OPEN NETWORKING
FOUNDATION

Functional Requirements for Transport API

June 10, 2016

ONF TR-527

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

1 Introduction 8
1.1 Purpose 8
1.2 Scope 9
1.3 References 9
1.4 Abbreviations 9
1.5 Terms and Definitions 10
1.6 Conventions 12

2 Functional Architecture 13

3 Functional Requirements 15
3.1 Topology Service 15
3.1.1 Topology Retrieval APIs 15
3.2 Connectivity Service 17
3.2.1 Connectivity Retrieval APIs 18
3.2.2 Connectivity Request APIs 21
3.3 Path Computation Service 23
3.3.1 Path Computation Request APIs 23
3.4 Virtual Network Service 25
3.4.1 Virtual Network Retrieval APIs 25
3.4.2 Virtual Network Request APIs 26
3.5 Notification Service 27
3.5.1 Notification Subscription and Filtering APIs 28
3.5.2 Notification Message Types 31
3.6 TAPI Data Types 32

4 Appendix A: Transport API Concepts Overview 38
4.1 Context 38
4.2 Node and Topology Aspects of Forwarding Domain 39
4.3 Hierarchical Control Domains and Contexts 42
4.4 Topology Traversal using APIs 44
4.5 Service, Connection and Route 46
4.6 Node Edge Point v/s Service End Point v/s Connection End Point 48

5 Appendix B: Transport API Examples Use cases 50
5.1 10GE EPL Service over ODU2 Connection over 100G OTN network 51
5.2 1G EVPL Service over ODU0 Connection over 100G OTN network 54
5.3 Var-rate EVPL Service over EVC Connection over 100G OTN network 57
5.4 EVPL Service with Load Balancing 57
5.5 Anycast EVPL Service 58

6 Appendix C: Multi-layer and Multi-domain Use cases 60
6.1 Multi-layer and Multi-domain Topology Initialization 60
6.2 Multi-layer and multi-domain services/connections 61

6.3 Topology after service/connection Setup 64
6.4 Further work 65

7 Appendix D: Transport API Information Model Skeleton 66

8 Contributors 71

9 Version History 71

List of Figures

Figure 1: T-API Artifacts - ONF/OSSDN Project Dependencies 8
Figure 2: Transport API Functional Architecture 14
Figure 3 : Simple Physical Network Example 38
Figure 4: Shared Contexts - Architecture perspective 39
Figure 5: Shared Contexts & Topology 39
Figure 6: Topological Decomposition of Node 40
Figure 7: Recursive Topological Decomposition of Node 41
Figure 8: Node/Topology perspectives of recursively partitioned Forwarding Domain (FD) 41
Figure 9: View of Controller-1 Context based on Views exported by Controllers 2 & 3 42
Figure 10: Views of Controller-2 Contexts 43
Figure 11: Views of Controller-3 Contexts 43
Figure 12: API Client’s View of Controller-1 Context without retrieving Topology details 44
Figure 13: API Client’s View of Controller-1 Context by retrieving top-most level of Topology 44
Figure 14: API Client’s View of Controller1 Context by retrieving 2 levels of Topology details 45
Figure 15: API Client’s View of Controller-1 Context by retrieving 3 levels of Topology details 45
Figure 16: Service & Connections from Controller-1 perspective 47
Figure 17: Service & Connections from Controller-2 perspective 47
Figure 18: Service & Connections from Controller-3 perspective 48
Figure 19: Example Physical Network Topology 50
Figure 20: Example 10GE EPL Service over ODU2 52
Figure 21: 10G EPL - Customer View of Connectivity 52
Figure 22: 10G EPL - Provider’s View of Topology exported to the Customer 53
Figure 23: 10G EPL - Provider’s View of Service/Connections exported to the Customer 53
Figure 24: 10G EPL - Customer's view of Service/Connectivity 54

Figure 25: Example 1G EVPL Service over ODU0 55
Figure 26: 1G EVPL - Customer View of Connectivity 55
Figure 27: 1G EVPL - Provider's View of Topology exported to the Customer 56
Figure 28: 1G EVPL - Provider's View of Service/Connections exported to the Customer 56
Figure 29: 1G EVPL - Customer's view of Service/Connectivity 57
Figure 30: EVPL Service with Load Balancing 58
Figure 31: Example Anycast EVPL Service 58
Figure 32: Multi-layer and Multi-domain Example Network Configuration 60
Figure 33: Network Topology in Controller 1 61
Figure 34: Multi-layer and Multi-domain service/connection setup (Option A) 62
Figure 35: Multi-layer and Multi-domain service/connection setup (Option B) 64
Figure 36: Topology instance diagram after service/connection setup 64
Figure 37: Transport API Information Model Skeleton 66
Figure 38: Topology Service Skeleton 67
Figure 39: Connectivity Service Skeleton 68
Figure 40: Virtual Network Service Skeleton 69
Figure 41: Path Computation Service Skeleton 70

List of Functional Requirement Tables

TAPI_FR 1: Get Topology List 15
TAPI_FR 2: Get Topology Details 15
TAPI_FR 3: Get Node Details 16
TAPI_FR 4: Get Link Details 16
TAPI_FR 5: Get Node Edge Point Details 17
TAPI_FR 6: Get Service End Point List 18
TAPI_FR 7: Get Service End Point Details 18
TAPI_FR 8: Get Connectivity Service List 18
TAPI_FR 9: Get Connectivity Service Details 19
TAPI_FR 10: Get Connection Details 20
TAPI_FR 11: Get Connection End Point Details 20
TAPI_FR 12: Create Connectivity Service 21
TAPI_FR 13: Update Connectivity Service 22

TAPI_FR 14: Delete Connectivity Service 22
TAPI_FR 15: Compute P2P Path 23
TAPI_FR 16: Optimize P2P Path 24
TAPI_FR 17: Get Virtual Network Service List 25
TAPI_FR 18: Get Virtual Network Service Details 25
TAPI_FR 19: Create Virtual Network Service 26
TAPI_FR 20: Delete Virtual Network Service 27
TAPI_FR 21: Discover Supported Notification Types 28
TAPI_FR 22: Create Notification Subscription 29
TAPI_FR 23: Modify Notification Subscription 29
TAPI_FR 24: Delete Notification Subscription 30
TAPI_FR 25: Suspend Notification Subscription 30
TAPI_FR 26: Resume Notification Subscription 30
TAPI_FR 27: Retrieve Notification Records 30
TAPI_FR 28: Object Creation Notification 31
TAPI_FR 29: Object Deletion Notification 31
TAPI_FR 30: Attribute Value Change Notification 31
TAPI_FR 31: State Change Notification 32
TAPI_FR 32: Layer Protocol Name 32
TAPI_FR 33: Capacity (Fixed Bandwidth) 32
TAPI_FR 34: Capacity (Profile) 33
TAPI_FR 35: Administration State 33
TAPI_FR 36: Operational State 33
TAPI_FR 37: Lifecycle State 34
TAPI_FR 38: Port Role 34
TAPI_FR 39: Port Direction 34
TAPI_FR 40: Termination Direction 34
TAPI_FR 41: Service End Point TRI format 34
TAPI_FR 42: Service Type 35
TAPI_FR 43: Connectivity Constraints 35
TAPI_FR 44: Virtual Network Service Constraints 35
TAPI_FR 45: Traffic Matrix 36
TAPI_FR 46: Path Optimization Constraint 36

TAPI_FR 47: Path Objective Function 36
TAPI_FR 48: Notification-Header 36
TAPI_FR 49: Notification-Type 37
TAPI_FR 50: Object-Type 37
TAPI_FR 51: Notification-Source-Indicator 37

1 Introduction

The 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 Document" (FRD) 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 upon several use cases that were deemed key for the application domain, the APIs have been developed with the intent of not precluding their being employed by additional valid application use cases.

The requirements specified in this document are intended to drive the detailed UML information model specifications, from which the YANG/JSON schema and Swagger APIs are generated. The following figure illustrates the inter-relationships between various T-API project artifacts:

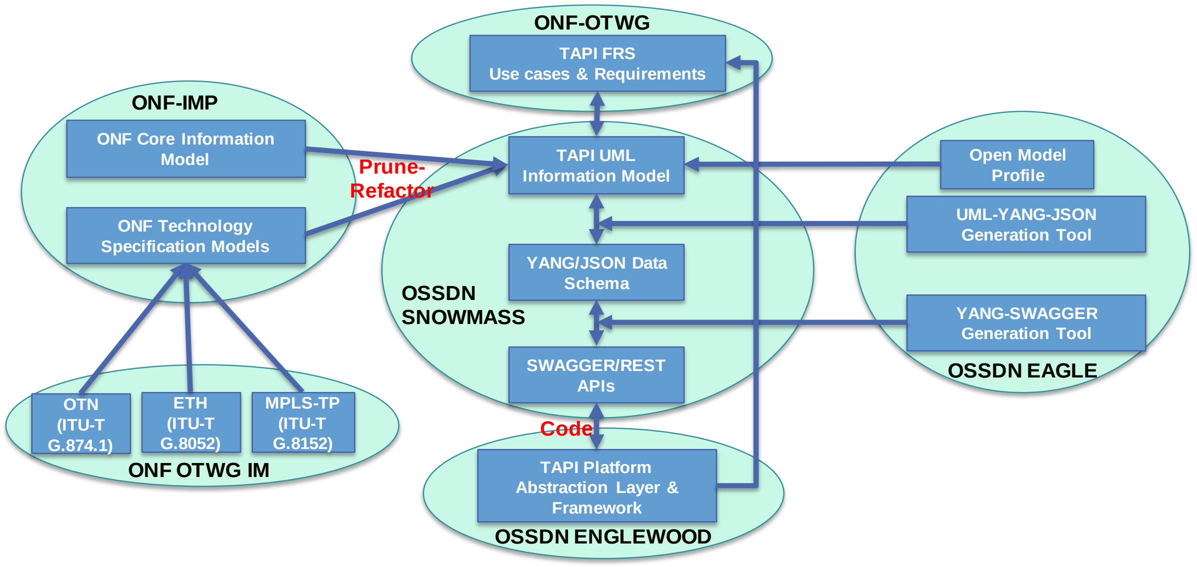

ONF-OTWG
TAPI FRS Use cases & Requirements
ONF-IMP
ONF Core Information Model
ONF Technology Specification Models
Prune-Refactor
TAPI UML Information Model
YANG/JSON Data Schema
SWAGGER/REST APIs
OSSDN SNOWMASS
Open Model Profile
UML-YANG-JSON Generation Tool
YANG-SWAGGER Generation Tool
OSSDN EAGLE
OTN (ITU-T G.874.1)
ETH (ITU-T G.8052)
MPLS-TP (ITU-T G.8152)
ONF OTWG IM
Code
TAPI Platform Abstraction Layer & Framework
OSSDN ENGLEWOOD

Figure 1: T-API Artifacts - ONF/OSSDN Project Dependencies

1.2 Scope

This issue of the document specifies APIs for 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 distinct/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
TRI	Transport Resource Identifier

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.

Context (API Context)

The T-API defines the scope of control, interaction 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.

Topology The T-API *Topology* is an abstract representation of the topological-aspects of a particular set of *Network Resources*. It is described in terms of the underlying topological network of *Nodes* and *Links* that enable the forwarding capabilities of that particular set of *Network Resources*.

Node

The T-API *Node* is an abstract representation of the forwarding-capabilities of a particular set of *Network Resources*. It is described in terms of the aggregation of set of ports (*Node-Edge-Point*) belonging to those *Network Resources* and the potential to enable forwarding of information between those edge ports.

Link

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

TE Link

The T-API *(Traffic Engineered)TE-Link*[1] is an abstract representation of the effective adjacency between two[2] associated *Nodes* (or *NodeEdgePoints*) in a *Topology*, that has TE properties and is used in the description of the output of path computation APIs. It is terminated by *Node-Edge-Points* of the associated *Nodes*.

[1] The TAPI TE-Link reflects the TE-Link as defined in the RFC-4202
[2] A TAPI Link could be a multi-point entity, with more than two end points.

Node-Edge-Point

The T-API *Node-Edge-Point* represents the inward network-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*.

Service-End-Point

The T-API *Service-End-Point* represents the outward customer-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. shared addressing, capacity, resource availability, etc.), that enable the clients to request connectivity 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*.[3]

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*.

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

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.

