *ConnectivityService* entity identified by the provided inputs. • This includes references to *ServiceEndPoints* terminating the *ConnectivityService*. • This optionally includes references to any *Connections* realizing the *ConnectivityService*.
Pre-conditions	
Inputs	• *Service* ID or Name : String - When NULL is provided, this API call should return an error condition • Include *Connections* : true or false
Outputs	• List of IDs, Names, Labels and Extensions (if any) • Administrative, Operational, and Lifecycle States • Connectivity Constraints including – Required Constraints such as Capacity – Optional Constraints such as Layer, Latency, Cost, etc • List of following details for every *ServiceEndPoint* associated with the *ConnectivityService* – Role of the terminating *ServiceEndPoint* in the context of the *Service* – Directionality of the terminating *ServiceEndPoint* in the context of the *Service* – Reference to terminating *ServiceEndPoint* • Optionally List of *Connections* realizing the *ConnectivityService*
Notifications	
Error-conditions	
Post-conditions	

Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get Edge) OIF Topology API Draft Spec oif-2014.91.xx (Get Edge)

TAPI_FR_000x	**Get Connection Details**
Description	• Returns attributes of the *Connection* entity identified by the provided inputs. • This includes references to *ConnectionEndPoints* terminating the *Connection*. • This includes references to *Paths* in the underlying *Topology*. • This includes reference to the *Node* containing this *Connection*.
Pre-conditions	
Inputs	• *Service* ID or Name : String - ID/name of the containing *ConnectivityService* that requested this *Connection* - When NULL is provided, this API call should return an error condition • *Connection* ID or Name : String - When NULL is provided, this API call should return an error condition
Outputs	• List of IDs, Names, Labels and Extensions (if any) • Operational, and Lifecycle States • Connectivity requirements such as Capacity • Connectivity Constraints such as Layer, Latency, Cost, etc • Validation characteristics • Reference to the parent (containing) *Node* • List of following details for every *ConnectionEndPoint* associated with the *Connection* – Role of the terminating *ConnectionEndPoint* in the context of the *Connection* – Directionality of the terminating *ConnectionEndPoint* in the context of the *Connection* – Reference to terminating *ConnectionEndPoint* • List of *Paths* of the specified *Connection* and details of each including – List of references to lower-level *Connections* that describe the *Path* of the specified *Connection* through the *Nodes* in the underlying *Topology*
Notifications	
Error-conditions	
Post-conditions	
Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get Edge) OIF Topology API Draft Spec oif-2014.91.xx (Get Edge)

TAPI_FR_000x	**Get Connection End Point Details**
Description	• Returns attributes of *ConnectionEndPoint* identified by the provided inputs. • This includes references to the server and client (if any) *NodeEdgePoints* for this *ConnectionEndPoint*. • This includes references to peer (if any) *ConnectionEndPoint* that is connected to this *ConnectionEndPoint*.
Pre-conditions	

Inputs	• *Service* ID or Name : String - ID/name of the containing *ConnectivityService* that requested this *Connection* - When NULL is provided, this API call should return an error condition • *Connection* ID or Name : String - ID/name of the containing *Connection* that owns or references this *ConnectionEndPoint* - When NULL is provided, this API call should return an error condition • *ConnectionEndPoint* ID or Name : String - When NULL is provided, this API call should return an error condition
Outputs	• List of IDs, Names, Labels and Extensions (if any) • Operational State • List of supported *Layer-Protocols* including attribute-details indexed by Layer • Reference to the Server (containing) and Client (if any) *NodeEdgePoint* • Reference to the Peer (if any) *ConnectionEndPoint*
Notifications	
Error-conditions	
Post-conditions	
Sources	TMF MTNM/MTOSI 3.5 OIF-ONF Demo Topology API Spec oif-p105.11.xx (Get End-point, Edge-End, Edge-End-Resource) OIF Topology API Draft Spec oif-2014.91.xx (Get End-point, Edge-End, Edge-End-Resource)

3.2.2 Connectivity Request APIs

TAPI_FR_0001	**Create Connectivity Service**
Description	• Causes creation of a *ForwardingConstruct* representing the *ConnectivityService* request to connect the *ServiceEndPoints* within the shared *Context* between API Client and Provider • Returns Service ID to be used as reference for future actions • Initial definition will be for a basic point-to-point bidirectional service
Pre-conditions	• Requestor/Client has visibility of the set of *Service-End-Points* between which connectivity is desired within the *Context* • Requestor/Client has information about the types of connectivity available and constraints it can specify such as Service Level • Requestor/Client may be aware of other existing *ConnectivityServices* and their IDs
Inputs	• List of following details for every *ServiceEndPoint* for the *ConnectivityService* – Role of the terminating *ServiceEndPoint* in the context of the *Service* – Directionality of the terminating *ServiceEndPoint* in the context of the *Service* – Reference (Name/ID) to terminating *ServiceEndPoint* – Optionally the Layer of the *ServiceEndPoint* if it supports multiple layers • Connectivity Constraints including – Required Constraints such as Capacity – Optional Constraints such as Layer, Latency, Cost, etc • Start Time & End Time

Outputs	• Service ID • Operational State • Lifecycle State • Confirmation of Service Characteristics : See above inputs
Notifications	Success/Failure Change of Operational State
Error-conditions	Service not supported Service input not supported Endpoint not recognized Inconsistent input Layer constraints
Post-conditions	
Sources	Oif – cite specific documents Onf IETF

TAPI_FR_0002	**Update Connectivity Service**
Description	• Causes modification of an existing *Forwarding-Construct* representing the *ConnectivityService* request identified by the inputs • Returns confirmation or rejection of modification
Pre-conditions	• Requestor/Client already knows the existing *Service* ID • Requestor/Client has information about the types of Service Characteristics that can be modified
Inputs	• Service ID or Name - When NULL is provided, this API call should return an error condition • Connectivity Constraints including – Required Constraints such as Capacity – Optional Constraints such as Layer, Latency, Cost, etc • Start Time & End Time
Outputs	• Success/Failure • Operational State • Lifecycle State • Confirmation of Service Characteristics : See inputs above
Notifications	Success/Failure
Error-conditions	Modification could not be supported Modification parameter not understood Modification not allowed
Post-conditions	Service modified
Sources	

TAPI_FR_0003	**Delete Connectivity Service**
Description	• Causes deletion of an existing *ConnectivityService* • Returns confirmation of deletion
Pre-conditions	• Requestor/Client already knows the existing *Service* ID
Inputs	• *Service* ID or Name: String - When NULL is provided, this API call should return an error condition
Outputs	• Success/Failure
Notifications	Change of operational mode

Error-conditions	
Post-conditions	Service deleted
Sources	

3.3 Path Computation Service

The APIs in this section have been defined making certain assumptions on the division of responsibilities and sequence flow of interactions between different T-API Service interfaces. For example, it is assumed that a connection control module that handles the *ConnectivityService* request, is in charge of the management and implementation of the *Connections* in terms of real resource commitment for the routes (*Paths*) of an *Connection*. In contrast, a routing control module that handles the *PathComputationService* requests (from the internal connection-control module or external applications) is responsible for computing and providing the *Paths* for a potential *Connection* as output.

3.3.1 Path Computation Request APIs

TAPI_FR_xxxx	**Compute P2P Path**
Description	Path computation can be called in the context of service request since path computation is provided in a domain according to the policy which has to refer to specification of service for which the path computation request is required. The client side of the API can request the server side of the API to compute a single path or a batch of paths with consideration of a set of constraints
Pre-conditions	The server side of this API should have the topology information (including TE information) of the network in which the path computation applies. Includes Connectivity matrix, port label restriction (only applicable to optical layer path computation)
Inputs	• List of following details for every[3] *ServiceEndPoint* for the *ConnectivityService* – Role[4] of the terminating *ServiceEndPoint* in the context of the *Service* – Directionality of the terminating *ServiceEndPoint* in the context of the *Service* – Reference (Name/ID) to terminating *ServiceEndPoint* – Optionally the Layer of the *ServiceEndPoint* if it supports multiple layers • Connectivity Constraints including – Required Constraints such as Capacity – Optional Constraints such as Layer, Latency, Cost, etc • Objective function • If concurrent Paths should be computed
Outputs	List of paths computed containing following information (only “one” if shouldComputeConcurrentPaths is false) • Path identifier (identifier of the calculated route) • Routing constrains (Description of routing constraints that are met) • Path (ordered list of strict hops (TE Links) or loose hops/(Nodes)
Notifications	

[3] The number of ServiceEndPOints is constraint to only 2 for the Path Computation request
[4] The value for Role is constrained to only Symmetric for the Path Computation request

Error-conditions	Cause of failure
Post-conditions	
Sources	Oif Onf IETF

TAPI_FR_xxxx	**Optimize P2P Path**
Description	A connection can be reconfigured to meet new constraints and achieve path optimization via this API. Reconfiguration may involve intermediate-point changes and route changes
Pre-conditions	The server side of this API should have the topology information (including TE information) of the network in which the path computation applies
Inputs	• Path Id: Identifier of route to be modified • Connection Id: Identifies resources in use for the Connection for the path being optimized • Connectivity Constraints including – Required Constraints such as Capacity – Optional Constraints such as Layer, Latency, Cost, etc • Objective Function • Optimization Constraint • If concurrent Paths should be computed
Outputs	List of paths computed containing following information (only "one" if shouldComputeConcurrentPaths = false) • Path identifier (identifier of the calculated route) • Routing constrains (Description of routing constraints that are met) • Path (ordered list of strict hops (TE Links) or loose hops/(Nodes)
Notifications	
Error-conditions	Cause of failure: • Optimization fail due to insufficient resources • Cannot readjust resource allocation without interruption of existing services. • Cannot satisfy other constraints, such as timing issue or performance threshold.
Post-conditions	
Sources	Oif Onf IETF

3.4 Virtual Network Service

Comment [KS1]: Experimental: This section needs further work to align with the rest of T-API

3.4.1 Virtual Network Request APIs

TAPI_FR_00xx	**Create Virtual Network**
Description	For the client side of the API to request creation of a virtual network from a network (maybe physical or virtual network, recursively) provided by the server side of this API, according to the traffic volume between the access points of the client. As a result, the server side of this API will reserve a set of resources to build up the virtual network, over which the client side of the API is allowed to e.g. configure virtual connections (through other transport APIs).

Pre-conditions	The server side of this API should have the topology information of the network under its control.
Inputs	• Access points: The access points of the client to be connected to the virtual network. • Traffic matrix: A matrix to describe the traffic (e.g. bandwidth) between each pair of the client's access points. • VN SLA: The service level agreement of a virtual network, e.g. the capability of recovery from virtual network topology level failures.
Outputs	• Result of the request: Succeed or fail to create the requested virtual network. • VN ID: The identifier of the virtual network. • Description of VN topology: A detailed description of the created virtual topology, which may include: – Virtual node information: the information of the virtual nodes in the created virtual network, e.g., virtual node ID. – Virtual link information: the information of the virtual links in the created virtual network, e.g., virtual link ID, virtual port ID and bandwidth information.
Notifications	• A notification should be sent to the client when virtual network failure happens (e.g., some virtual link or virtual node failures happen). • A notification should be sent to the client when the virtual network has recovered.
Error-conditions	• There are not enough resources to set up the virtual network that meets the client traffic requirement.
Post-conditions	• The server side of this API reserves a set of resources to build up the virtual network. • The server side of this API maintains the resources and the status of the created virtual networks, as well as the mapping relationship between the created virtual networks and the network under control of the server side. • The client side of this API is allowed to have virtual connection control over the virtual network.
Sources	TBD

TAPI_FR_00xx	**Update Virtual Network**
Description	TThe client side can modify the virtual network which has already beencreated. Such modification may include adding and/or deleting virtual nodes and/or virtual links, modifying link capacity of virtual links, SLA of the virtual network, etc.. As a result, the server side of this API will modify the reservation of resources under its control to form the modified virtual network, according to the new traffic matrix.
Pre-conditions	• The server side of this API has the topology information of the network under its control. • The client side of this API has already requested a virtual network from the server side of this API successfully.

Inputs	• VN ID: The identifier of the virtual network to be modified. • Modified access points: The access points of the client to be connected to the modified virtual network. • Modified traffic matrix: A matrix to describe the modified traffic (e.g. bandwidth) between each pair of client's access points. • Modified VN SLA: The service level agreement of the modified virtual network, e.g. the capability of recovery from virtual network topology level failures.
Outputs	• Result of the modification request: Succeed or fail to modify the virtual network. • VN description: A detailed description of the modified virtual network, which may include: – Virtual node information: the information of the virtual nodes in the created virtual network, e.g., virtual node ID. – Virtual link information: the information of the virtual links in the created virtual network, e.g., virtual link ID, virtual port ID and bandwidth information.
Notifications	• A notification should be sent to the client when virtual network failure happens (e.g., some virtual link or virtual node failures happen). • A notification should be sent to the client when virtual network has recovered.
Error-conditions	• The requested VN (to be modified) ID does not exist at the server side. • There are not enough resources to support modification of the virtual network.
Post-conditions	• The server side of this API modifies the reservation of resource under its control to form the modified virtual network. • The server side of this API maintains the resource and the status of the modified virtual network, as well as the mapping relationship between the modified virtual network and the network under control of the server side. • The client side of this API is allowed to have virtual connection control over the modified virtual network.
Sources	TBD

TAPI_FR_00xx	**Delete Virtual Network**
Description	For the client side of the API to delete the virtual network that it owns. As a result, the server side of this API will release the resources used by this virtual network.
Pre-conditions	• The server side of this API has the topology information of the network under its control. • The client side of this API has requested a virtual network from the server side of this API. • All virtual connections in the virtual network should be deleted by the client before deleting the virtual network.
Inputs	• VN ID: The identifier of the virtual network to be deleted
Outputs	• Result of the request: Succeed or fail to delete the requested virtual network.
Notifications	
Error-conditions	• The requested VN (to be deleted) ID does not exist at the server side. • One or more virtual connections remain in the virtual network.
Post-conditions	• The server side of this API releases the resources used by the virtual network.
Sources	TBD

3.5 Notification Service

Notifications refer to the set of autonomous messages that provide information about events, for example, alarms, performance monitoring (PM) threshold crossings, object creation/deletion, attribute value change (AVC), state change, etc. In some standards, notifications are referred to as event reports. The specification of functional requirements for Alarms (FM) and TCAs (PM) notifications will be provided in the next release of this document.