Route (Connection Route)

The T-API *Route* represents the route of a *Connection* through the *Nodes* in the underlying *Topology*. It is described as a list of references to the underlying *Connections*.[4]

Path

The TAPI *Path* is used to represent the output of path computation APIs and is described by an ordered list of *TE Links*, either as strict hops (*NodeEdgePoints*) or as loose hops (*Nodes*).

[3]Criteria for assigning/mapping ServiceEndPoints to NodeEdgePoints are out of scope of this FRD, but are typically part of implementation agreement (IAs) and some examples are provided by the use cases in the appendices.
[4] The TAPI Connection Route is described in terms of Cross-Connections rather than Link-Connections. Conceptually a Connection Route is concatenation of Link Connections (resources associated with a Link) and Cross-Connections (resources within the *Nodes* in the underlying *Topology).*

Virtual Network Service

The T-API *Virtual-Network-Service* (VNS) represents a request for creation and offering of a virtual network *Topology* that maps two or more *Service-End-Points*, by an API-provider to an API client in accordance with agreements reached between them (e.g., satisfying the users' objectives). As such, *Virtual-Network-Service* is a container for virtual network *Topology* request details and is distinct from the *Topology* that realizes the request.

1.6 Conventions

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

2 Functional Architecture

The 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. It aims to achieve these goals by providing programmable access to typical transport SDN Controller functions. 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. A shared *Context* models everything that exists in an API provider to support a given API client. 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.

Typically, the shared *Context* setup also includes association attributes to establish identity and security that permit secure client-provider communication sessions. A session is the mechanism[5] that supports information exchange between specific instances of an API client and an API provider within a shared *Context* that has been secured by appropriate authentication and security credentials and prevents unauthorized access. Similar to user login, the session normally begins with an exchange of identity and security credentials, followed by agreement on an initial state, much of which may be re-stored from prior sessions. During the session, the API client may invoke services on and modify the state of resources within the shared *Context*. Each information exchange should be attributable to a session, for example in an audit log. A session may continue indefinitely, or end with an explicit logout, a failure, or a timeout. Since the shared *Context* supports only one session (or vice-versa), the session identification and association with the shared *Context* is implicit.

Thus a shared *Context* determines the makeup of the network resource abstraction instances over which the API operates. For example, the API client could

[5] The actual implementation-specific mechanisms to maintain, exchange or enforce the session state information is out of scope of this document and could be either maintained by the stateful API provider/server or offered by the API clients (as in stateless RESTful communication architectures)

- 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
- Subscribe to notification of events within the shared *Context*

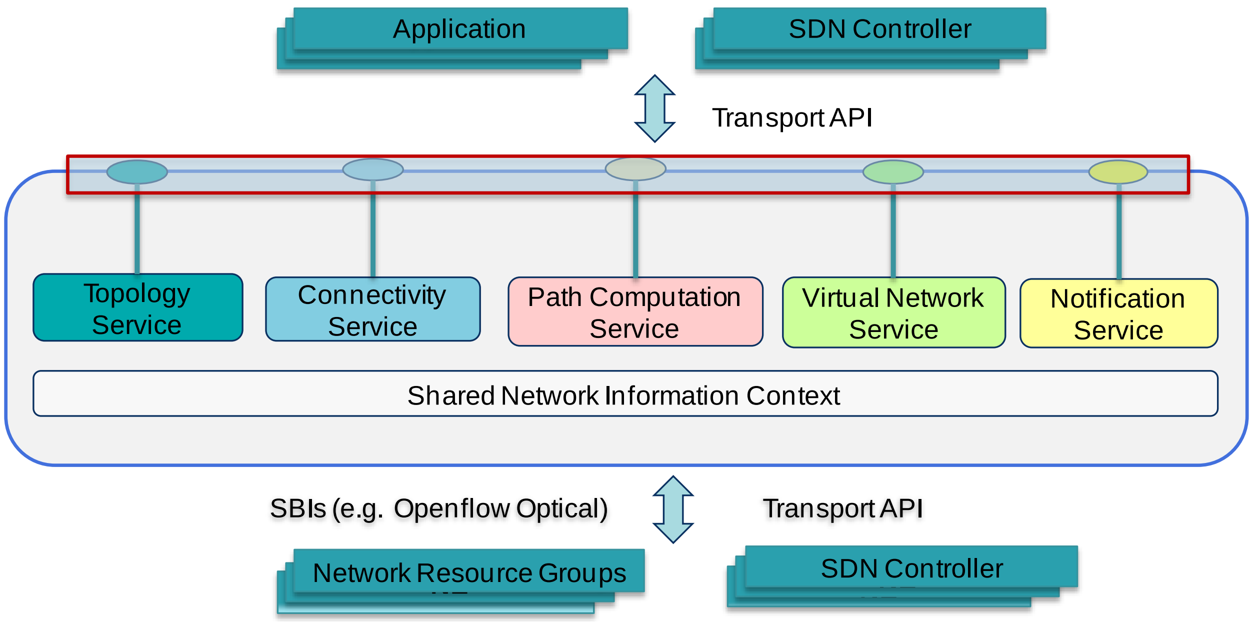


Figure 2: Transport API Functional Architecture

3 Functional Requirements

3.1 Topology Service

The Topology Service APIs allow an API client to retrieve topological information that is within its shared Context.

3.1.1 Topology Retrieval APIs

TAPI_FR 1: Get Topology List	
Description	• Returns list of top-level *Topology* instances directly scoped by the *Context* • This also includes details for each *Topology* including references to lower-level *Nodes* and *Links* encompassed by the *Topology* as allowed by policy
Pre-conditions	
Inputs	• Retrieve Scope Filter: *Layer-Protocol* List : Enumeration value - If set/non-empty, the API call will return references to only those *Topology* instances that support at least one of the specified layer protocols
Outputs	List of *Topology* entities and details for each including: • List of IDs, Names, User-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	

TAPI_FR 2: 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 - 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 encompassed *Nodes* and *Links* that support at least one of the specified layer protocols
Outputs	• List of IDs, Names, User-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	

TAPI_FR 3: 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* - When NULL is provided, this API call should return an error. • *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, User-Labels and Extensions (if any) • List of supported *Layer-Protocol* Names • Administrative, Operational and Lifecycle States • Transfer characteristics such as Cost, Timing, Integrity and Capacity • List of references to aggregated *NodeEdgePoints* indexed by Layer
Notifications	
Error-conditions	
Post-conditions	

TAPI_FR 4: 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* - When NULL is provided, this API call should return an error. • *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, User-Labels and Extensions (if any) • Administrative, Operational, and Lifecycle 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 adjacent 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	

TAPI_FR 5: 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* - When NULL is provided, this API call should return an error. • *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, User-Labels and Extensions (if any) • Administrative, Operational, and Lifecycle States • List of supported *Layer-Protocols* including attribute-details indexed by Layer
Notifications	
Error-conditions	
Post-conditions	

3.2 Connectivity Service

The Connectivity Service APIs allow an API client to retrieve connectivity information and request connectivity service within its shared Context.

3.2.1 Connectivity Retrieval APIs

TAPI_FR 6: 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, User-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	

TAPI_FR 7: 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, User-Labels and Extensions (if any) • Lifecycle 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	

TAPI_FR 8: 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, User-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	

TAPI_FR 9: 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, User-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	

TAPI_FR 10: 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, User-Labels and Extensions (if any) • Operational, and Lifecycle States • Connectivity Constraints including – Required Constraints such as Capacity – Optional 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	

TAPI_FR 11: 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, User-Labels and Extensions (if any) • Operational and Lifecycle 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	

3.2.2 Connectivity Request APIs

TAPI_FR 12: 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	• *ObjectCreation* notifications on *ConnectivityService*, associated/created *Connections* and *ConnectionEndPoints* • *AttributeValueChange* notifications on affected *ServiceEndPoints* • *StateChanges* on related *State* attributes in the affected objects
Error-conditions	Service not supported Service input not supported Endpoint not recognized
Post-conditions	

TAPI_FR 13: 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	• *AttributeValueChange* notifications on *ConnectivityService*, associated/affected *Connections*, *ConnectionEndPoints* and *ServiceEndPoints* • May also result in *ObjectCreation* and/or *ObjectDeletion* notifications on associated/affected *Connections* and/or *ConnectionEndPoints* • *StateChanges* on related *State* attributes in the affected objects
Error-conditions	Modification could not be supported Modification parameter not understood Modification not allowed
Post-conditions	ConnectivityService modified

TAPI_FR 14: Delete Connectivity Service	
Description	• Causes deletion of an existing *ConnectivityService* • Deletes all associated *Connections* that were owned/created by the *ConnectivityService*

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	• ID of the deleted *ConnectivityService*
Notifications	• *ObjectDeletion* notifications on *ConnectivityService*, associated/deleted *Connections* and *ConnectionEndPoints* • *AttributeValueChange* notifications on affected *ServiceEndPoints* • *StateChanges* on related *State* attributes in the affected objects
Error-conditions	
Post-conditions	ConnectivityService deleted

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 15: 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[6] *ServiceEndPoint* for the *ConnectivityService* – Role[7] 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 • Routing (Connectivity) Constraints including – Required Constraints such as Capacity – Optional Constraints such as Layer, Latency, Cost, etc. • Objective function
Outputs	List of paths computed containing following information (only "one" if shouldComputeConcurrentPaths is false) • Path identifier (identifier of the calculated route) • Routing constraints (Description of connectivity constraints that are met) • Path which is an ordered list of TE Links – described either as strict hops (NodeEdgePoints) or loose hops(Nodes)
Notifications	• *ObjectCreation* notifications on *computed*/created *Paths*
Error-conditions	Cause of failure
Post-conditions	

TAPI_FR 16: 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 path to be modified • Connection Id: (optional) Identifies resources in use for the Connection for the path being optimized • Routing (Connectivity) Constraints including – Required Constraints such as Capacity – Optional Constraints such as Layer, Latency, Cost, etc. • Objective Function • Optimization Constraint
Outputs	List of paths computed containing following information (only "one" if shouldComputeConcurrentPaths = false) • Path identifier (identifier of the calculated route) • Routing constrains (Description of connectivity constraints that are met) • Path which is an ordered list of (TE Links) – described either as strict hops (NodeEdgePoints) or loose hops(Nodes)
Notifications	• *AttributeValueChange* notifications on affected *Paths*

[6] The number of ServiceEndPoints is restricted to 2 for the Path Computation request
[7] The value for Role is constrained to only Symmetric for the Path Computation request

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	

3.4 Virtual Network Service

3.4.1 Virtual Network Retrieval APIs

TAPI_FR 17: Get Virtual Network Service List	
Description	• Returns list of *VirtualNetworkService* entities that represent the virtual network requests that were received • This also includes attribute details for each *VirtualNetworkService* including – References to *ServiceEndPoints* of the *Service* – Optionally, references to the virtual *Topology* realizing the *VirtualNetworkService*
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 *VirtualNetworkServices* that support at least one of the specified layer protocols • Include *Topology* : true or false
Outputs	List of *VirtualNetworkServices* indexed by *Layer* and details for each including: • List of IDs, Names, User-Labels and Extensions (if any) • Administrative, Operational, and Lifecycle States • Virtual Network Constraints including – Required Constraints such as Service Level and Traffic Matrix – Any optional Constraints • Optionally the *Topology* realizing the *VirtualNetworkService*
Notifications	
Error-conditions	
Post-conditions	

TAPI_FR 18: Get Virtual Network Service Details	
Description	• Returns attributes of the *VirtualNetworkService* entity identified by the provided inputs. • This includes references to *ServiceEndPoints* of the *VirtualNetworkService*. • This optionally includes references to the *Topology* realizing the *VirtualNetworkService*.
Pre-conditions	
Inputs	• *Service* ID or Name : String - When NULL is provided, this API call should return an error condition • Include *Topology* : true or false

Outputs	• List of IDs, Names, User-Labels and Extensions (if any) • Administrative, Operational, and Lifecycle States • Virtual Network Constraints including – Required Constraints such as Service Level and Traffic Matrix – Any optional Constraints • Optionally the *Topology* realizing the *VirtualNetworkService*
Notifications	
Error-conditions	
Post-conditions	

3.4.2 Virtual Network Request APIs

TAPI_FR 19: Create Virtual Network Service	
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	• List of following details for every *ServiceEndPoint* for the *Virtual Network Service* – Reference (Name/ID) to the *ServiceEndPoint* • Virtual Network Constraints including – Required Constraints such as Traffic Matrix – Any optional Constraints such as Service Level • Start Time & End Time
Outputs	• Virtual Network Service ID: The identifier of the Virtual Network Service instance that was created that includes identifier/reference of the virtual *Topology* that was created.
Notifications	• *ObjectCreation* notifications on *VirtualNetworkService*, and associated/created *Topology, Nodes, Links and NodeEdgePoints* • *AttributeValueChange* notifications on affected *ServiceEndPoints* • *StateChanges* on related *State* attributes in the affected objects
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.

TAPI_FR 20: Delete Virtual Network Service	
Description	For the client side of the API to delete the *VirtualNetworkService* and the associated *Topology* 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	• Virtual Network Service ID: The identifier of the virtual network to be deleted
Outputs	• Id *VirtualNetworkService* that was deleted
Notifications	• *ObjectDeletion* notifications on *VirtualNetworkService*, and associated/deleted *Topologies, Nodes, Links and NodeEdgePoints* • *AttributeValueChange* notifications on affected *ServiceEndPoints* • *StateChanges* on related *State* attributes in the affected objects
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.

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 (which are defined as) 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[8].

It will also be possible to retrieve notification records at a later time, by querying the notifications manager for the history records of the generated notifications. This result of the query depends on other configuration policies such as the record retention policies, which are currently out of the scope of this document.

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 21: Discover Supported Notification Types	
Description	Allows an API Client to discover the notifications capabilities supported by an API Provider
Pre-conditions	Knowledge of the Notification Server (e.g. URI, IP/Port, etc.)
Inputs	
Outputs	• Supported *Notification-Type* List : Enumeration value The notification types are specified in section 3.5.2. • Supported *Object-Type* List : Enumeration value The object types are specified in section 3.5.2.
Notifications	
Error-conditions	Reason for Failure
Post-conditions	Supported notification and object types discovered

[8] A session is the mechanism that supports information exchange between specific instances of an API client and an API provider within a shared Context that has been secured by appropriate authentication and security credentials.

TAPI_FR 22: Create Notification Subscription	
Description	Allows an API Client to subscribe to receive autonomous notifications from API provider, as per the specified filters
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. • Subscription Scope Filter: *Object-Id* List : List of globally unique object Ids (uuid). If set/non-empty, the system will push *Notifications* related to the specified Object instances only, irrespective of other filter attribute settings
Outputs	subscriptionId
Notifications	notificationId Object Creation Notification per the successful subscription.
Error-conditions	Reason for Failure
Post-conditions	Subscription created

TAPI_FR 23: Modify Notification Subscription	
Description	Allows an API Client to modify its subscription to receive autonomous notifications from API provider, as per the specified filters
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. • Subscription Scope Filter: *Object-Id* List : List of globally unique object Ids (uuid). If set/non-empty, the system will push *Notifications* related to the specified Object instances only, irrespective of other filter attribute settings
Outputs	subscriptionId
Notifications	notificationId Object Creation Notification per the successful subscription.
Error-conditions	Reason for Failure
Post-conditions	Subscription created

TAPI_FR 24: Delete Notification Subscription	
Description	Allows an API Client to delete its subscription to stop receiving autonomous notifications from API provider
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

TAPI_FR 25: Suspend Notification Subscription	
Description	Allows an API Client to modify its subscription to temporarily stop receiving autonomous notifications from API provider
Pre-conditions	Knowledge of available notification subscriptions The notification subscription is active and not suspended
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

TAPI_FR 26: Resume Notification Subscription	
Description	Allows an API Client to modify its subscription to resume receiving autonomous notifications from API provider
Pre-conditions	Knowledge of available notification subscriptions The notification subscription is suspended
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

TAPI_FR 27: Retrieve Notification Records	
Description	Allows an API Client to retrieve notification records by querying the API provider for records of the generated notifications. This result of the query depends on other configuration policies such as the record retention policies, which are currently out of the scope of this document.
Pre-conditions	Knowledge of available notifications types and their sources Knowledge of any Record retention polices that affect the availability of the queried *Notification* records.

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. • Subscription Scope Filter: *Object-Id* List : List of globally unique object Ids (uuid). If set/non-empty, the system will push *Notifications* related to the specified Object instances only, irrespective of other filter attribute settings • Start Time & End Time – Returns all *Notifications* whose *Event-Time* falls under this range. This operation ignores the existence of any record retention policies and assumes knowledge of those polices via external means.
Outputs	List of Notification records
Notifications	
Error-conditions	Reason for Failure
Post-conditions	Subscription created

3.5.2 Notification Message Types

TAPI_FR 28: Object Creation Notification	
Values	This message shall minimally support the following attributes • Notification Header (as per above) • Additional Information • Additional Text

TAPI_FR 29: Object Deletion Notification	
Values	This message shall minimally support the following attributes • Notification Header (as per above) • Additional Information • Additional Text

TAPI_FR 30: Attribute Value Change Notification	
Values	This message shall minimally support the following attributes • Notification Header (as per above) • Attribute Value Change List each consisting of o { attributeName, oldAttributeValue, newAttributeValue } • Additional Information • Additional Text

TAPI_FR 31: State Change Notification	
Values	This message shall minimally support the following attributes • Notification Header (as per above) • State Change List each consisting of ○ { attributeName, oldStateValue, newStateValue } • Additional Information • Additional Text

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

TAPI_FR 33: 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 34: Capacity (Profile)	
Values	The following are the required Bandwidth Profile parameters • Committed Information Rate (CIR): The rate of the packets that the transport networks commit to forward with some negotiated QoS objectives (packet loss, packet delay, packet delay variation). CIR-conformant packets are also referred to as green packets. The following are the optional Bandwidth Profile parameters • Bandwidth Profile Type: Indicates the type of algorithm (e.g., MEF 10.1, RFC 2697, RFC 2698, RCF 4115, etc.). If not specified, it is selected based on the controller's policy. • Committed Burst Size (CBS): The maximum burst-size at which frames can be received at the line speed while still being CIR-conformant. If not specified, it is selected based on the controller's policy. • Excessive Information Rate (EIR): The rate of packet that the transport network allows to be forwarded but without guaranteed any QoS objectives. EIR-conformant packets are also referred to as yellow packets. The default value is 0. • Excessive Burst Size (EBS): The maximum burst-size at which frames can be received at the line speed while still being EIR-conformant. If not specified, it is selected based on the controller's policy. • Color Mode (CM): Indicates whether the packets are marked with the color information (color-aware) or not (color-blind). The default value is color-blind. • Coupling Flag (CF): Indicates the mode of operations for the MEF 10.1 Bandwidth Profile Type. When set, the rate of yellow frames is bounded by the Peak Information Rate (PIR = CIR + EIR); otherwise the rate of yellow frames is bounded by the EIR. If not specified, it is selected based on the controller's policy.

TAPI_FR 35: Administration State	
Values	• LOCKED • UNLOCKED

TAPI_FR 36: Operational State	
Values	• ENABLED • DISABLED

TAPI_FR 37: Lifecycle State	
Values	• PLANNED • POTENTIAL • INSTALLED • IN_CONFLICT • PENDING_REMOVAL

TAPI_FR 38: Port Role	
Values	Denotes the role of the *End-Point* with respect to the *Forwarding-Construct* • Symmetric • Root • Leaf

TAPI_FR 39: Port Direction	
Values	Denotes the directionality of the signal-flow in the *Port* with respect to the *Forwarding-Construct* • Input • Output • Bidirectional

TAPI_FR 40: Termination Direction	
Values	Denotes the directionality of the signal-flow in the Service*EndPoint* or *ConnectionEndPoint* • Sink • Source • Bidirectional

TAPI_FR 41: 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 FRD and is typically part of the implementation agreements (IAs)

TAPI_FR 42: Service Type	
Values	The Service Type attribute shall minimally support following values • POINT_TO_POINT • POINT_TO_MULTIPOINT • MULTIPOINT

TAPI_FR 43: 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 Descriptor– a abstract label 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 44: Virtual Network Service Constraints	
Values	The following are the required Virtual Network Service parameters • Traffic Matrix: A matrix to describe the traffic (e.g., bandwidth) between each pair of ServiceEndPoints The following are the optional Virtual Network constraint parameters • VN Service Level Descriptor – a abstract label the meaning of which is mutually agreed – typically represents metrics such as – type of topology abstraction, class of service, priority, resiliency, availability

TAPI_FR 45: Traffic Matrix	
Values	The *Traffic Matrix* consists of a list of elements, each describing possible individual traffic flow comprising the matrix, including: • *ServiceEndPoint* details for the *Traffic flow* • Reference (Name/ID) to the Source *ServiceEndPoint* • Reference (Name/ID) to the Sink *ServiceEndPoint* • Optionally the Layer of the *ServiceEndPoint* if it supports multiple layers • Traffic Constraints including • Required Constraints such as Capacity • Optional Constraints such as Layer, Latency, Cost, etc.

TAPI_FR 46: Path Optimization Constraint	
Values	The following are the optimization constraint parameters to be provided as input to the Path Optimization API • Whether traffic interruption allowed or not

TAPI_FR 47: Path Objective Function	
Values	The following are the Objective Function parameters to be provided as input to the Path Computation Service API • Allow to compute concurrent Paths or not • Minimize the cost • Resource sharing (max re-usage/min re-usage) Whether new resource can be used or no • Minimum/maximum link utilization value • 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 48: 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) – Globally unique ID of the object on which this notification is being raised • Object Instance Name List • Event Time Stamp • Notification Source Indicator

TAPI_FR 49: 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 50: 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 • Context • Topology • Node • Link • Node-Edge-Point • Service-End-Point • Connection-End-Point • Connectivity-Service • Virtual Network Service • Connection • Path

TAPI_FR 51: Notification-Source-Indicator	
Description	Notification agents shall identify the source of notification messages.
Values	This enumeration attribute shall minimally support following values • ResourceOperation • ManagementOperation • Unknown

4 Appendix A: Transport API Concepts Overview

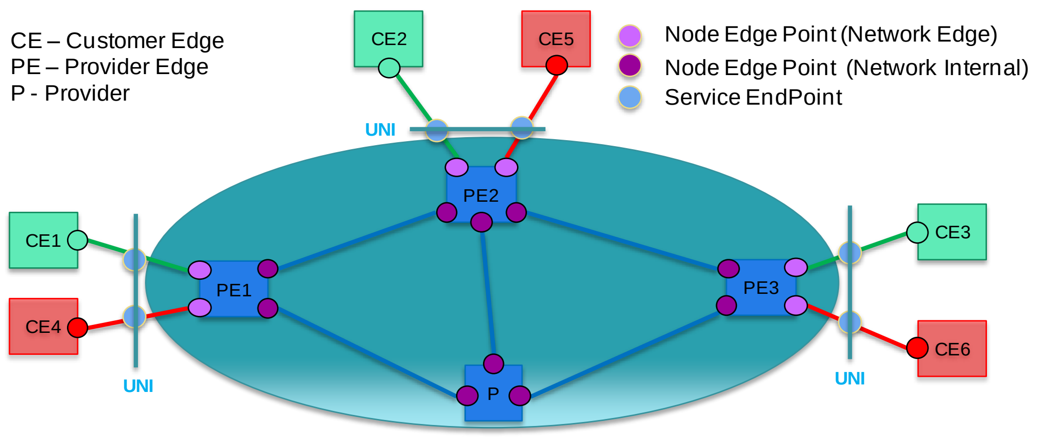
4.1 Context

An SDN controller or manager typically organizes its information and operates on that information within specific contexts. A *Context* is an abstraction that allows for logical isolation and grouping of network resource abstractions for specific purposes/applications and/or information exchange with its users/clients over an interface. Thus, *Context* defines the scope of control and naming that a particular SDN controller, manager or a client application has with respect to the information it operates on internally or exchanges over an interface. It therefore determines the makeup of the network resource abstractions within that domain of control.