Notifications specifications are generally written around a model of a manager and an agent. The term manager is used to designate an entity that governs notification subscriptions and receives notification messages, while the term agent is used to designate an entity that recognizes events, turns them into notification messages, and transmits them to pertinent subscribers. Thus, the agent represents (acts on behalf of) managed objects that are subject to events, and emits notification messages to inform zero or more managers (receiving entities) of these events. The manager-agent terminology is being retained for compatibility with existing standards; however, there is no intention to imply a distinction between management and control.

Notifications may be separated into two classes, primitive notifications from managed object instances to agents, and those that are processed and emitted by the notifications agents into a form suitable for exposure to some managed object instance (subscriber). A notification agent (NA) is modeled as the publisher of notification messages, to any number of subscription target destinations. Examples of further processing include interpretation, correlation, filtering, embellishment with time stamp, sequence number, system and managed object identifier. Primitive notifications are out of scope of this document.

3.5.1 Notification Subscription and Filtering APIs

Notifications shall follow a publish and subscribe model. A notifications manager shall be able to create, query, modify, suspend, resume and delete a notifications subscription. The notifications available to a manager for subscription are bounded by the (virtual) resources and privileges visible to that manager. Subscriptions shall not time out and be automatically deleted during the lifetime of a given session.

Knowledge of available notifications and their sources is a precondition for subscription. A notifications agent shall permit a notifications manager to discover its supported notification types. Particularly in the case of virtualized resources of SDN, notifications discovery may be a feature of a general resource discovery mechanism.

Notification subscribers specify their interest according to filter, where a filter is any combination of (event related) criteria that can be unambiguously evaluated against an input to produce an accept/reject result. A filter is an attribute of a subscription, and may be modified over the lifetime of the subscription. Filters do not exist as separate managed object instances, are local to one subscription, and do not survive the deletion of that subscription. The actual notifications delivered to a target are those that pass the subscription filter.

TAPI_FR_xxxx	**Discover Supported Notification Types**
Description	

Pre-conditions	Knowledge of the Notification Server (e.g. URI, IP/Port, etc)
Inputs	
Outputs	• *Notification-Type* List : Enumeration value The notification types are specified in section 3.5.2.
Notifications	
Error-conditions	Reason for Failure
Post-conditions	Supported notification types discovered
Sources	

TAPI_FR_xxxx	**Create Notification Subscription**
Description	
Pre-conditions	Knowledge of available notifications types and their sources
Inputs	• Subscription Scope Filter: *Notification-Type* List : Enumeration value If set/non-empty, the system will push *Notifications* of one of the specified notification types only. The notification types are specified in section 3.5.2. • Subscription Scope Filter: *Object-Type* List : Enumeration value If set/non-empty, the system will push *Notifications* related to one of the specified object types only. The object types are specified in section 3.5.2. • Subscription Scope Filter: *Layer-Protocol* List : Enumeration value If set/non-empty, the system will push *Notifications* related to one of the specified layer protocols only. The layer protocols are specified in section 3.5.2.
Outputs	subscriptionId
Notifications	notificationId Object Creation Notification per the successful subscription.
Error-conditions	Reason for Failure
Post-conditions	Subscription created
Sources	

TAPI_FR_xxxx	**Modify Notification Subscription**
Description	
Pre-conditions	Knowledge of available notifications subscriptions, types and their sources
Inputs	• Subscription Id: String • Subscription Scope Filter: *Notification-Type* List : Enumeration value If set/non-empty, the system will push *Notifications* of one of the specified notification types only. The notification types are specified in section 3.5.2. • Subscription Scope Filter: *Object-Type* List : Enumeration value If set/non-empty, the system will push *Notifications* related to one of the specified object types only. The object types are specified in section 3.5.2. • Subscription Scope Filter: *Layer-Protocol* List : Enumeration value If set/non-empty, the system will push *Notifications* related to one of the specified layer protocols only. The layer protocols are specified in section 3.5.2.
Outputs	subscriptionId
Notifications	notificationId Object Creation Notification per the successful subscription.
Error-conditions	Reason for Failure
Post-conditions	Subscription created
Sources	

TAPI_FR_xxxx	Delete Notification Subscription
Description	
Pre-conditions	Knowledge of available notification subscriptions
Inputs	• Subscription Id: String.
Outputs	SubscriptionId of the notification subscription that was deleted
Notifications	Object Deletion Notification per the successful subscription.
Error-conditions	Reason for Failure
Post-conditions	Subscription deleted
Sources	

TAPI_FR_xxxx	Suspend Notification Subscription
Description	
Pre-conditions	Knowledge of available notification subscriptions
Inputs	• Subscription Id: String.
Outputs	SubscriptionId of the notification subscription that was suspended
Notifications	State Change Notification per the successful subscription suspension.
Error-conditions	Reason for Failure
Post-conditions	Subscription suspended
Sources	

TAPI_FR_xxxx	Resume Notification Subscription
Description	
Pre-conditions	Knowledge of available notification subscriptions
Inputs	• Subscription Id: String.
Outputs	SubscriptionId of the notification subscription that was resumed
Notifications	State Change Notification per the successful subscription that was resumed
Error-conditions	Reason for Failure
Post-conditions	Subscription resumed
Sources	

3.5.2 Notification Message Types

TAPI_FR_000x	Object Creation Notification
Description	
Values	This message shall minimally support the following attributes • Notification Header (as per above) • Source Indicator • Attribute List – each consisting of ○ {attributeName, attributeValue} • Additional Information • Additional Text
Sources	ITU-T M.3702

TAPI_FR_000x	**Object Deletion Notification**
Description	
Values	This message shall minimally support the following attributes • Notification Header (as per above) • Source Indicator • Attribute List - each consisting of ○ {attributeName, attributeValue} • Additional Information • Additional Text
Sources	ITU-T M.3702

TAPI_FR_000x	**Attribute Value Change Notification**
Description	
Values	This message shall minimally support the following attributes • Notification Header (as per above) • Source Indicator • Attribute Value Change List each consisting of ○ { attributeName, oldAttributeValue, newAttributeValue } • Additional Information • Additional Text
Sources	ITU-T M.3702

TAPI_FR_000x	**State Change Notification**
Description	
Values	This message shall minimally support the following attributes • Notification Header (as per above) • Source Indicator {resourceOperation, managementOperation, unknown} • State Change List each consisting of ○ { attributeName, oldStateValue, newStateValue } • Additional Information • Additional Text
Sources	ITU-T M.3702

3.6 TAPI Data Types

This section identifies the data types, formats and values commonly associated with the parameters of the various TAPI service interface APIs.

TAPI_FR_000x	**Layer Protocol Name**
Values	The Layer-Protocol attribute shall minimally support following for Connectivity layers • OCH • ODU • ETH • MPLS-TP

TAPI_FR_000x	**Capacity (Fixed Bandwidth)**
Values	The Capacity (Bandwidth) attribute is applicable for digital layers and shall minimally support following values in Mbps • 10 (Ethernet Lan) • 100 (Fast Ethernet) • 1000 (Gigabit Ethernet) • 2400 (ODU1/OTU1) • 10000 (10GBE/ODU2/OTU2) • 40000 (40GBE/ODU3/OTU3) • 100000 (100GBE/ODU4/OTU4)

TAPI_FR_000x	**Administration State**
Values	• LOCKED • UNLOCKED

TAPI_FR_000x	**Operational State**
Values	• ENABLED • DISABLED

TAPI_FR_000x	**Lifecycle State**
Values	• PLANNED • POTENTIAL • INSTALLED • IN_CONFLICT • PENDING_REMOVAL

TAPI_FR_000x	**Port Role**
Values	Denotes the role of the *End-Point* with respect to the *Forwarding-Construct* • Symmetric • Root • Leaf

TAPI_FR_000x	**Port Direction**
Values	Denotes the directionality of the signal-flow in the *Port* with respect to the *Forwarding-Construct* • Input • Output • Bidirectional

TAPI_FR_000x	**Termination Direction**

Values	Denotes the directionality of the signal-flow in the Service*EndPoint* or *ConnectionEndPoint* • Sink • Source • Bidirectional

TAPI_FR_000x	**Service End Point TRI format**
Values	The End-Point Name shall minimally support following formats • TRI • URI • Domain-specific String The formats for the TRI is out of scope for this FRS and is typically part of the implementation agreements (IAs)

TAPI_FR_000x	**Service Type**
Values	The Service Type attribute shall minimally support following values • POINT_TO_POINT • POINT_TO_MULTIPOINT • MULTIPOINT

TAPI_FR_000x	**Connectivity Constraints**
Values	The following are the required Connectivity parameters • Service Type: The type of connectivity requested • Capacity: Requested bandwidth (fixed or range) The following are the optional Connectivity constraint parameters • Service Layer: Represents the layer of transported service • Service Level – a abstract value the meaning of which is mutually agreed – typically represents metrics such as - Class of service, priority, resiliency, availability • Latency – integer value and unit - upper bound • Cost – Vector of one or more metrics that would enable the provider to make a decision when implementing the Service • SRLG /Diversity – an exclude Service ID – indicates that the requested service should be diverse (not share resources) from specified service • Include *Path* – indicates partial or complete set of nodes and/or NodeEdgePoints to be used (TE Links) • Exclude *Path* - indicates partial set of nodes and/or NodeEdgePoints to be avoided

TAPI_FR_000x	**Path Optimization Constraint**
Values	The following are the optimization constraint parameters to be provided as input to the Path Optimization API • Flag resource sharing (max re-usage/min re-usage) Whether new resource can be used or no • Minimum/maximum link utilization value • Whether traffic interruption allowed or not

TAPI_FR_000x	**Path Objective Function**
Values	The following are the Objective Function parameters to be provided as input to the Path Computation Service API • Minimize the cost • Maximize the amount of successfully computed paths (Only for concurrent path computation) • Minimize aggregate Bandwidth Consumption (Only for concurrent path computation) • Minimize the load of the Most Loaded Link (Only for concurrent path computation) • Minimize Cumulative Cost of a set of paths (Only for concurrent path computation)

TAPI_FR_000x	**Notification-Header**
Description	All notifications shall contain a common header that identifies the type of the notification, the producer of the notification (object class, object instance), the time at which the underlying event occurred, a unique notification identifier, and the object instance identifier of the system hosting the agent.
Values	This header shall minimally support the following attributes • Notification Identifier (uuid) • Notification Type • Object Type • Object Instance Identifier (uuid) • Object Instance Name List • Event Time Stamp • Correlated Notification Information (optional) ○ Notification Identifier ○ Detector State Model ○ Correlation Text

TAPI_FR_000x	**Notification-Type**
Description	Notification agents shall classify notification messages into notification types; additional notification types may be defined as needed.
Values	This enumeration shall minimally support following values • Object Creation • Object Deletion • Attribute Value Change • State Change

TAPI_FR_000x	**Object-Type**
Description	Notification agents shall identify the type of the object on which a notification is raised and allow filtering of the notifications based on object types; additional object types may be defined as needed.

Values	This enumeration shall minimally support following values • Topology • Node • Link • Node-Edge-Point • Service-End-Point • Connection-End-Point • Connectivity-Service • Connection

TAPI_FR_000x	**Source-Indicator**
Description	Notification agents shall identify the source of notification messages.
Values	This enumeration attribute shall minimally support following values • ResourceOperation • ManagementOperation • Unknown
Sources	M.xxxx

4 Appendix A: Transport API Concepts Overview

4.1 Node and Topology Aspects of Forwarding Domain

The *Forwarding-Domain* represents the opportunity to enable forwarding between points represented by the *LTPs* available at the boundary of the *Forwarding-Domain*. The *Forwarding-Domain* can hold zero or more instances of *Connections* and provides the context for requesting and instructing the formation, adjustment and removal of *Connections*.

The *Forwarding-Domain* supports a recursive aggregation relationship such that the internal construction of a *Forwarding-Domain* can be exposed as multiple lower level *Forwarding-Domains* and associated *Links* (partitioning). A *Forwarding-Domain* within a Network-Element may represent a switch matrix/fabric as well as those cases where the matrix is itself decomposed so the *Forwarding-Domain* may be smaller than the scope of the matrix. On the other-end of the aggregation relationship spectrum, it is expected that there will always be an all-encompassing top-most Forwarding-Domain that corresponds to the scope of the entire *Network-Control-Domain* (*Context*)

For the purposes of API requirements, the two aspects of the *Forwarding-Domain* have been refactored as separate entities:

- Node – which represents the forwarding-potential between its edge-points (*LTP*s)
- Topology – which represents the topological-aggregation of lower-level *Links* and *Nodes*

Depending on the frame of reference for an API invocation (or the position of an imaginary observer), only the opaque *Node*-aspects of a *Forwarding-Domain* would be visible (placing the observer external to the *Forwarding-Domain*) or the entire *Topology*-structure of a *Forwarding-Domain* would be visible (placing the observer internal to the Forwarding-Domain)

An instance of *Forwarding-Domain* is associated with zero or more *LTP* instances.

The effective adjacency between two or more *Forwarding-Domains* is modeled by a *Link*. In its basic form (i.e., point-to-point Link) it associates a set of *LTP* client layers on one *Forwarding-Domain* with an equivalent set of *LTP* client layers on another *Forwarding-Domain*. A *Link* may offer parameters such as capacity and delay depending on the type of technology that supports the *Link*.

A *Forwarding-Domain* may aggregate *Links* that are wholly within the bounds of the *Forwarding-Domain*. A *Link* with an Off-network end cannot be encompassed by a *Forwarding-Domain*.

The association of the *Link* to *LTPs* is made via the *Link-Ends* which are essentially the ports of the *Link*. In T-API IM, the *Link-Ends* are refactored into the *Link* itself so that the *Link* maintains the *Link-End* attributes (such as link-end *Role*) and has direct association to its terminating *LTPs* (*NodeEdgePoints*).

The *Link* can support multiple transport layers via its associated *LTP* instance.

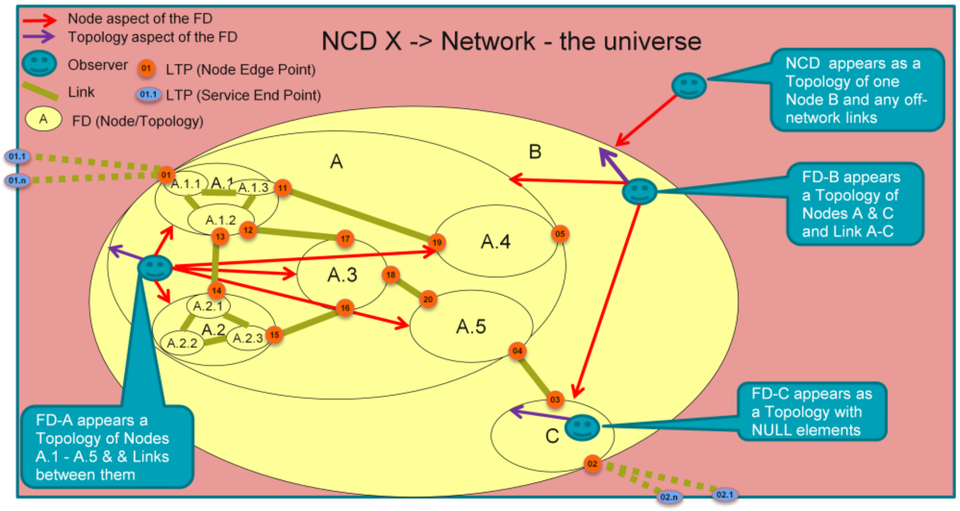

Node aspect of the FD
Topology aspect of the FD
Observer
Link
FD (Node/Topology)
LTP (Node Edge Point)
LTP (Service End Point)
NCD X -> Network - the universe
NCD appears as a Topology of one Node B and any off-network links
FD-B appears a Topology of Nodes A & C and Link A-C
FD-A appears a Topology of Nodes A.1 - A.5 & & Links between them
FD-C appears as a Topology with NULL elements
A
B
C
A.1
A.1.1
A.1.2
A.1.3
A.2
A.2.1
A.2.2
A.2.3
A.3
A.4
A.5
01.1
01.n
02.1
02.n

Figure 2: Node/Topology perspectives of recursively partitioned Forwarding Domain (FD)

4.2 Network Control Domain and Contexts

The T-API *Context* is a realization of the *Network-Control-Domain* as defined in the ONF Core Information Model and defines the scope of control and naming that a particular SDN controller or application module has with respect to a specific instance of network abstraction.

In interfaces where an abstracted view of network is offered, e.g. in client/server SDN controller relationship, the *Context* entity defines the scope of control of the client SDN controller on the abstracted/virtual network view that has been provided by the server SDN controller. Thus *Context* relates to an abstraction of the partitioned provider resources allocated to that particular client. In such cases, the *Context* also provides the scope of naming space for identifying objects representing the (virtual) resources within the (virtual) network view.

Often, a provider controller may also create and maintain multiple abstractions of its controlled network for its own purposes and not necessarily to service any particular business clients.

The following figures illustrate few examples of NCD *Contexts* in a hierarchical SDN controller system:

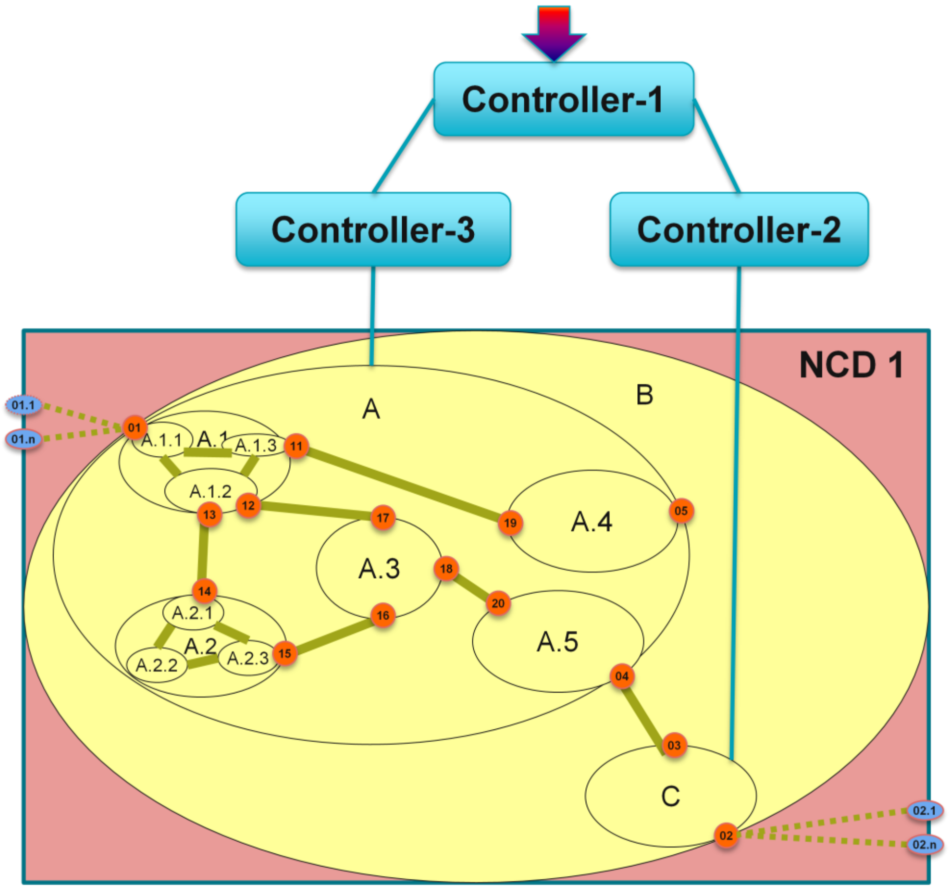


Figure 3: View of Controller-1 NCD based on Views exported by Controllers 2 & 3

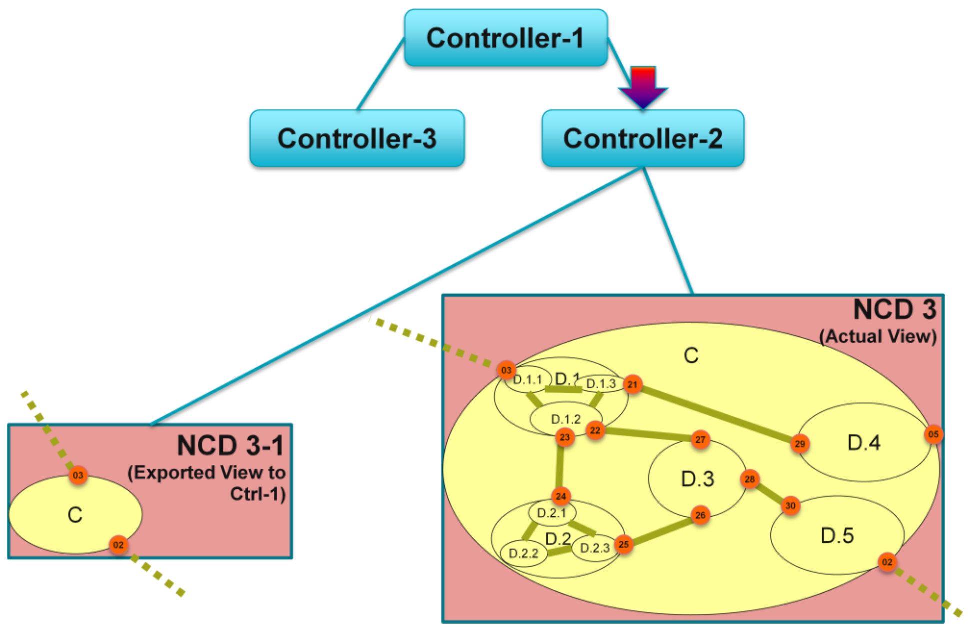


Figure 4: Views of Controller-2 NCDs

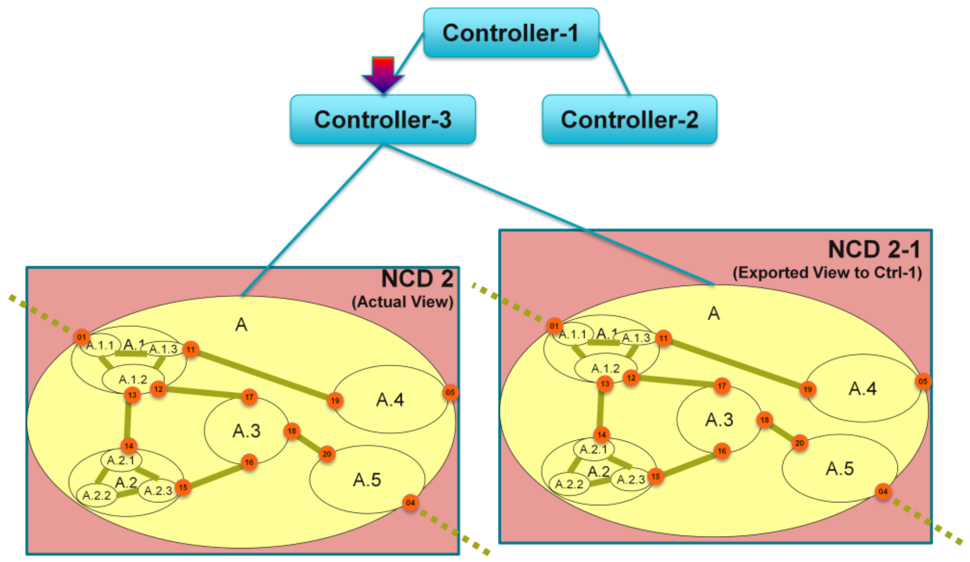


Figure 5: Views of Controller-3 NCDs

4.3 Topology Traversal using APIs

The following figures illustrate few examples of views of a provider topology, that a T-API client application may obtain using the APIs.

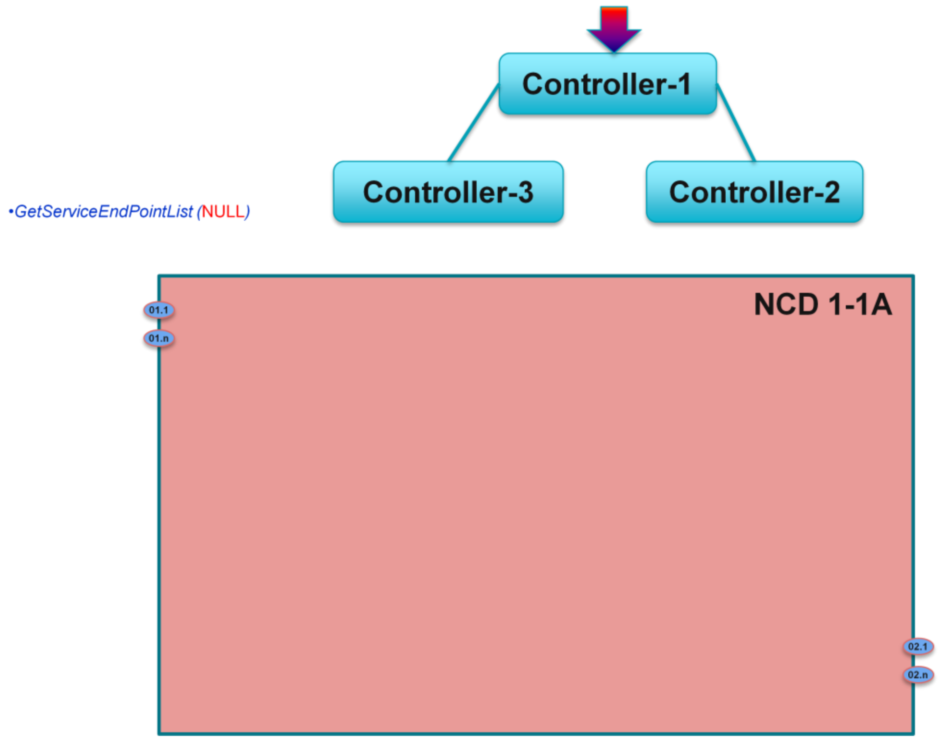
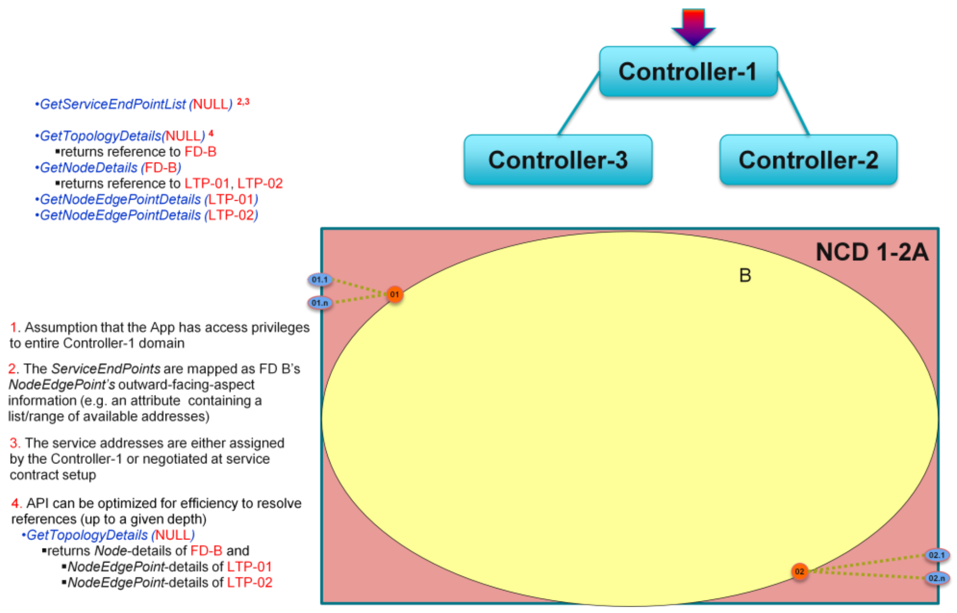