More specifically, a T-API Context:

- Is defined by a set of Service-End-Points and its capabilities (capacity, layers, etc.)
- Utilizes one or more shared namespace in information exchanges over the interface
- Includes *one or more top-level Topology* abstractions that are:
    - Either statically assigned by a controller or dynamically created on client request
    - Defined by a set of *Nodes* and *Links*
- Provides scope for control (create/retrieve/update/delete) of Topology abstractions
- Determines the level of abstraction exposed over an interface

To further illustrate the concept of Context, consider the simple single domain physical network example shown in Figure-3[9]. It depicts a Network Provider (*Blue*) with two Customers (*Red* and *Green*). In this example, the network provider controller has 3 *Contexts* – its own internal/admin *Context* representing all the resources under its control (Control Domain), and one *Context* per customer (*Red* and *Green*) that it shares over its interfaces. This is shown in Figure-4 & Figure-5 below. In this example, the provider exposes a single-*Node Topology* abstraction to customer *Red*, while it exposes a multi-*Node Topology* abstraction to customer *Green*.



- A Network Provider (Blue) with two Customers (Red and Green)
- All UNI interfaces are ETH (e.g. 10GE), I-NNI interfaces are OTU (e.g. 100G OTN)
- All PE-NE are ODU/ETH capable, while P-NE is only ODU capable.

Figure 3 : Simple Physical Network Example

[9] The capabilities of the PE-NE in the figure-2 above are further described in detail in Appendix C

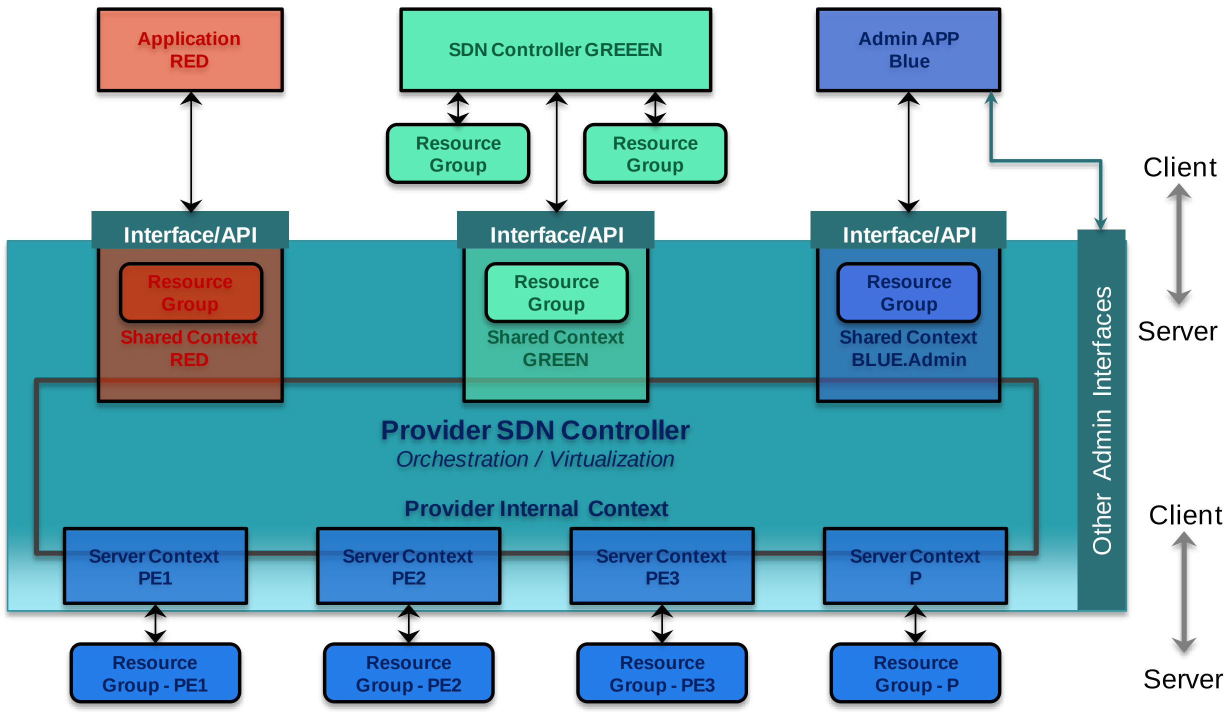


Figure 4: Shared Contexts - Architecture perspective

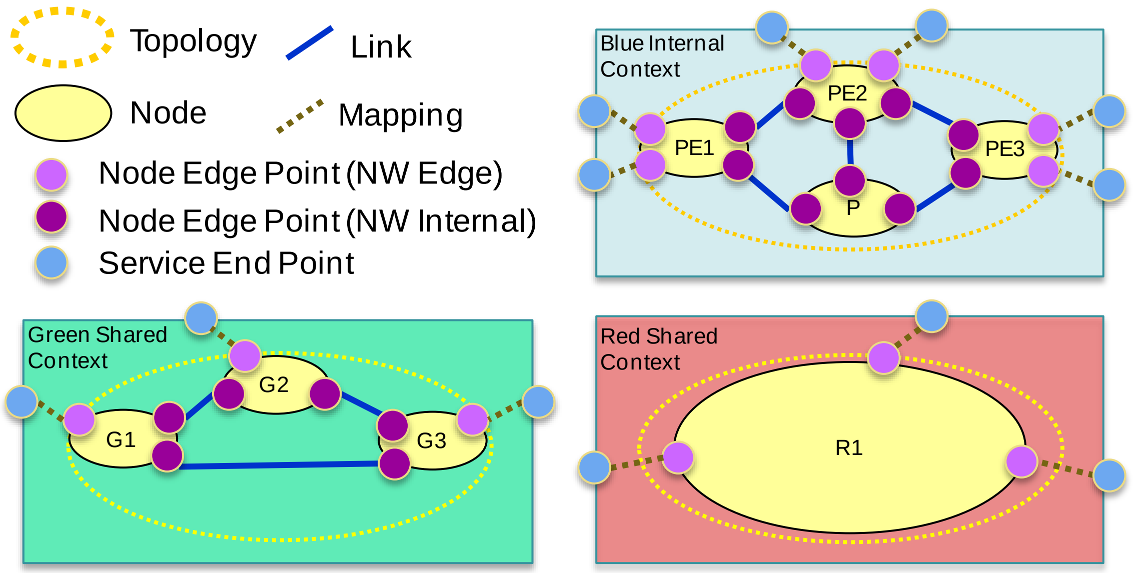


Figure 5: Shared Contexts & Topology

4.2 Node and Topology Aspects of Forwarding Domain

The *Forwarding-Domain* described in the ONF Core IM, represents the opportunity to enable forwarding between its edge-points. 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).

For the purposes of API requirements, the *Forwarding-Domain* has been refactored as two 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)

In this representation, a *Node* is a logical abstraction of forwarding capability, and as such could encompass an internal *Topology*. In such a case, a *Node* can be recursively decomposed into its lower-level *Nodes* and *Links*. So a Node at a top-level could abstract the *Topology* of an entire network while a *Node* at the bottom-most level could abstract a switch matrix within a device/NE.

To illustrate the concept, consider the physical network example presented in Figure-3 above. The *Node-R1* in the Red shared Context can be decomposed into a *Topology* of 3 *Nodes R1.1*, *R1.2*, *R1.3* and the *Links* between them, as shown in the Figure-6 below.

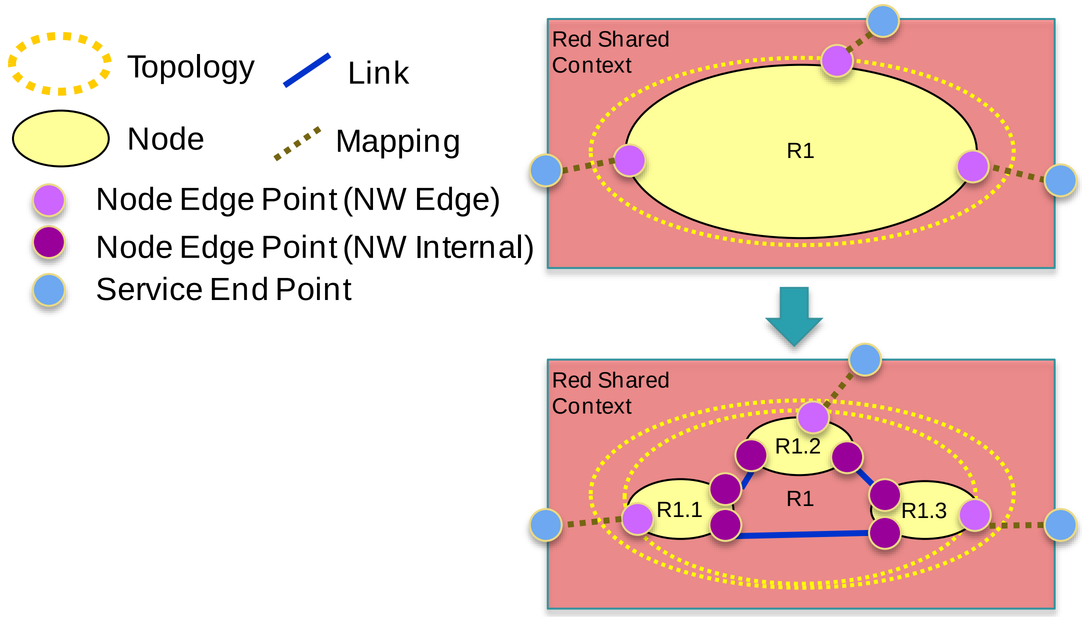


Figure 6: Topological Decomposition of Node

Figure-7 illustrates another example of hierarchical topology recursion from a top level topology abstraction (node B) to successively more detailed levels of topology abstraction until the lowest level of interest is reached (in this example, C, A1.1-A1.3, A2.1-A2.3).

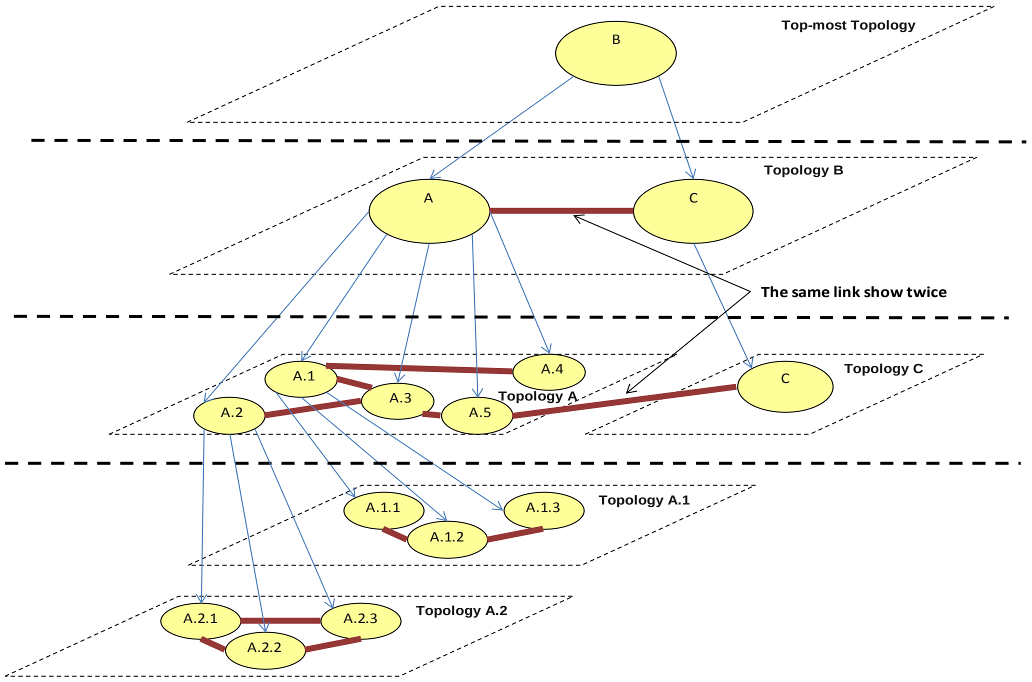


Figure 7: Recursive Topological Decomposition of Node

And Figure-8 below illustrates the same network example, from the perspectives of an imaginary observer, viewing the *ForwardingDomain* from different locations within the abstraction.