Figure 6: API Client’s View of Controller-1 NCD without retrieving Topology details


•*GetServiceEndPointList* (NULL) [2,3]

•*GetTopologyDetails*(NULL) [4]
  ▪returns reference to FD-B
•*GetNodeDetails* (FD-B)
  ▪returns reference to LTP-01, LTP-02
•*GetNodeEdgePointDetails* (LTP-01)
•*GetNodeEdgePointDetails* (LTP-02)

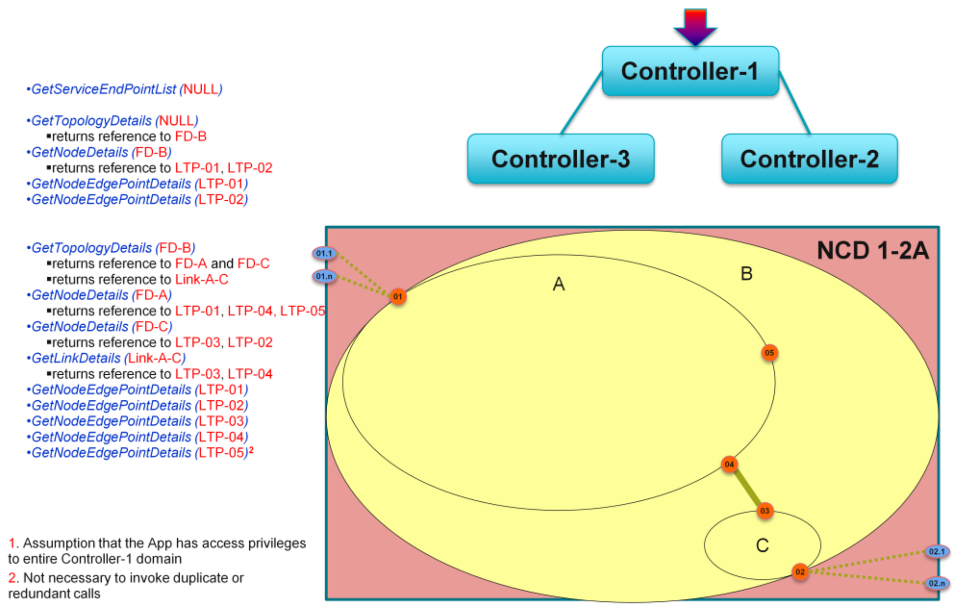
1. Assumption that the App has access privileges to entire Controller-1 domain

2. The *ServiceEndPoints* are mapped as FD B's *NodeEdgePoint's* outward-facing-aspect information (e.g. an attribute containing a list/range of available addresses)

3. The service addresses are either assigned by the Controller-1 or negotiated at service contract setup

4. API can be optimized for efficiency to resolve references (up to a given depth)
  •*GetTopologyDetails* (NULL)
    ▪returns *Node*-details of FD-B and
      ▪*NodeEdgePoint*-details of LTP-01
      ▪*NodeEdgePoint*-details of LTP-02

Figure 7: API Client's View of Controller-1 NCD by retrieving top-most level of Topology details



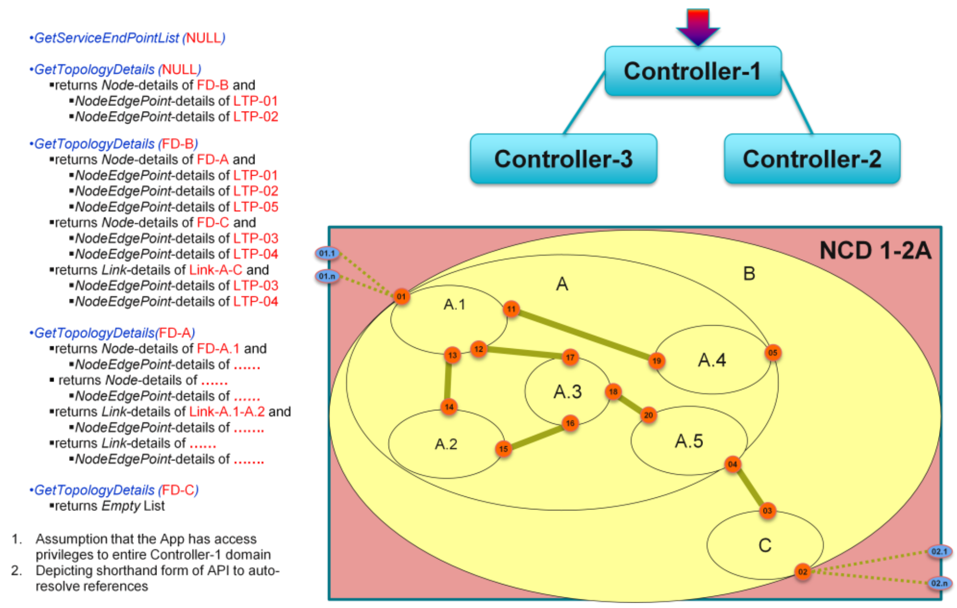
•*GetServiceEndPointList* (NULL)

•*GetTopologyDetails* (NULL)
  ▪returns reference to FD-B
•*GetNodeDetails* (FD-B)
  ▪returns reference to LTP-01, LTP-02
•*GetNodeEdgePointDetails* (LTP-01)
•*GetNodeEdgePointDetails* (LTP-02)

•*GetTopologyDetails* (FD-B)
  ▪returns reference to FD-A and FD-C
  ▪returns reference to Link-A-C
•*GetNodeDetails* (FD-A)
  ▪returns reference to LTP-01, LTP-04, LTP-05
•*GetNodeDetails* (FD-C)
  ▪returns reference to LTP-03, LTP-02
•*GetLinkDetails* (Link-A-C)
  ▪returns reference to LTP-03, LTP-04
•*GetNodeEdgePointDetails* (LTP-01)
•*GetNodeEdgePointDetails* (LTP-02)
•*GetNodeEdgePointDetails* (LTP-03)
•*GetNodeEdgePointDetails* (LTP-04)
•*GetNodeEdgePointDetails* (LTP-05)[2]

1. Assumption that the App has access privileges to entire Controller-1 domain
2. Not necessary to invoke duplicate or redundant calls

Figure 8: API Client's View of Controller1 NCD by retrieving 2 levels of Topology details



- *GetServiceEndPointList* (NULL)

- *GetTopologyDetails* (NULL)
  - returns *Node*-details of FD-B and
    - *NodeEdgePoint*-details of LTP-01
    - *NodeEdgePoint*-details of LTP-02

- *GetTopologyDetails* (FD-B)
  - returns *Node*-details of FD-A and
    - *NodeEdgePoint*-details of LTP-01
    - *NodeEdgePoint*-details of LTP-02
    - *NodeEdgePoint*-details of LTP-05
  - returns *Node*-details of FD-C and
    - *NodeEdgePoint*-details of LTP-03
    - *NodeEdgePoint*-details of LTP-04
  - returns *Link*-details of Link-A-C and
    - *NodeEdgePoint*-details of LTP-03
    - *NodeEdgePoint*-details of LTP-04

- *GetTopologyDetails*(FD-A)
  - returns *Node*-details of FD-A.1 and
    - *NodeEdgePoint*-details of ……
  - returns *Node*-details of ……
    - *NodeEdgePoint*-details of ……
  - returns *Link*-details of Link-A.1-A.2 and
    - *NodeEdgePoint*-details of …….
  - returns *Link*-details of ……
    - *NodeEdgePoint*-details of …….

- *GetTopologyDetails* (FD-C)
  - returns *Empty* List

1. Assumption that the App has access privileges to entire Controller-1 domain
2. Depicting shorthand form of API to auto-resolve references

Figure 9: API Client's View of Controller-1 NCD by retrieving 3 levels of Topology details

4.4 Service, Connection and Path

A *Service* represents an "intent-request" for connectivity between two or more *Edge-Points* (*LTPs*) exposed by the all-encompassing top-most *Forwarding-Domain* corresponding to the scope of the entire *Network-Control-Domain*. As such, *Service* is distinct from the *Connection* that realizes the *Service*. The requestor of the *Service* is expected to be able to express their intent using just an "external" view of *Forwarding-Domain* and its advertised *Edge-Points* and not require knowledge of the "internal" topology of the *Forwarding-Domain*.

The association of the *Service* to *LTPs* is made via the *Service-End-Points* which are essentially the ports of the *Service*.

The *Service* is modeled by the *Forwarding-Construct* entity defined in the ONF Core Information Model.

The *Connection* represents an enabled potential for forwarding (including all circuit and packet forms) between two or more Edge-Points (*LTPs)* of a *Forwarding-Domain*. A connection request typically utilizes its knowledge of the "internal" topology view of FD.

The association of the *Connection* to *LTPs* is made via *Connection-End-Points* (essentially the ports of the *Connection*) where each *End-Point* of the *Connection* has a role in the context of the *Connection*. The traffic forwarding between the associated *End-Points* of the *Connection* depends upon the type of *Connection*.

The *Connection* can be used to represent many different structures including point-to-point (P2P), point-to-multipoint (P2MP), rooted-multipoint (RMP) and multipoint-to-multipoint (MP2MP) bridge and selector structure for linear, ring or mesh protection schemes.

A *Connection* can be in only one *Forwarding-Domain*.

A *Connection* supports a recursive aggregation relationship such that the internal construction of a *Connection* can be exposed as multiple lower-level *Connection* instances (partitioning). A *Connection* can have zero or more *Paths*, each of which is defined as a list of lower level *Connection* instances. At the lowest level of recursion, a *Connection* represents a cross-connection within a switch matrix/fabric in a Network Element.

The *Connection* is modeled by the *Forwarding-Construct* entity defined in the ONF Core Information Model.

The *Path* represents the individual routes of a *Connection*. It is represented by a list of *Connections* at a lower level. Note that depending on the *Service* supported by a *Connection*, the *Connection* can have multiple routes.

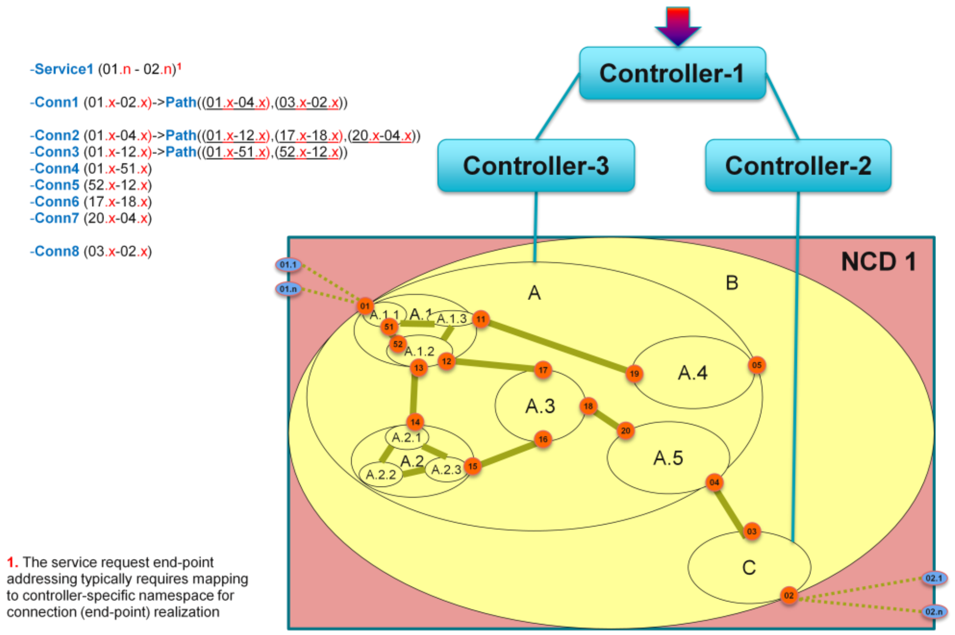

-Service1 (01.n - 02.n)1
-Conn1 (01.x-02.x)->Path((01.x-04.x),(03.x-02.x))
-Conn2 (01.x-04.x)->Path((01.x-12.x),(17.x-18.x),(20.x-04.x))
-Conn3 (01.x-12.x)->Path((01.x-51.x),(52.x-12.x))
-Conn4 (01.x-51.x)
-Conn5 (52.x-12.x)
-Conn6 (17.x-18.x)
-Conn7 (20.x-04.x)
-Conn8 (03.x-02.x)
1. The service request end-point addressing typically requires mapping to controller-specific namespace for connection (end-point) realization
Controller-1
Controller-3
Controller-2
NCD 1
A
B
C
A.1
A.1.1
A.1.2
A.1.3
A.2
A.2.1
A.2.2
A.2.3
A.3
A.4
A.5
01.1
01.n
02.1
02.n
01
51
52
11
12
13
14
15
16
17
18
19
20
05
04
03
02

Figure 10: Service & Connections from Controller-1 perspective

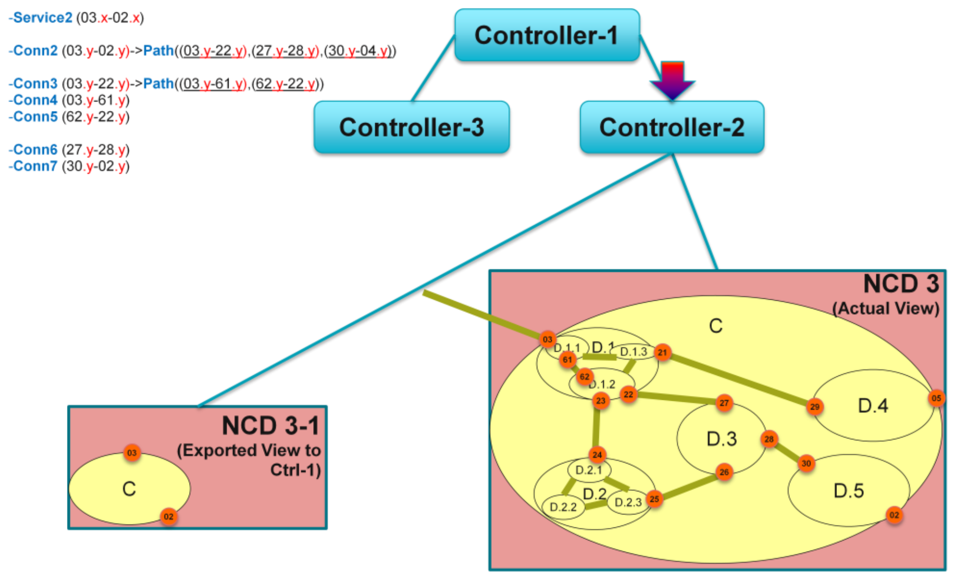


Figure 11: Service & Connections from Controller-2 perspective

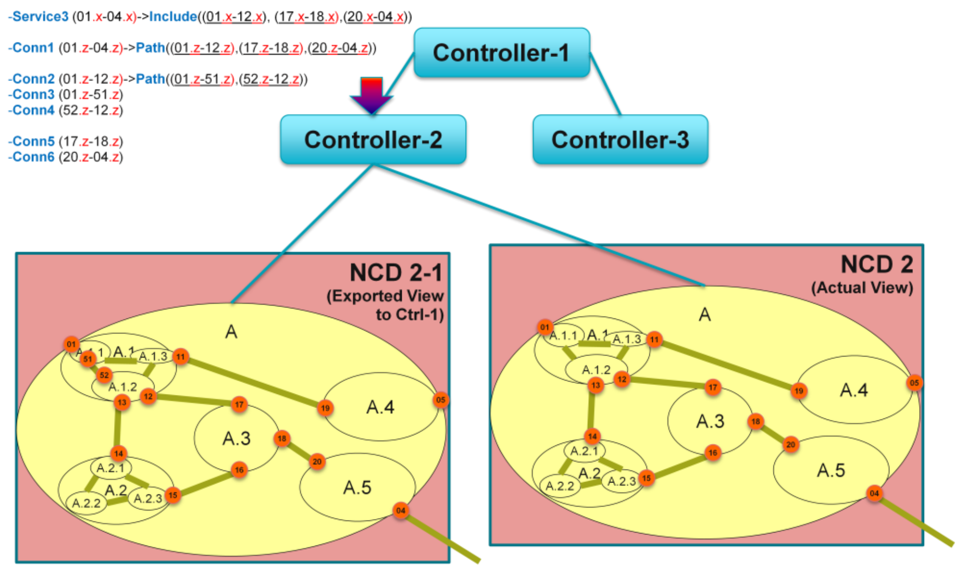

-Service3 (01.x-04.x)->Include((01.x-12.x), (17.x-18.x),(20.x-04.x))
-Conn1 (01.z-04.z)->Path((01.z-12.z),(17.z-18.z),(20.z-04.z))
-Conn2 (01.z-12.z)->Path((01.z-51.z),(52.z-12.z))
-Conn3 (01.z-51.z)
-Conn4 (52.z-12.z)
-Conn5 (17.z-18.z)
-Conn6 (20.z-04.z)
Controller-1
Controller-2
Controller-3
NCD 2-1
(Exported View to Ctrl-1)
NCD 2
(Actual View)
A
A.1
A.1.1
A.1.2
A.1.3
A.2
A.2.1
A.2.2
A.2.3
A.3
A.4
A.5

Figure 12: Service & Connections from Controller-3 perspective

4.5 Node Edge Point v/s Service End Point v/s Connection End Point

The *Logical-Termination-Point* (*LTP*) represents encapsulation of the addressing, mapping, termination, adaptation and OAM functions of one or more transport layers (including circuit and packet forms). Where the client – server relationship is fixed 1:1 and immutable, the different layers can be encapsulated in a single *LTP* instance. Where there is a n:1 relationship between client and server, the layers are split over separate instances of *LTP*.

Functions that can be associated/disassociated to/from an *Connection*, such as OAM, protection switching, and performance monitoring are referenced as secondary entities through the associated *LTP* instance.

5 Appendix B: Transport API Examples Use cases

Figure 13 below shows the reference physical network ("God View") where Service Provider (SP) physical NEs (colored in blue) are interconnected each other within the SP network and three Customer Edge (CE) NEs (colored in red) are connected through the SP network.

The circles represent the "interface number" of the physical interfaces attached to each node.

The UNI links (in read) are 10GE physical links while the NNI links (in blue) are 100G OTN physical links (OTU4).

It is also assumed that the CE NEs are IP routers using the SP network to setup IP links between them.

Interface P.4 (interface number 4 of node P) is not connected to any peer NE.

It is assumed that the whole SP network is managed by a single-domain SDN controller (SP Controller).

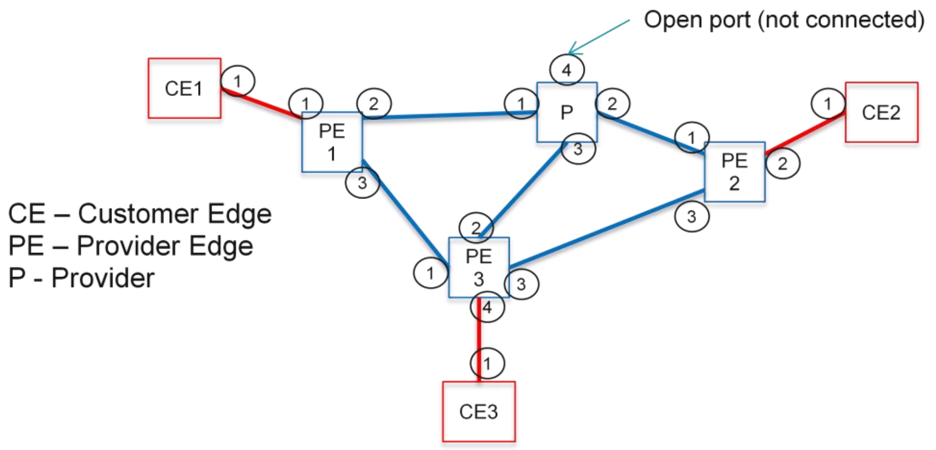


Figure 13: Example Physical Network Topology

In order to be able to setup a connectivity service, Service EndPoints, representing shared knowledge between the customer and the provider, needs to be pre-configured, based on customer and provider negotiation.

We consider three different scenarios where connectivity services can be requested:

a) In the simplest case, there is no information about an abstract network topology shared between the client application and service provider.

   In this case, only the service end points are shared knowledge.

Only the connectivity service APIs are used by the client to manage connectivity services between service end points. No path constraints can be requested in the connectivity setup request and no path information can be returned for a connection.

When a connectivity service request T-API is received, a connection controller within the service provider will internally call its path computation to setup the connection within the service provider network. This interaction is outside the scope of this document.

Topology and Path Computation T-APIs are not used between the client application and the service provider.

b. The client application and service provider can also have shared knowledge of an abstract network topology.

   The shared topology could be known a priori or retrieved via Topology API. This topology can be used to provide path constrains in the connectivity setup request and/or as a reference topology for returning the path of a connection.

   The client application can internally call its path computation to derive the path constraints, based on this shared network topology view, of a connectivity setup request. This interaction is outside the scope of this document.

   Path Computation T-APIs are not used.

c. When the client application and service provider can also have shared knowledge of an abstract network topology, a further enhancement is possible.

   Client application call the Path Computation T-API, with set of constrains based on abstracted NW view, to get a “list” of paths matching customer application constrains.

   Client application can use this information to provide path constraints in the Connectivity Request Setup T-API to force the SP controller to select the path it prefers from the list returned by the Path Computation API.

   This approach seems more useful in more complex scenarios e.g., a multi-domain network scenario where an orchestrator controller can request a domain controller to setup a sub-optimal path within its domain which would be part of the optimal multi-domain path.

5.1 10GE EPL Service over ODU2 Connection over 100G OTN network

In this use case the customer is willing to dynamically create a 10G IP Link between two of its CE routers connected to the SP network via two 10GE physical interfaces: for example an IP Link between CE1 and CE2 routers.

In order to support this use case in the reference network example, it is sufficient to pre-configure three Service EndPoints: X, Y and Z such that, to create an IP Link between CE1 and CE2, a 10GE EPL Service needs to be requested between SEPs X and Y.

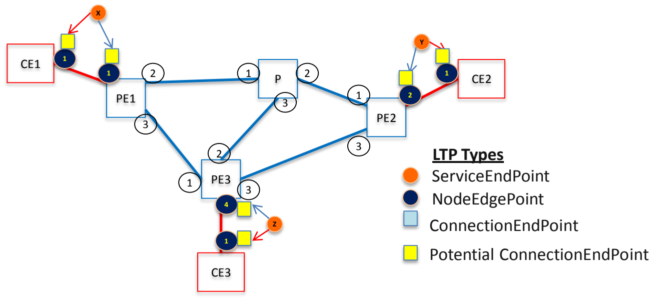


Figure 14: Example 10GE EPL Service over ODU2

In the shared T-API context, X is a pool of one and only one potential L-EC (Link Ethernet Connection, as defined in G.8021) Connection EndPoint (CEP).

The customer controller knows also the mapping between this potential L-EC CEP within the shared context and the potential L-EC CEP associated to the CE1, port 1, and therefore it can infer that the SEP X maps to that L-EC CEP within its context.

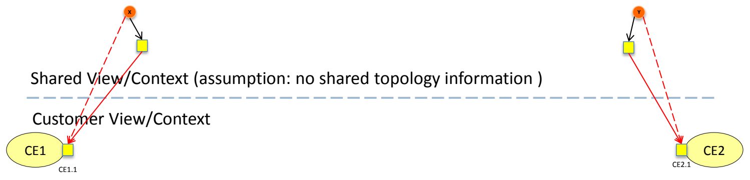

Shared View/Context (assumption: no shared topology information)
Customer View/Context
CE1
CE1.1
CE2
CE2.1

Figure 15: Customer View of Connectivity

In a similar manner, the SP controller can infer that the SEP X maps to the potential L-EC CEP, within its context, associated with PE1, port 1:

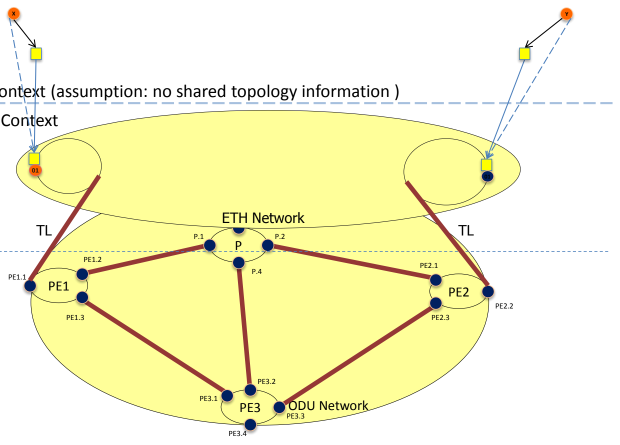
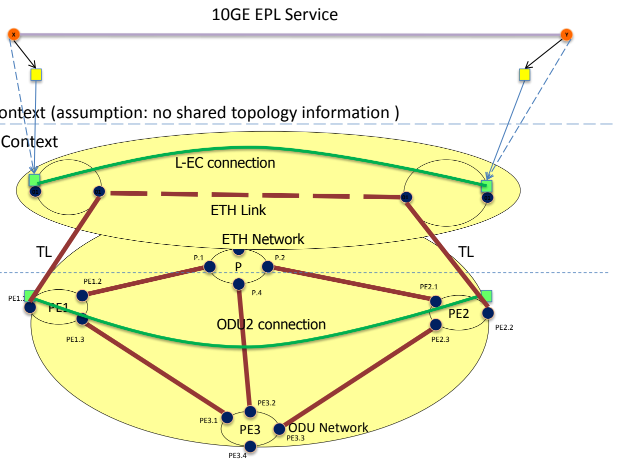


Figure 16: Provider's View of Topology and Service/Connections exported to the Customer

Similar one-to-one mappings apply to Y and Z Service EPs.

When the 10GE EPL service is requested, an L-EC (Link Ethernet Connection, as defined in G.8021.1) connection (between PE1.1 and PE2.2) within the SP network will be created:



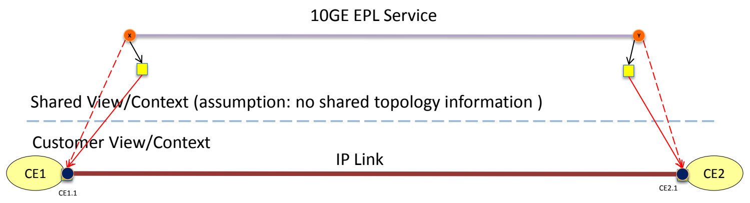
Three different implementations, within the service provider, are possible

- ODU2 connection
- Service EC (S-EC) connection
- PW connection

The choice can be based on network capability, service-provider policy, a pre-negotiated policy between customer and provider, dynamically chosen by the service provider controller e.g. based on the feedbacks from the path computation used within the SP controller.

Note – if there is a multi-layer shared abstract topology view, the path constraints of the service request can be used by the customer to constrain also the selection of the connection type. Detailed description of this use case is for further study.

As soon as the service is successfully created, the customer can create the IP Link, within the customer controller context, since there is a one-to-one mapping between the SEPs and the IP NEPs of the IP Link supported by the 10GE EPL Service:


10GE EPL Service
X
Y
Shared View/Context (assumption: no shared topology information)
Customer View/Context
IP Link
CE1
CE2
CE1.1
CE2.1

In this use case, there is one and only one L-EC connection within the shared context that can support the requested service. There is no need to report this L-EC connection, since it does not provide to the customer controller any additional information besides the fact that the service has been successfully setup.

5.2 1G EVPL Service over ODU0 Connection over 100G OTN network

In this use case the customer is willing to dynamically create a 1G IP Link between two of its CE routers, connected to the SP network via two 10GE physical interfaces which can be shared by different IP Links (using VLANs).

In order to support this use case in the reference network example, it is sufficient to pre-configure three SEPs: X, Y and Z such that, to create a 1G VLAN-based IP Link between CE1 and CE2, a 1G EVPL Service needs to be requested between SEPs X and Y.