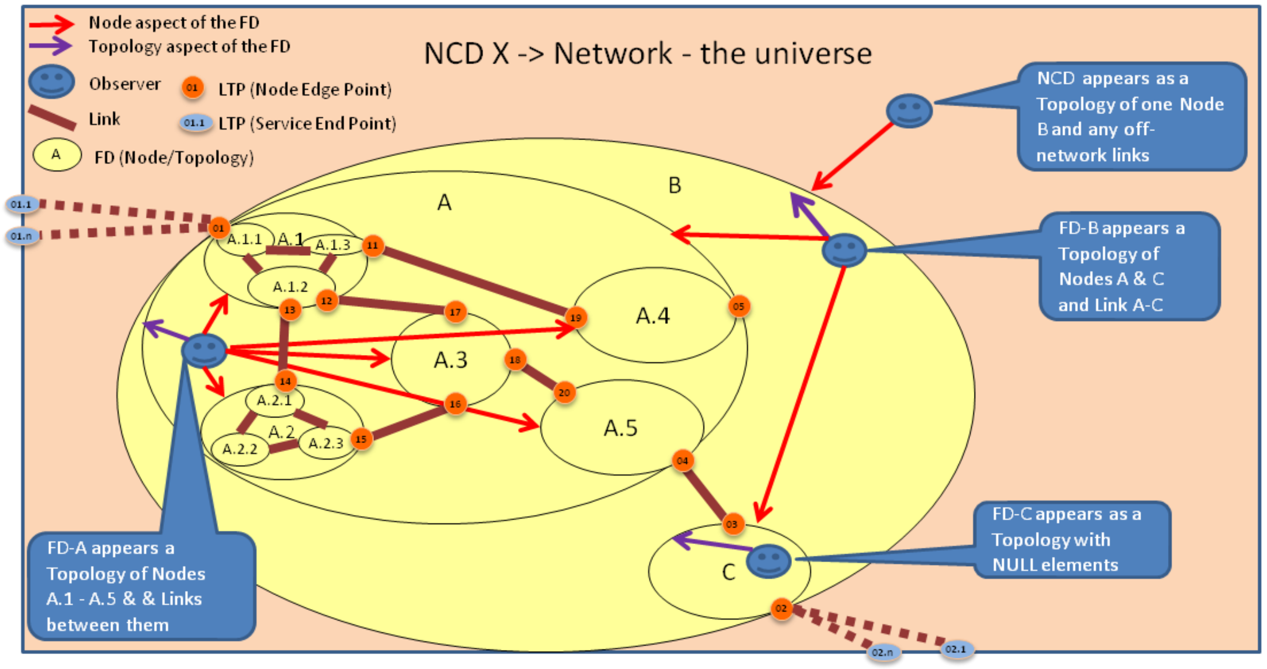

Node aspect of the FD
Topology aspect of the FD
Observer
01 LTP (Node Edge Point)
Link
01.1 LTP (Service End Point)
A FD (Node/Topology)
NCD X -> Network - the universe
NCD appears as a Topology of one Node B and any off-network links
FD-B appears a Topology of Nodes A & C and Link A-C
FD-C appears as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

Figure 8: Node/Topology perspectives of recursively partitioned Forwarding Domain (FD)

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 (*Node-)Edge-Point* client layers on one *Forwarding-Domain* with an equivalent set of (*Node-)Edge-Point* 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 *Link* can support multiple transport layers via the associated (*Node-)Edge-Point* instances on which it terminates.

4.3 Hierarchical Control Domains and Contexts

The T-API *Context* is a realization of the *Network-Control-Domain* as defined in the ONF Core Information Model

In interfaces where an abstracted view of network is offered, e.g. in client/server SDN controller relationship, the *Context* 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 abstracted view of the partitioned provider resources allocated to that particular client. In such cases, the *Context* also scopes the namespace for identifying objects representing the (virtual) resources within the (virtual) network view. The following figures illustrate few examples of *Contexts* in a hierarchical SDN controller system:

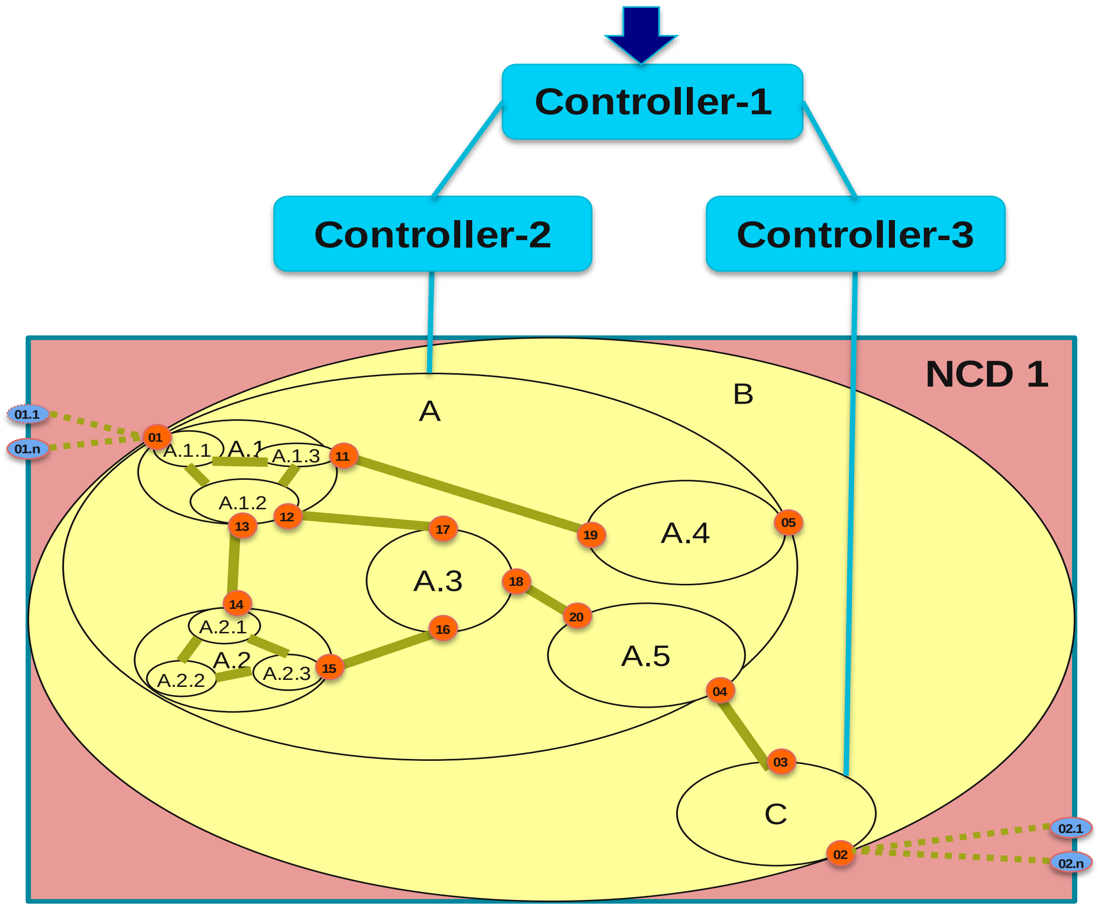


Figure 9: View of Controller-1 Context based on Views exported by Controllers 2 & 3

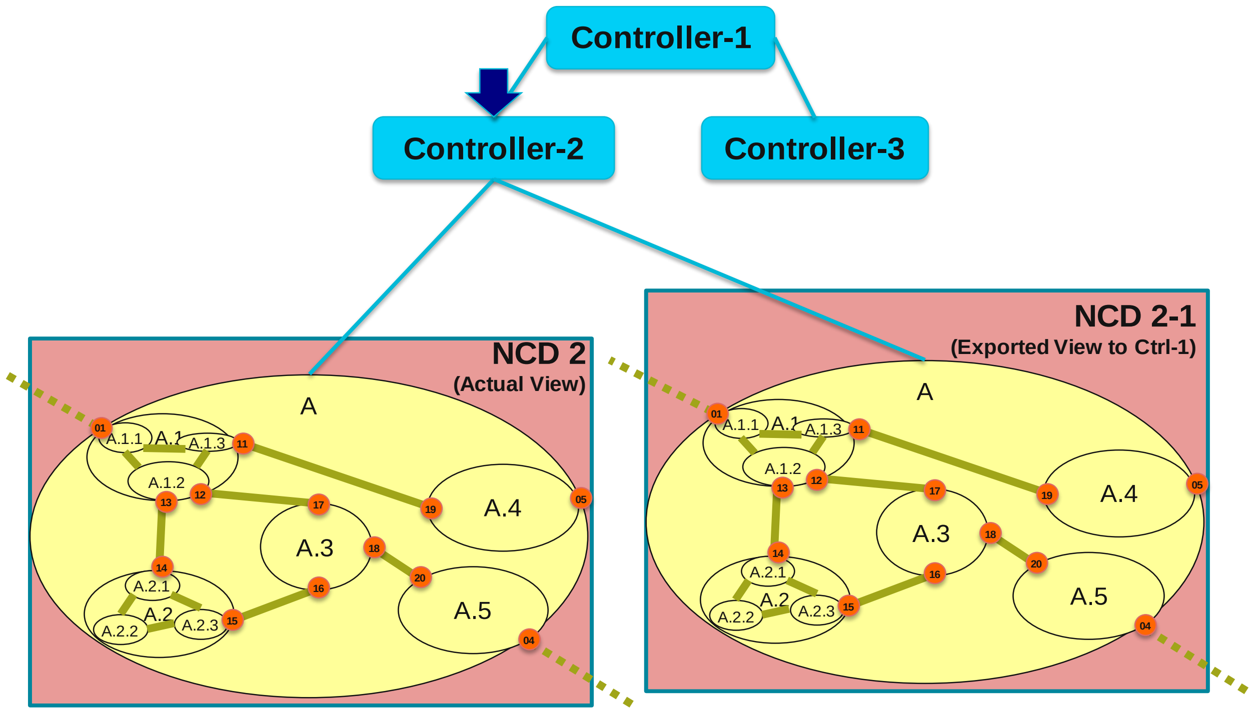


Figure 10: Views of Controller-2 Contexts

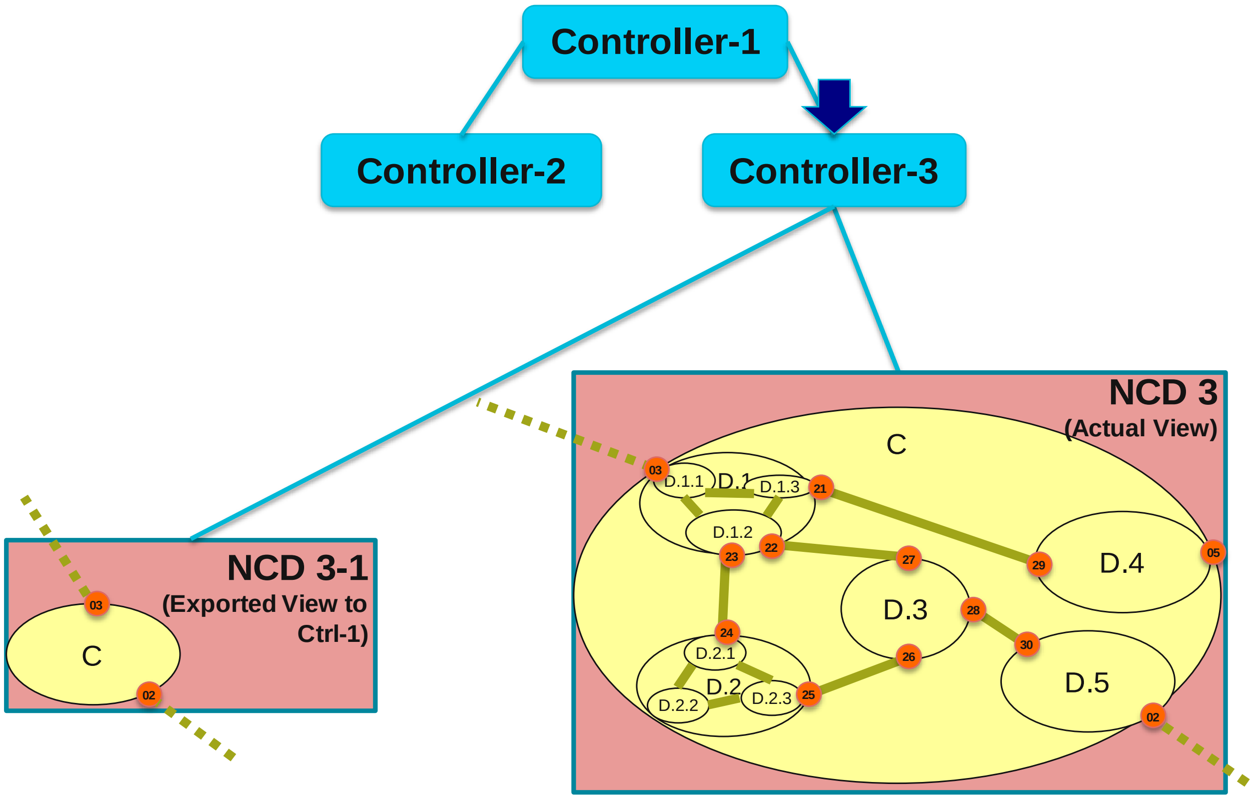


Figure 11: Views of Controller-3 Contexts

4.4 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.