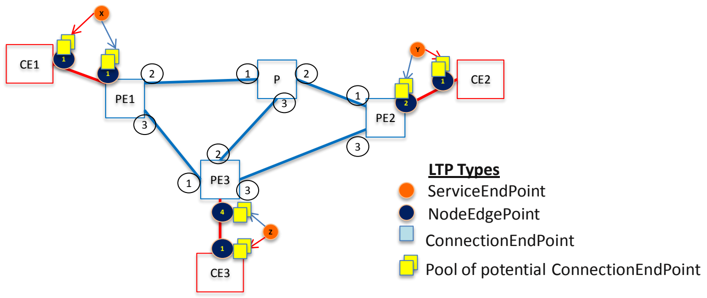


Figure 17: Example 1G EVPL Service over ODU0

In the shared T-API context, X is a pool of 4k C-EC (Customer Ethernet Connection, as defined in G.8021) potential CEPs.

The customer controller knows also the mapping between these 4k potential C-EC CEPs, within the shared context, and the 4k potential C-EC CEPs as within its context, associated to the CE1, port 1, and therefore it can infer that the SEP X maps to those 4k potential C-EC CEPs within its context.

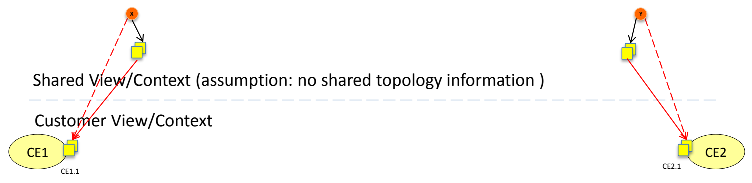

X
Y
Shared View/Context (assumption: no shared topology information)
Customer View/Context
CE1
CE1.1
CE2
CE2.1

Figure 18: Customer View of Connectivity

In a similar manner, the SP controller can infer that the SEP X maps to all the 4k potential C-EC CEPs, within its context, associated with PE1, port 1:

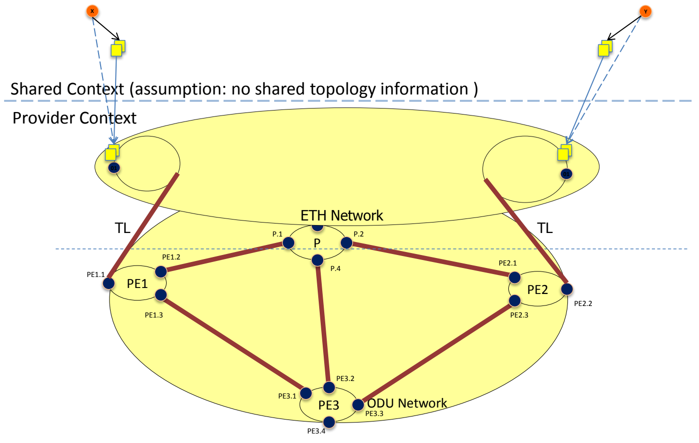
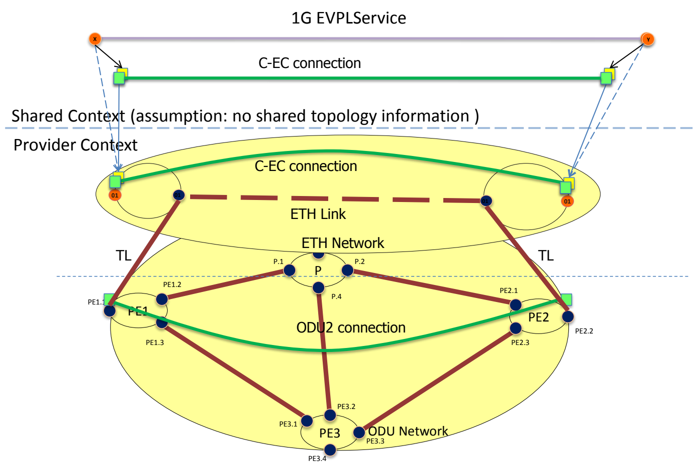


Figure 19: Provider's View of Topology and Service/Connections exported to the Customer

Similar mappings apply to Y and Z Service EPs.

When the 1G EVPL service is requested, a C-EC connection (between PE1.1 and PE2.2) within the SP network will be created:



Three different implementations, within the service provider, are possible

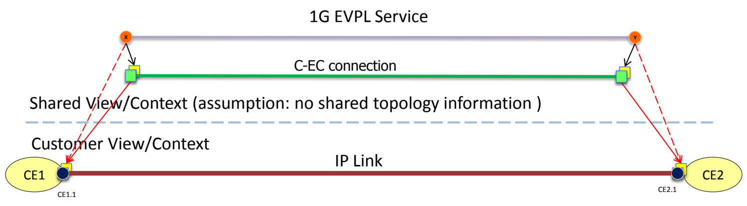
- ODU0 connection
- S-EC connection
- PW connection

The choice can be based on network capability, service-provider policy, a pre-negotiated policy between customer and provider, dynamically chosen by the service provider controller e.g. based on the feedbacks from the path computation used within the SP controller.

Note – if there is a multi-layer shared abstract topology view, the path constraints of the service request can be used by the customer to constrain also the selection of the connection type. Detailed description of this use case is for further study.

As soon as the service is successfully created, the Service Provider shall also report, within the shared context, a C-EC connection between C-EC CEPs that map to the actual C-EC CEPs, within the SP network. In particular, the actual C-EC CEPs, within the shared context, provide information of the C-VLAN ID values to be used at the edge of the SP network.

Based on the C-EC connection, the customer controller can create the 1G VLAN-based IP Link, supported by the EVPL Service, between IP NEPs that map to the actual C-EC CEPs, within the shared context. In particular, the configuration of the C-VLAN ID values to be used on the IP NEPs, within the customer context, is inferred from the information in the associated actual C-EC CEPs, within the shared context.



In this use case, there is many possible C-EC connections within the shared context that can support the requested service. There is a need to report, within the shared context, the actual C-EC connection implementing the requested service to provide the customer controller the information it needs to properly configure the IP NEPs within its own context (e.g., the C-VLAN ID values).

5.3 Var-rate EVPL Service over EVC Connection over 100G OTN network

For further study

5.4 EVPL Service with Load Balancing

Detailed description of this use case is for further study. This section just provides few guidelines:

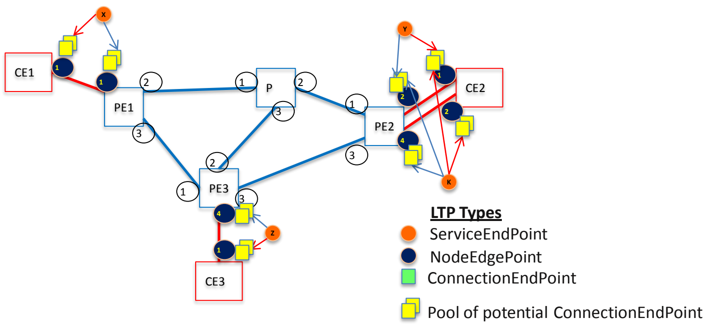


Figure 20: EVPL Service with Load Balancing

In order to support this use case in the reference network example, in addition to the X, Y and Z SEPs, another SEP K needs to be created, such that, to create a 1G VLAN-based IP Link between CE1 and CE2, a 1G EVPL Service needs to be requested between SEPs X and K.

K is a pool of 8k C-EC potential CEPs in the shared context which, within the context of the Customer controller, maps with all the 8k potential VLAN-based IP Node EndPoints that can be created over CE2, ports 1 and 2.

Within the Provider controller, these 8k C-EC potential CEPs map with all the 8k potential C-EC CEPs associated with PE2, ports 2 and 4.

It is worth noting that SEPs Y and K have an overlapping set of potential CEPs.

5.5 Anycast EVPL Service

Detailed description of this use case is for further study. This section just provides few guidelines:

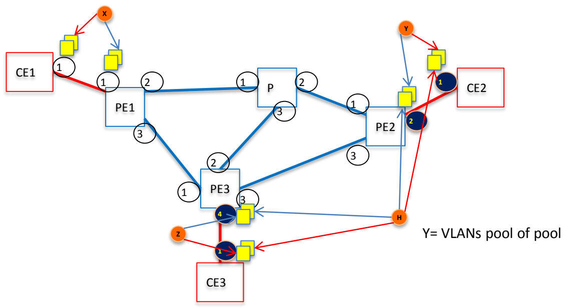


Figure 21: Example Anycast EVPL Service

In order to support this use case in the reference network example, in addition to the X, Y and Z SEPs, another SEP H needs to be created, such that, to create a 1G VLAN-based IP Link between CE1 and either CE2 or CE3, a 1G EVPL Service needs to be requested between SEPs X and H.

H is a pool of 8k C-EC potential CEPs in the shared context which, within the context of the Customer controller, maps with all the 8k potential VLAN-based IP Node EndPoints that can be created over CE2, port 1 and CE3, port 1.

Within the Provider controller, these 8k C-EC potential CEPs map with all the 8k potential C-EC CEPs associated with PE2, port 2 and PE3, port 1.

It is worth noting that SEPs Y and Z K have overlapping set of potential CEPs with SEP H.

6 Appendix C: Multi-layer and Multi-domain Use cases

This section gives the usage examples of TAPI information model and API in multi-layer and multi-domain network.

We assume a multi-layer and multi-domain network configuration as shown in the following figure. The network elements are packet OTN equipments. The network contains two domains, domain A and domain B, which are controlled by Controller-2 and Controller-3 respectively. Controller-1 controls the overall network (domain C) through Controller 2 and 3. Domain A and B are interconnected with OTU-4 links. The edge ports that connect to the client equipments are 10GE Ethernet ports.

An example service of 1GE EVPL over ODU0 (multi-layer and multi-domain service) between ServiceEndPoint 1 and ServiceEndpoint 2 is requested in this network.

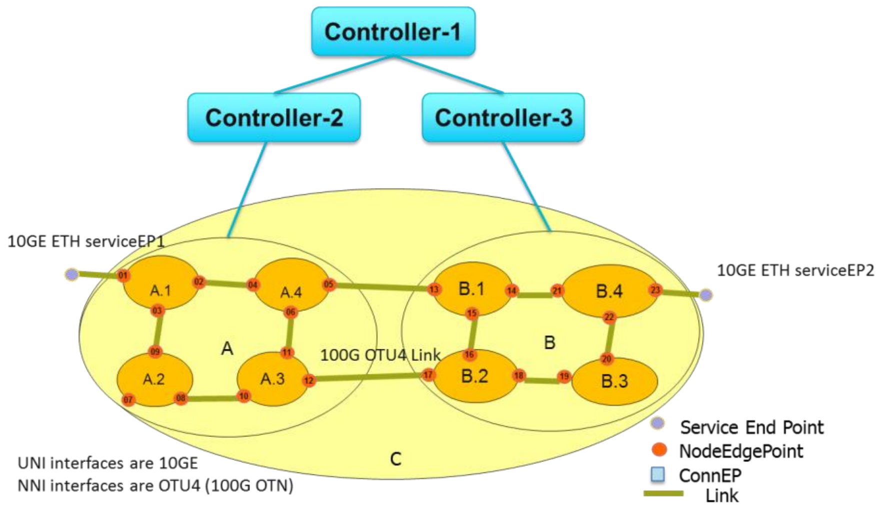


Figure 22: Multi-layer and Multi-domain Example Network Configuration

6.1 Multi-layer and Multi-domain Topology Initialization

We assume that all the 3 controllers are within one service provider's scope, so that all the internal topology in domain A and B are exposed to Controller 1. Since this is a multilayer network, there should be two topology instances for Ethernet and ODUk respectively. The two layer topologies are interconnected with transitional links between internal Ethernet port and ODU port (NodeEP 24, 25, 26, 27) inside node A.1 and B.4 (as shown in the following figure). This multi-layer topology enables cross-layer route computation in the controller. The serviceEndPoints at domain boundaries (serviceEP 3 and 4) should be instantiated for cross-domain service setup.

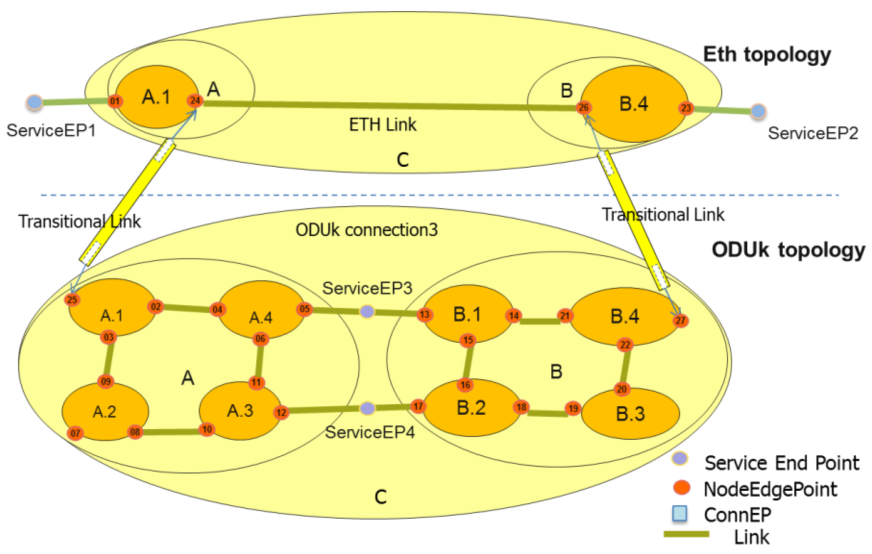


Figure 23: Network Topology in Controller 1

6.2 Multi-layer and multi-domain services/connections

To setup Ethernet over OTUk service in this multi-domain and multi-layer network, there are two options.

Option A: Setup multi-layer services within each domain.

This option allows controller 1 to send multi-layer service request to its subordinate controllers (Ethernet serviceEP to ODUk ServiceEP). The following are the API message exchange between controller 1 and controller 2, 3.

1. User sends ETH Connectivity Service Request to Controller 1.
2. Controller 1 receives ETH Connectivity Service Request.
3. Controller 1 computes multi-layer and multi-domain path using its knowledge of overall topology.
4. Controller 1 sends multilayer Connectivity Service 1 Request (between Ethernet serviceEP 1 to ODUk ServiceEP 3) to Controller 2.
5. Controller 2 receives multi-layer Connectivity Service Request from Controller 1.
6. Controller 2 computes multilayer path, and selects the internal NodeEPs for connection setup.
7. Controller 2 creates ODUk connection 1 and ETH connection 1 internally and returns them to Controller 1.

8. Controller 1 sends multilayer Connectivity Service 2 Request (between Ethernet serviceEP 1 to ODUk ServiceEP 3) to Controller 3.
9. Controller 3 receives multi-layer Connectivity Service Request from Controller 1.
10. Controller 3 computes multilayer path, and selects the internal NodeEPs for connection setup.
11. Controller 3 creates ODUk connection 2 and ETH connection 2 internally and returns them to Controller 1.
12. Controller 1 creates end-to-end ODUk connection 3 internally based on the received ODUk connection 1 and 2.
13. Controller 1 creates ETH link internally between A.1 and B.4 on top of connection 3.
14. Controller 1 creates the requested end-to-end ETH connection 3 based on the received ETH conection 1, 2 and ETH link internally.
15. Controller 1 returns ETH service to User.

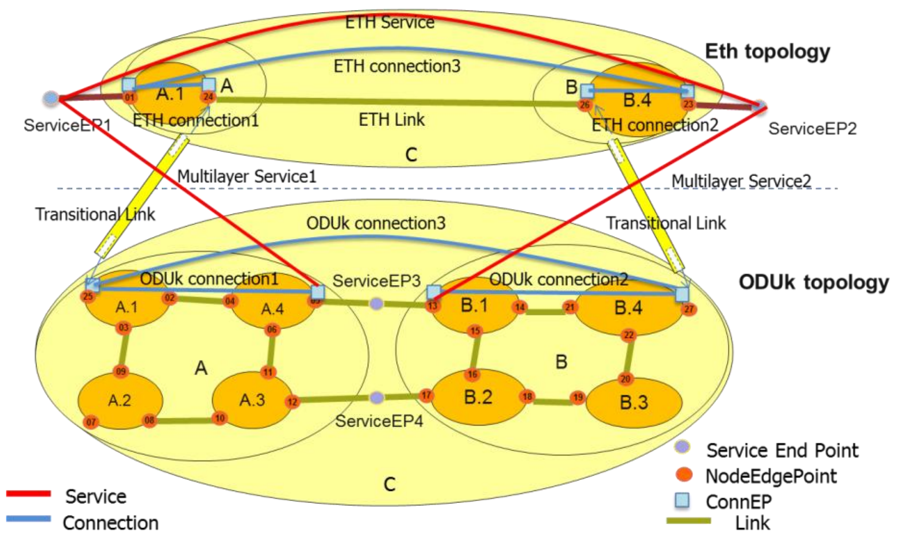


Figure 24: Multi-layer and Multi-domain service/connection setup (Option A)

Option B: Setup single-layer services within each domain.

This option only allows controller 1 to send single layer service request to its subordinate controllers. The following are the API message exchange between controller 1 and controller 2, 3.

1. User sends ETH Connectivity Service Request to Controller 1.
2. Controller 1 receives ETH Connectivity Service Request.
3. Controller 1 computes multi-layer and multi-domain path using its knowledge of overall topology.
4. Controller 1 sends ODU Connectivity Service Request (between ServiceEP3 and NodeEP 26) to Controller 2. (NodeEP 26 should also be assigned a serviceEP).
5. Controller 1 sends ETH Connectivity Service Request (between ServiceEP1 and NodeEP 24) to Controller 2. (NodeEP 24 should also be assigned a serviceEP).
6. Controller 2 creates requested ODUk connection 1 and ETH connection 1 internally and returns them to Controller 1.
7. Controller 1 sends ODU Connectivity Service Request (between ServiceEP3 and NodeEP 27) to Controller 3. (NodeEP 27 should also be assigned a serviceEP).
8. Controller 1 sends ETH Connectivity Service Request (between ServiceEP2 and NodeEP 28) to Controller 3. (NodeEP 28 should also be assigned a serviceEP).
9. Controller 3 creates requested ODUk connection 1 and ETH connection 1 internally and returns them to Controller 1.
10. Controller 1 creates end-to-end ODUk connection 3 internally based on the received ODUk connection 1 and 2.
11. Controller 1 creates ETH link internally between A.1 and B.4 on top of connection 3.
12. Controller 1 creates the requested end-to-end ETH connection 3 based on the received ETH conection 1, 2 and ETH link internally.
13. Controller 1 returns ETH service to User.

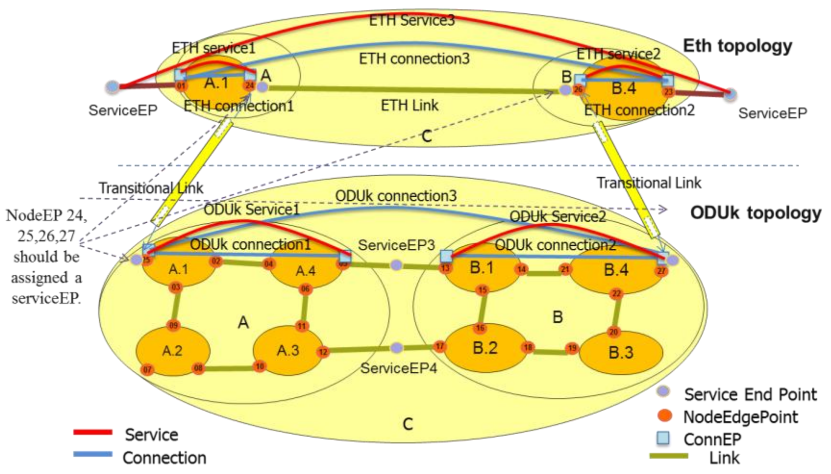


Figure 25: Multi-layer and Multi-domain service/connection setup (Option B)

6.3 Topology after service/connection Setup

After service and connection setup, the related connectionEndPoints and internal connections will be created. The following figure gives an example topology instance diagram of node A.1 after the EVPL service/connection setup. S-VLAN ConnEP in this figure is optional based on vendor implementation.

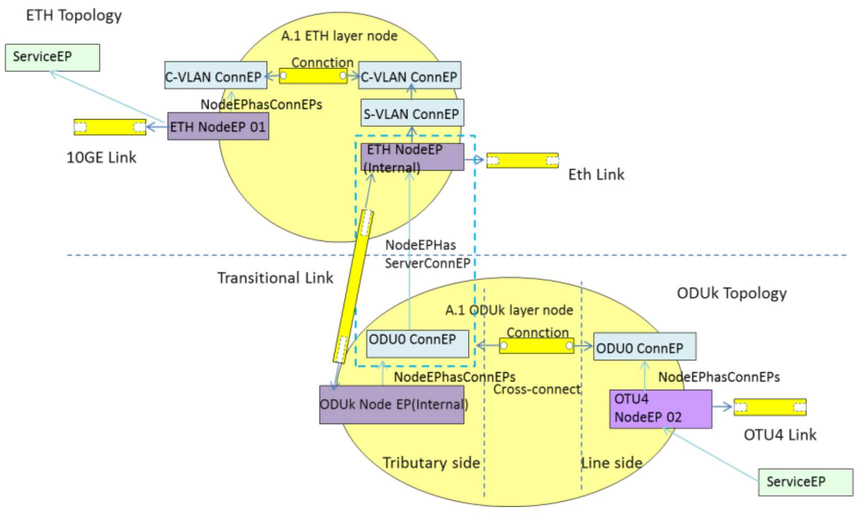


Figure 26: Topology instance diagram after service/connection setup

6.4 Further work

Multi-domain protection and multi-domain P2MP/MP2MP service use cases are for further study.

7 Appendix D: Transport API Information Model Skeleton

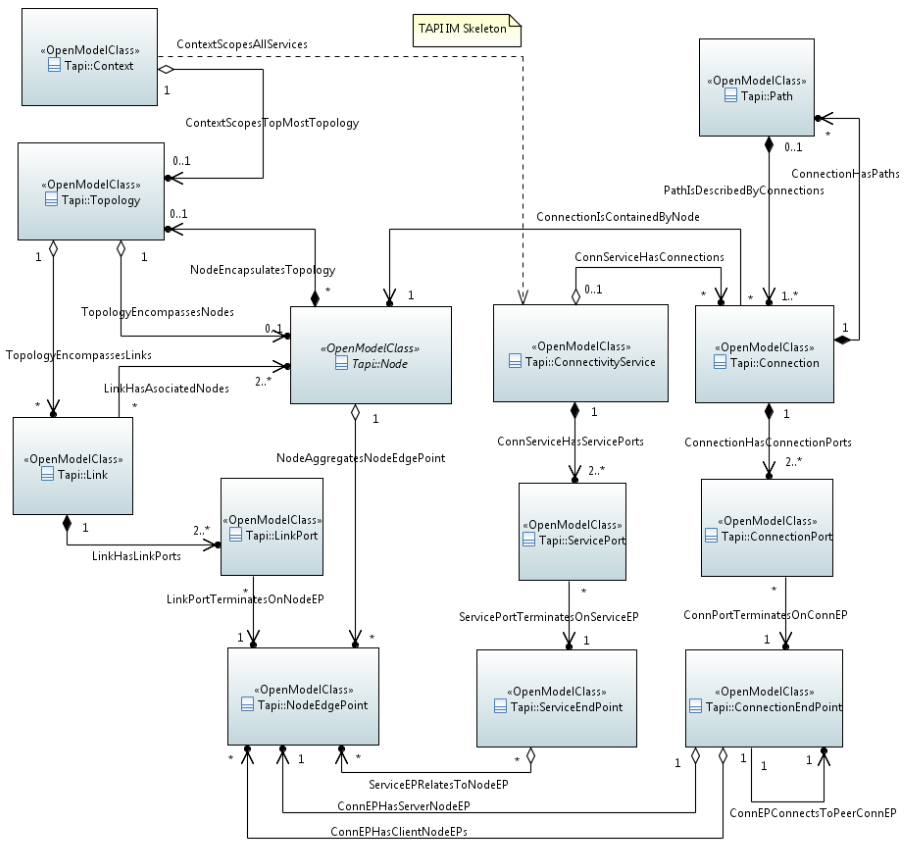

TAPI IM Skeleton
«OpenModelClass» Tapi::Context
«OpenModelClass» Tapi::Topology
«OpenModelClass» Tapi::Node
«OpenModelClass» Tapi::Link
«OpenModelClass» Tapi::LinkPort
«OpenModelClass» Tapi::NodeEdgePoint
«OpenModelClass» Tapi::ConnectivityService
«OpenModelClass» Tapi::ServicePort
«OpenModelClass» Tapi::ServiceEndPoint
«OpenModelClass» Tapi::Path
«OpenModelClass» Tapi::Connection
«OpenModelClass» Tapi::ConnectionPort
«OpenModelClass» Tapi::ConnectionEndPoint
ContextScopesAllServices
ContextScopesTopMostTopology
NodeEncapsulatesTopology
TopologyEncompassesNodes
TopologyEncompassesLinks
LinkHasAsociatedNodes
LinkHasLinkPorts
NodeAggregatesNodeEdgePoint
LinkPortTerminatesOnNodeEP
ConnectionIsContainedByNode
ConnServiceHasConnections
PathIsDescribedByConnections
ConnectionHasPaths
ConnServiceHasServicePorts
ConnectionHasConnectionPorts
ServicePortTerminatesOnServiceEP
ConnPortTerminatesOnConnEP
ServiceEPRelatesToNodeEP
ConnEPHasServerNodeEP
ConnEPHasClientNodeEPs
ConnEPConnectsToPeerConnEP

Figure 27: Transport API Information Model Skeleton

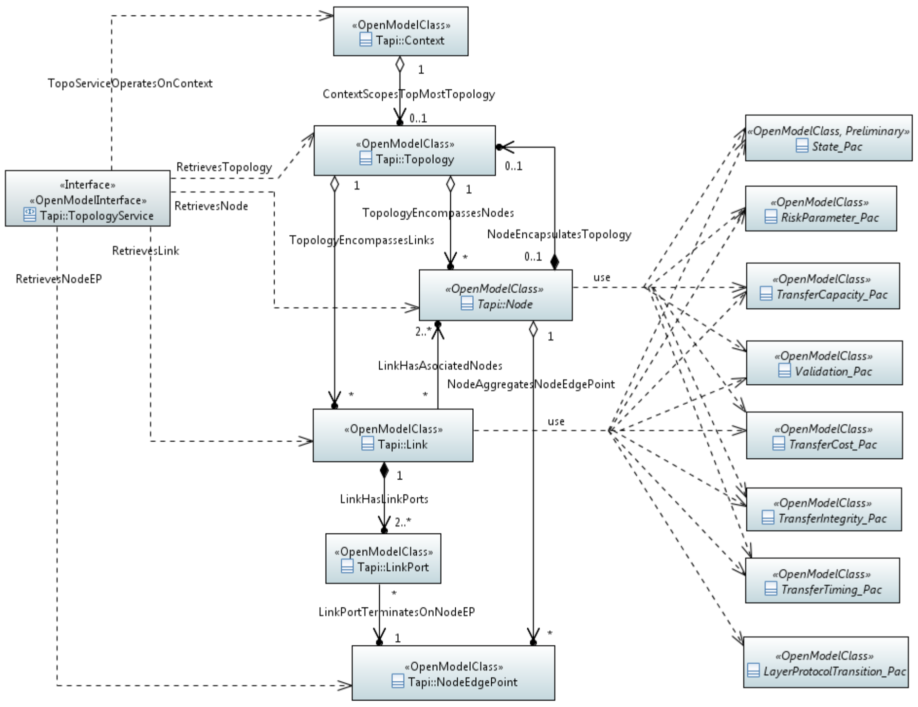

«OpenModelClass» Tapi::Context
TopoServiceOperatesOnContext
ContextScopesTopMostTopology
1
0..1
«OpenModelClass» Tapi::Topology
RetrievesTopology
RetrievesNode
RetrievesLink
RetrievesNodeEP
«Interface» «OpenModelInterface» Tapi::TopologyService
TopologyEncompassesNodes
TopologyEncompassesLinks
NodeEncapsulatesTopology
«OpenModelClass» Tapi::Node
LinkHasAsociatedNodes
NodeAggregatesNodeEdgePoint
«OpenModelClass» Tapi::Link
LinkHasLinkPorts
«OpenModelClass» Tapi::LinkPort
LinkPortTerminatesOnNodeEP
«OpenModelClass» Tapi::NodeEdgePoint
use
use
«OpenModelClass, Preliminary» State_Pac
«OpenModelClass» RiskParameter_Pac
«OpenModelClass» TransferCapacity_Pac
«OpenModelClass» Validation_Pac
«OpenModelClass» TransferCost_Pac
«OpenModelClass» TransferIntegrity_Pac
«OpenModelClass» TransferTiming_Pac
«OpenModelClass» LayerProtocolTransition_Pac