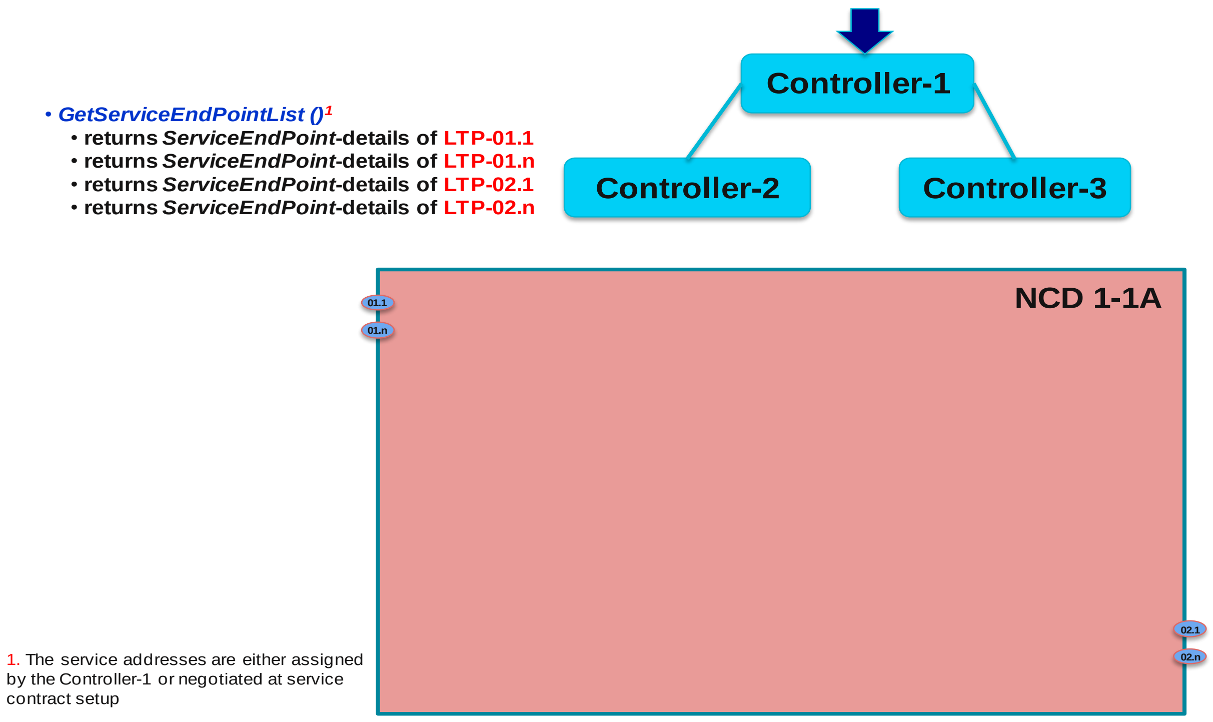

• GetServiceEndPointList ()1
• returns ServiceEndPoint-details of LTP-01.1
• returns ServiceEndPoint-details of LTP-01.n
• returns ServiceEndPoint-details of LTP-02.1
• returns ServiceEndPoint-details of LTP-02.n
Controller-1
Controller-2
Controller-3
NCD 1-1A
01.1
01.n
02.1
02.n
1. The service addresses are either assigned by the Controller-1 or negotiated at service contract setup

Figure 12: API Client's View of Controller-1 Context without retrieving Topology details

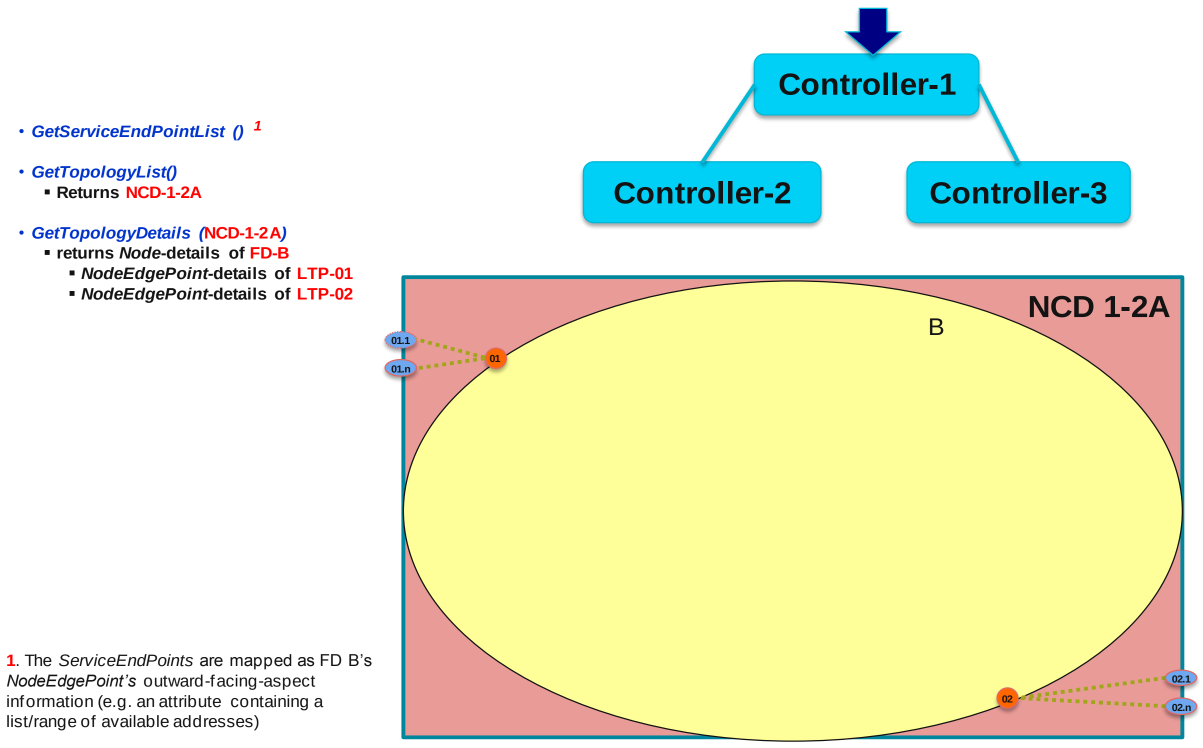

• GetServiceEndPointList () 1
• GetTopologyList()
▪ Returns NCD-1-2A
• GetTopologyDetails (NCD-1-2A)
▪ returns Node-details of FD-B
▪ NodeEdgePoint-details of LTP-01
▪ NodeEdgePoint-details of LTP-02
Controller-1
Controller-2
Controller-3
NCD 1-2A
B
01.1
01.n
01
02
02.1
02.n
1. The ServiceEndPoints are mapped as FD B's NodeEdgePoint's outward-facing-aspect information (e.g. an attribute containing a list/range of available addresses)

Figure 13: API Client's View of Controller-1 Context by retrieving top-most level of Topology

- *GetServiceEndPointList ()*
- *GetTopologyList()*
  - Returns NCD-1-2A
- *GetTopologyDetails (*NCD 1-2A*)*
  - returns *Node*-details of FD-B
    - *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

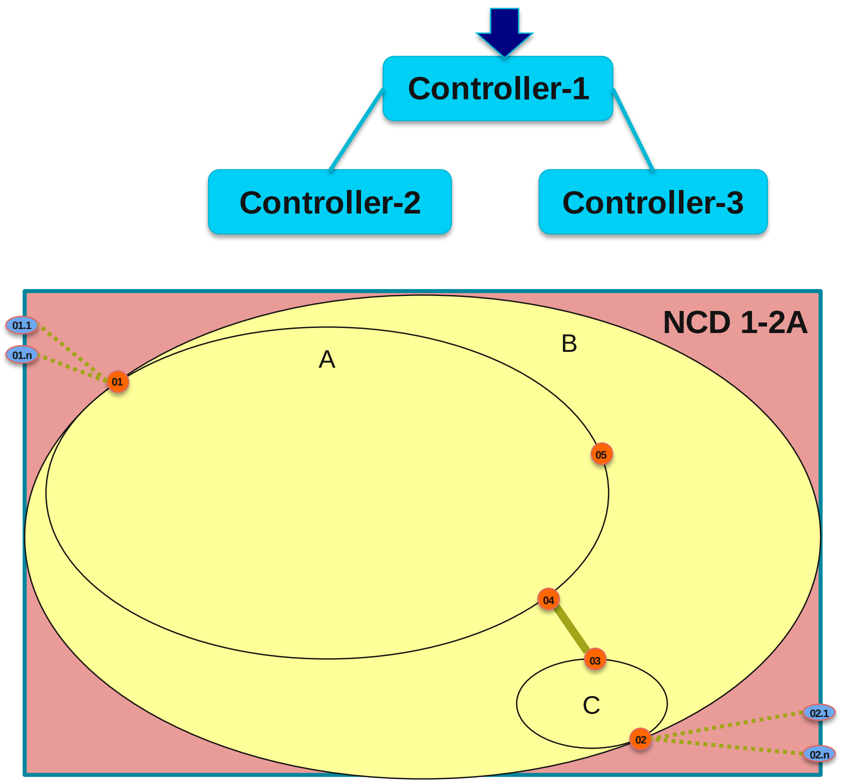


Figure 14: API Client's View of Controller1 Context by retrieving 2 levels of Topology details

- *GetServiceEndPointList ()*
- *GetTopologyList()*
  - Returns NCD-1-2A
- *GetTopologyDetails (*NCD-1-2A*)*
  - 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
- *GetTopologyDetails(*FD-A*)*
  - returns *Node*-details of FD-A.1 and
    - *NodeEdgePoint*-details of LTP-01
    - *NodeEdgePoint*-details of LTP-11
    - *NodeEdgePoint*-details of LTP-12
    - *NodeEdgePoint*-details of LTP-13
  - returns Node-details of FD-A.2 and
    - *NodeEdgePoint*-details of LTP-14
    - *NodeEdgePoint*-details of LTP-15
  - returns *Node*-details of ……
    - *NodeEdgePoint*-details of ……
  - returns *Link*-details of Link-A.1-A.2
  - returns *Link*-details of Link-A.2-A.3
  - returns *Link*-details of ……
- *GetTopologyDetails (*FD-C*)*
  - returns *Empty* List

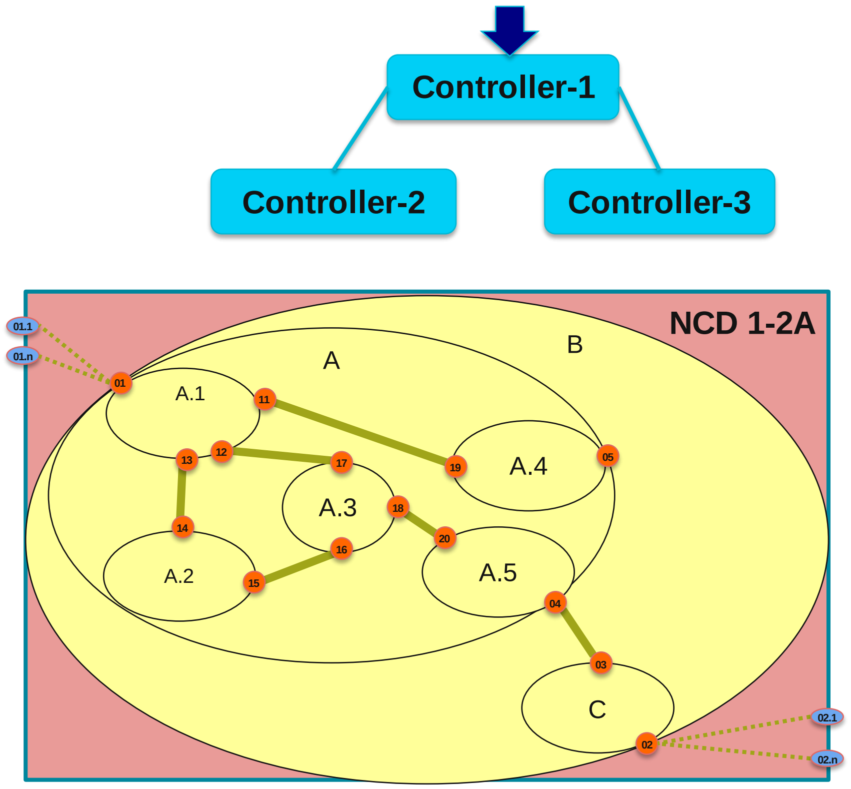


Figure 15: API Client's View of Controller-1 Context by retrieving 3 levels of Topology details

4.5 Service, Connection and Route

A *Connectivity-Service* represents an "intent-like" request for connectivity between two or more *Service-End-Points* (a realization of *LTP* described in the ONF Core IM) exposed by the *Context*. As such, *Connectivity-Service* is distinct from the *Connection* that realizes the *Connectivity-Service*. The requestor of the *Connectivity-Service* is expected to be able to express their intent using just an "external" *Node* view of *Forwarding-Domain* and the advertised *Service-End-Points* and not require knowledge of the "internal" *Topology* details of the *Forwarding-Domain*.

The association of the *Connectivity-Service* to *Service-End-Points* is made via the *Ports* of the *Connectivity-Service*.

The *Connectivity-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 *Node-Edge-Points* (another realization of *LTP* described in the ONF Core IM*)* from the *Node* aspect of the *Forwarding-Domain*. A *Connection* is typically described utilizing the "internal" *Topology* view of *Forwarding-Domain*.

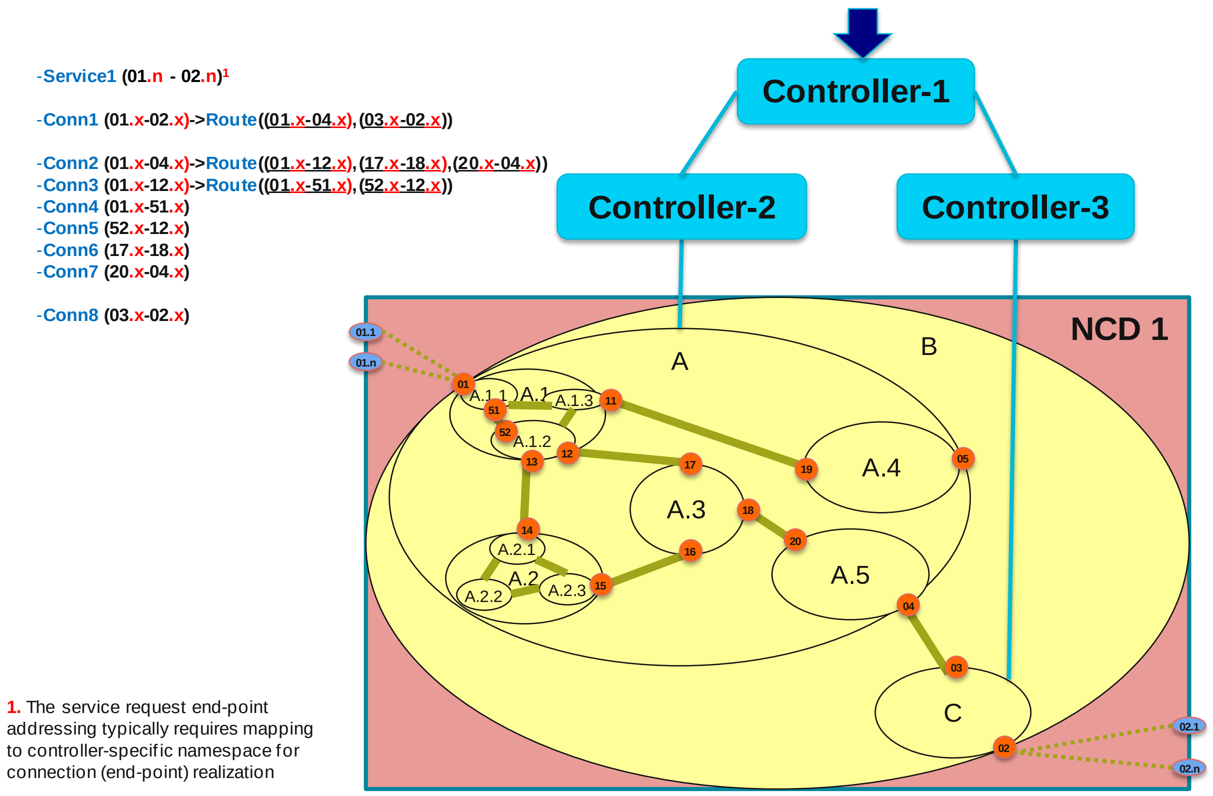
The *Connection* is modeled by the *Forwarding-Construct* entity defined in the ONF Core Information Model.

The association of the *Connection* to *Connection-End-Points* (yet another realization of *LTP* described in the ONF Core IM) is made via the *Ports* of the *Connection*, where each *Port* of the *Connection* has a role in the context of the *Connection*. The traffic forwarding between the associated *Connection-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* 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 *Routes*, 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.

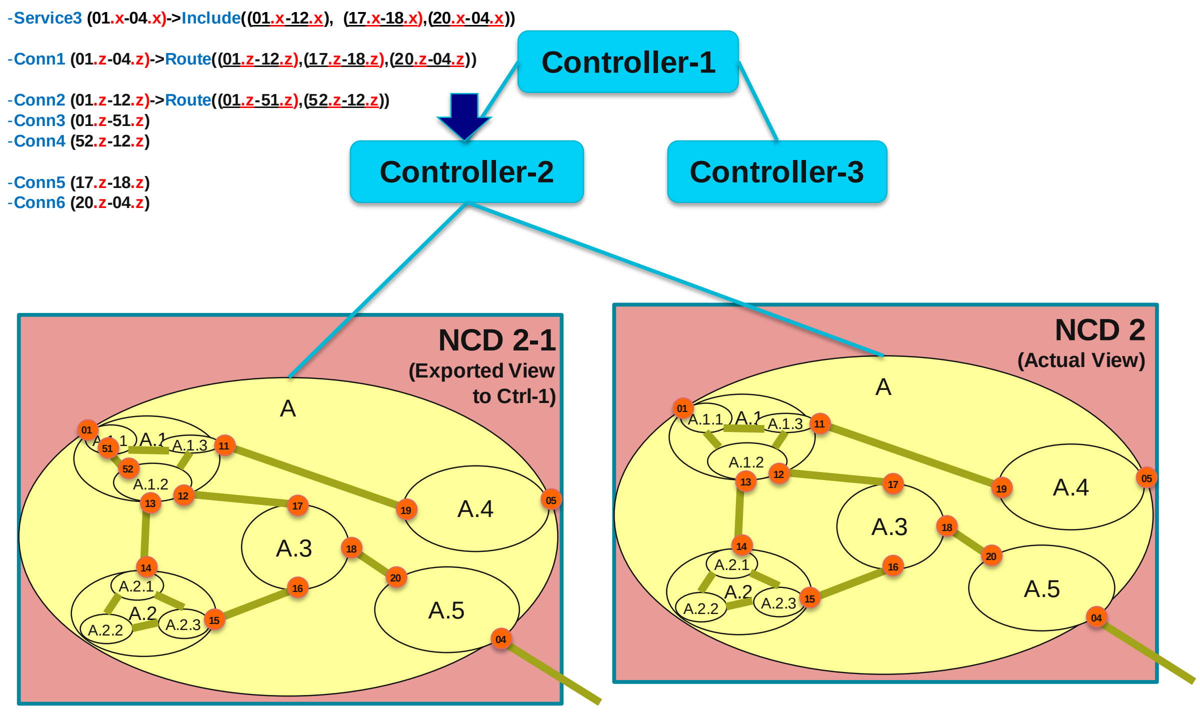
The *Route* represents the individual routes of a *Connection*. It is represented by a list of *Connections* at a lower level. Note that depending on the characteristics of the *Connectivity-Service* supported by a *Connection*, the *Connection* can have multiple *Routes*.



```
-Service1 (01.n - 02.n)1

-Conn1 (01.x-02.x)->Route((01.x-04.x),(03.x-02.x))

-Conn2 (01.x-04.x)->Route((01.x-12.x),(17.x-18.x),(20.x-04.x))
-Conn3 (01.x-12.x)->Route((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

Figure 16: Service & Connections from Controller-1 perspective



```
-Service3 (01.x-04.x)->Include((01.x-12.x), (17.x-18.x),(20.x-04.x))

-Conn1 (01.z-04.z)->Route((01.z-12.z),(17.z-18.z),(20.z-04.z))