Figure 28: Topology Service API Skeleton

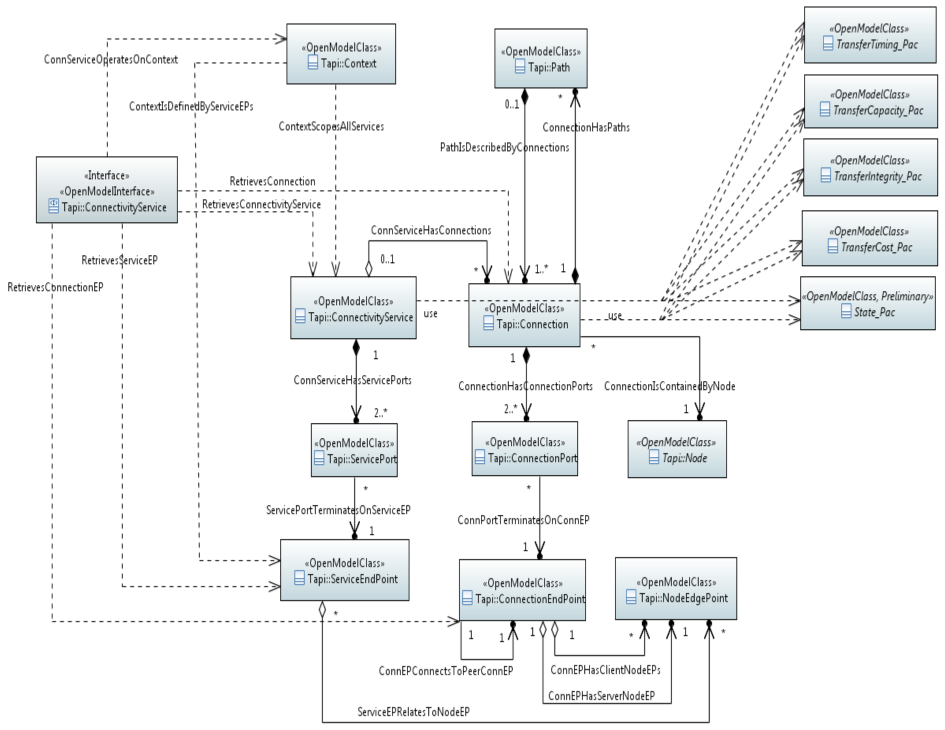

«OpenModelClass» Tapi::Context
«OpenModelClass» Tapi::Path
«OpenModelClass» TransferTiming_Pac
«OpenModelClass» TransferCapacity_Pac
«OpenModelClass» TransferIntegrity_Pac
«OpenModelClass» TransferCost_Pac
«OpenModelClass, Preliminary» State_Pac
«Interface» «OpenModelInterface» Tapi::ConnectivityService
«OpenModelClass» Tapi::ConnectivityService
«OpenModelClass» Tapi::Connection
«OpenModelClass» Tapi::ServicePort
«OpenModelClass» Tapi::ConnectionPort
«OpenModelClass» Tapi::Node
«OpenModelClass» Tapi::ServiceEndPoint
«OpenModelClass» Tapi::ConnectionEndPoint
«OpenModelClass» Tapi::NodeEdgePoint
ConnServiceOperatesOnContext
ContextIsDefinedByServiceEPs
ContextScopesAllServices
PathIsDescribedByConnections
ConnectionHasPaths
RetrievesConnection
RetrievesConnectivityService
ConnServiceHasConnections
RetrievesServiceEP
RetrievesConnectionEP
use
use
ConnServiceHasServicePorts
ConnectionHasConnectionPorts
ConnectionIsContainedByNode
ServicePortTerminatesOnServiceEP
ConnPortTerminatesOnConnEP
ConnEPConnectsToPeerConnEP
ConnEPHasClientNodeEPs
ConnEPHasServerNodeEP
ServiceEPRelatesToNodeEP
0..1
*
0..1
1..*
1
1
2..*
1
2..*
*
1
*
1
*
1
*
1
1
1
1
*
1
*

Figure 29: Connectivity Service API Skeleton

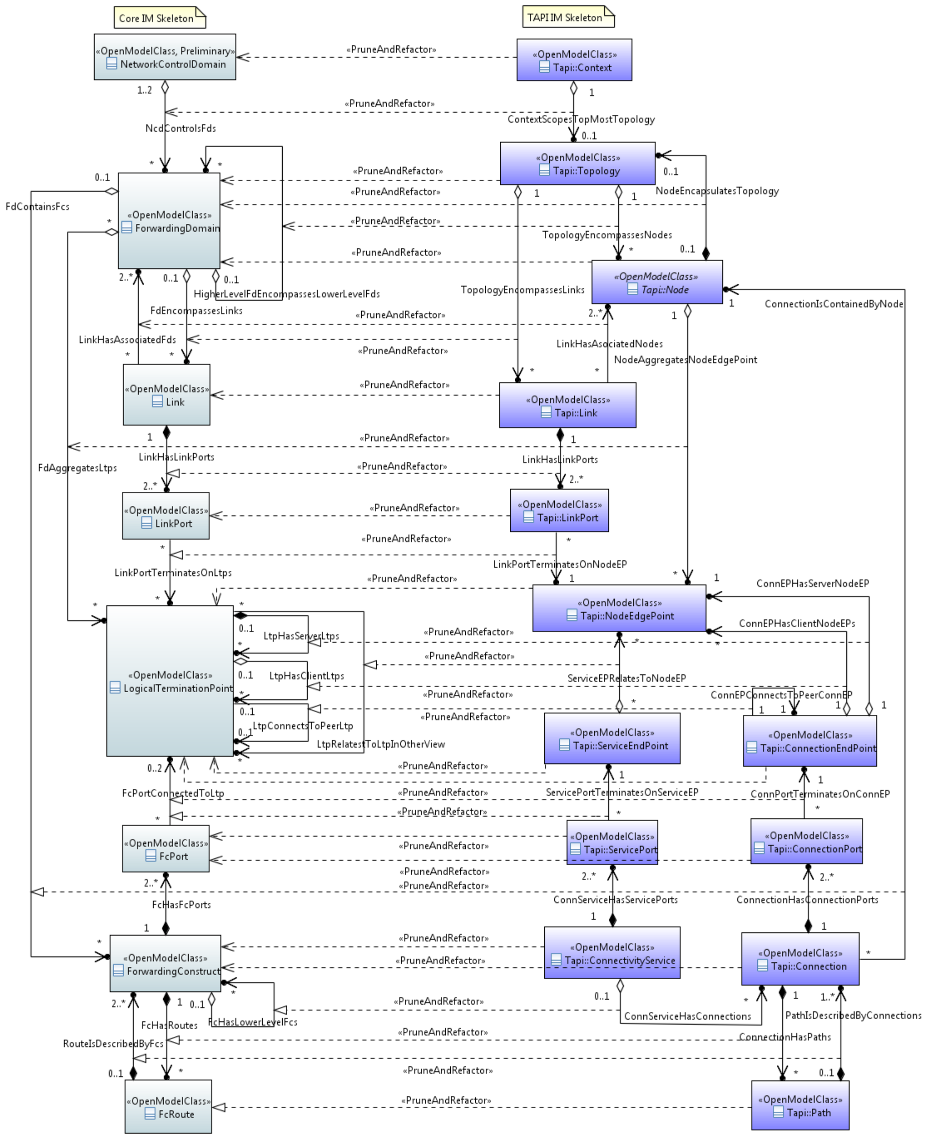

Core IM Skeleton
TAPI IM Skeleton
«OpenModelClass, Preliminary» NetworkControlDomain
«OpenModelClass» Tapi::Context
«PruneAndRefactor»
ContextScopesTopMostTopology
NcdControlsFds
«OpenModelClass» Tapi::Topology
NodeEncapsulatesTopology
«OpenModelClass» ForwardingDomain
FdContainsFcs
TopologyEncompassesNodes
«OpenModelClass» Tapi::Node
HigherLevelFdEncompassesLowerLevelFds
TopologyEncompassesLinks
FdEncompassesLinks
ConnectionIsContainedByNode
LinkHasAssociatedFds
LinkHasAssociatedNodes
NodeAggregatesNodeEdgePoint
«OpenModelClass» Link
«OpenModelClass» Tapi::Link
LinkHasLinkPorts
FdAggregatesLtps
«OpenModelClass» LinkPort
«OpenModelClass» Tapi::LinkPort
LinkPortTerminatesOnNodeEP
LinkPortTerminatesOnLtps
ConnEPHasServerNodeEP
«OpenModelClass» Tapi::NodeEdgePoint
ConnEPHasClientNodeEPs
LtpHasServerLtps
LtpHasClientLtps
«OpenModelClass» LogicalTerminationPoint
ServiceEPRelatesToNodeEP
ConnEPConnectsToPeerConnEP
LtpConnectsToPeerLtp
LtpRelatestToLtpInOtherView
«OpenModelClass» Tapi::ServiceEndPoint
«OpenModelClass» Tapi::ConnectionEndPoint
FcPortConnectedToLtp
ServicePortTerminatesOnServiceEP
ConnPortTerminatesOnConnEP
«OpenModelClass» FcPort
«OpenModelClass» Tapi::ServicePort
«OpenModelClass» Tapi::ConnectionPort
FcHasFcPorts
ConnServiceHasServicePorts
ConnectionHasConnectionPorts
«OpenModelClass» ForwardingConstruct
«OpenModelClass» Tapi::ConnectivityService
«OpenModelClass» Tapi::Connection
FcHasRoutes
FcHasLowerLevelFcs
ConnServiceHasConnections
PathIsDescribedByConnections
RouteIsDescribedByFcs
ConnectionHasPaths
«OpenModelClass» FcRoute
«OpenModelClass» Tapi::Path

Figure 30: Transport API IM Mapping to Core IM

8 Contributors

The Transport API Design team responsible for the writing of this document included:

- Eve Varma, ALU
- Guoying Zhang, CATR
- Italo Busi, Huawei
- Jia He, Huawei
- Karthik Sethuraman, NEC (Editor)
- Lyndon Ong, Ciena
- Nigel Davis, Ciena
- Sergio Bellotti, ALU

Special thanks to Chen Qiaogang, Erez Segev, Hui Ding, Ricard Vilalta, Victor Lopez, and others for their input and comments.

9 Version History

ONF2015.087.xx xx=	**DATE**	**VERSION COMMENTS**
00	Jan 9, 2015	Initial Draft
01	Feb 10, 2015	Draft Topology, Service and Virtual Network Service requirements from Karthik, Lyndon, Jia
02	Mar 6, 2015	Updates to Service, Draft Connection and Path requirements from Lyndon, Sergio and Shinji
03	Mar 19,2015	Updates and comments to Path Computation sections from Sergio and Jia
04	April 08,2015	Added definitions and some cleanup (resolved comments and changes) by Karthik and Lyndon
05	April 23, 2015	Topology/Service section comments cleanup by Karthik and Path Computation section cleanup by Sergio
06	May 14, 2015	Updated Definitions and Topology sections to reflect the latest design team discussion outcomes and to use Core IM terminology
0.7	June 3, 2015	Merged comments for review at the Darmstadt F2F. Clarified NULL FD concepts
0.8	June 11, 2015	Post Darmstadt F2F version with cleanup and accepting comments and changes. Version liaised to OIF.

0.9	Aug 17,2015	- Added/merged comments from Chen Qiaogang, Hui Ding, Ricard Vilalta, Victor Lopez, Erez Segev, Eve Varma and others. - Added initial draft of appendices for Concepts, Use cases and IM. - Rewrote the Terms and Definitions section. - Reorganized the Requirements section and API headings - Updated the Topology section to use TAPI terminology and removed redundant APIs - Added the Connectivity Retrieval API requirements - Removed the Connection Control section for now – may be added back in future
0.10	Sep 28, 2015	- Cleaned up the change bars and addressed comments from version 0.9 - Added additional text to the TAPI Functional Architecture section - Removed any direct dependency of the Service APIs on the Topology API - Renamed TAPI *View* to *Context*
0.11	Sept 30, 2015	- Cleaned up the change bars and addressed comments from version 0.10 - Updated Terminology, introductory sections - Added Lifecycle State - Separated the Connectivity Request API's Requirements and Constraints into their own requirements table in the Input/Output section
0.12	Dec 21, 2015	- Cleaned up the change bars and addressed comments from version 0.11 - Updated Terminology & introductory sections - Included lifecycle dependency constraint by passing "containing" parent id as required input in the retrieval APIs - Added support for multiple Contexts by passing in Context Id where appropriate - Added Topology Data Types section (3.1.2) - Added Notification Service section (3.5) - Updated the Appendix B with the Use Cases from Italo & Sergio - Updated the Appendix C with the Use Cases from Guoying - Updated the Appendix D with latest IM skeleton snapshot

0.13	Jan 21, 2016	- Cleaned up the change bars and addressed comments from version 0.12 - Deferred support for multiple Contexts - Updated Connectivity Constraints - Updated Path Computation Constraints - Consolidated all Data Types into a single section - Reformatting and editorial changes