-Conn2 (01.z-12.z)->Route((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)
```

Figure 17: Service & Connections from Controller-2 perspective

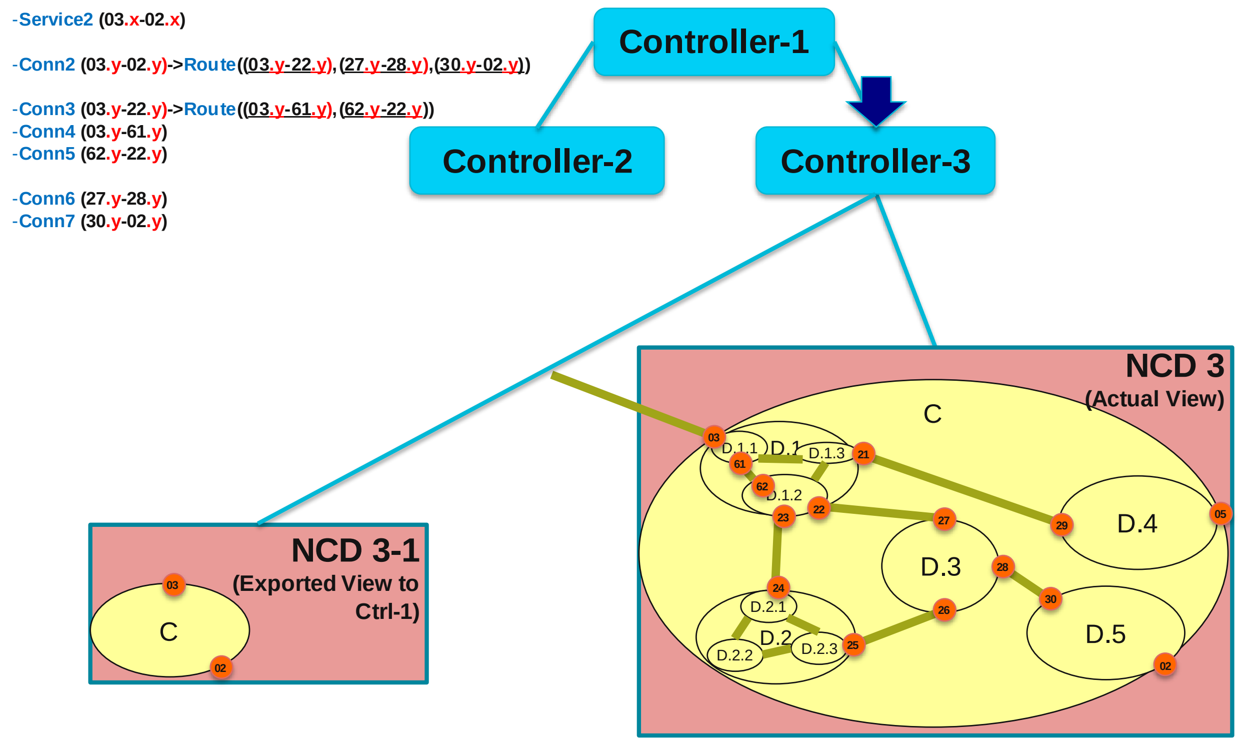


Figure 18: Service & Connections from Controller-3 perspective

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

The *Logical-Termination-Point* (*LTP*) described in the ONF Core IM, 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.

Three forms of *LTPs* are realized in T-API model:

- Node-Edge-Point - The *Node-Edge-Point* represents the inward network-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 a specific *Link*.

- Service-End-Point - The *Service-End-Point* represents the outward customer-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. shared addressing, capacity, resource availability, etc.), that enable the clients to request connectivity 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*.[10] The *Service-End-Points* have a specific *role* and *directionality* with respect to a specific *Connectivity-Service.*
- Connection-End-Point - The *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 a specific *Connection.*

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

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).

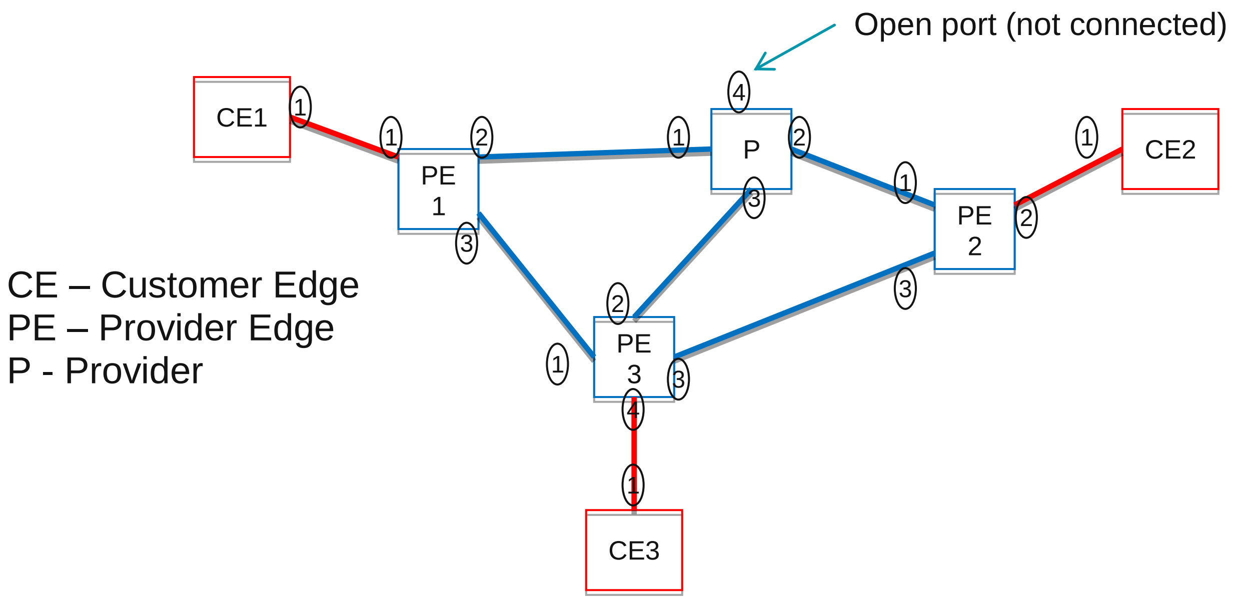


Figure 19: Example Physical Network Topology

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 this use case, it is assumed that the customer is requesting the service provider to forward all the Ethernet frames in the same manner, so only one Priority/CoS is implicitly defined for the EPL service.

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. Configuring a bandwidth constraint for this service is optional since, by definition, is shall be the same fixed bandwidth used on both the UNI Links associated with SEPs X and Y.

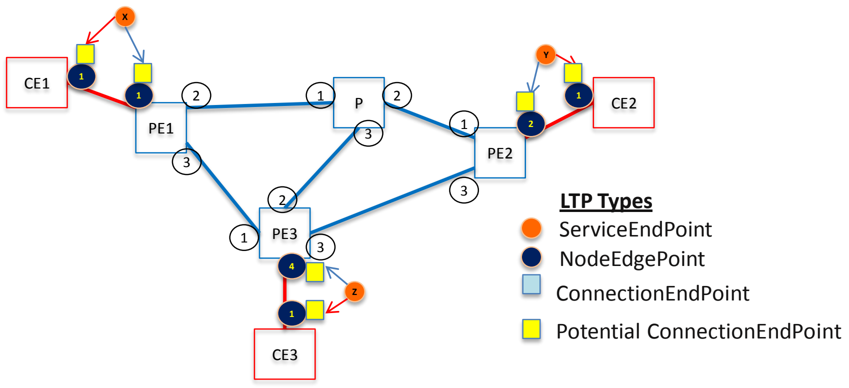


Figure 20: Example 10GE EPL Service over ODU2

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.

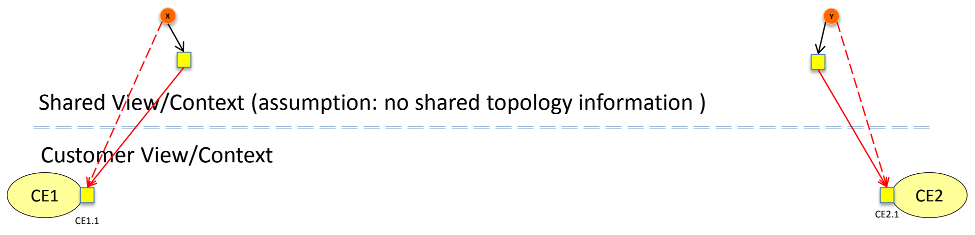

X
Y
Shared View/Context (assumption: no shared topology information)
Customer View/Context
CE1
CE1.1
CE2
CE2.1

Figure 21: 10G EPL - 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:

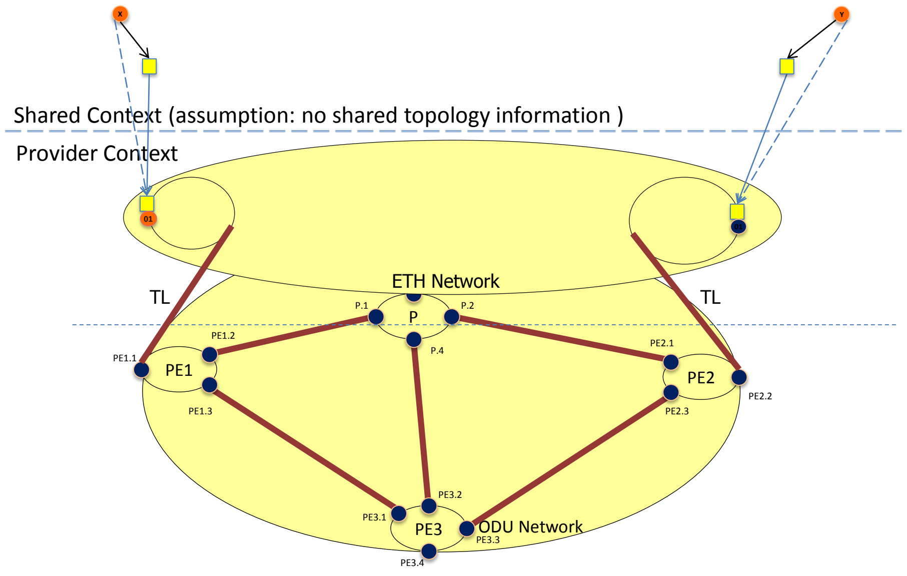


Figure 22: 10G EPL - Provider's View of Topology exported to the Customer

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:

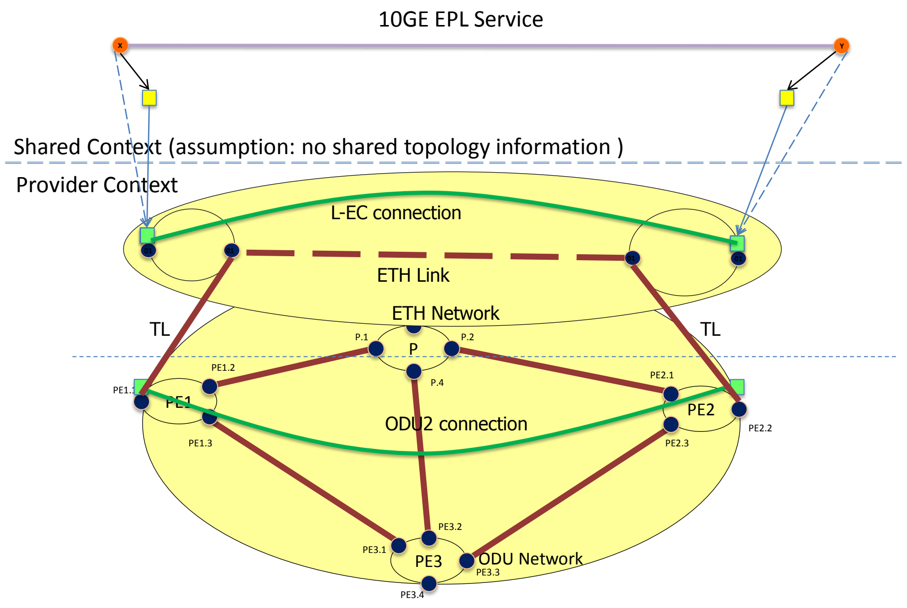


Figure 23: 10G EPL - Provider's View of Service/Connections exported to the Customer

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:

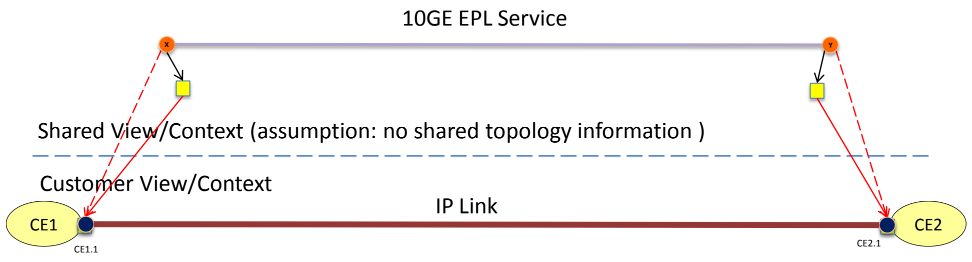

10GE EPL Service
X
Y
Shared View/Context (assumption: no shared topology information)
Customer View/Context
IP Link
CE1
CE1.1
CE2
CE2.1

Figure 24: 10G EPL - Customer's view of Service/Connectivity

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).

Also in this use case, it is assumed that the customer is requesting the service provider to forward all the Ethernet frames in the same manner, so only one Priority/CoS is implicitly defined for the EVPL service.

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. In this case the bandwidth profile for the EVPL service needs to be configured: it is assumed that the CIR is 1 Gb/s while the EIR is zero. The configuration of the CBS parameter is optional: if not specified, it can be chosen by the operator.

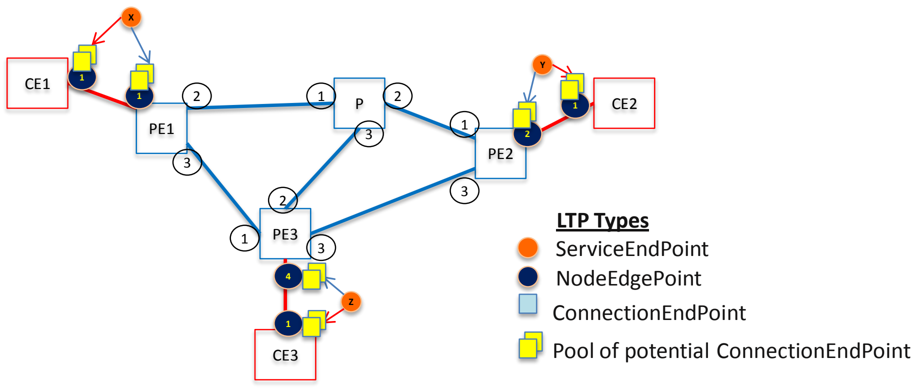


Figure 25: Example 1G EVPL Service over ODU0

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.

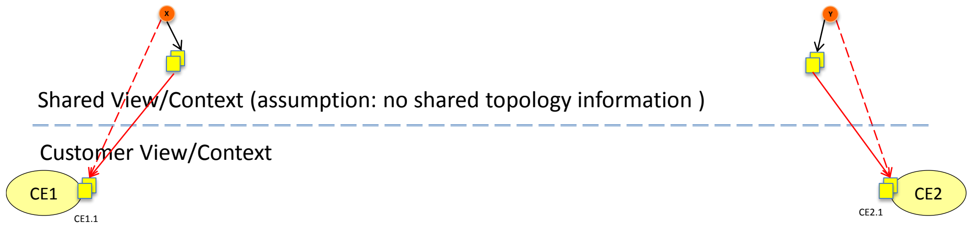

X
Y
Shared View/Context (assumption: no shared topology information)
Customer View/Context
CE1
CE1.1
CE2
CE2.1

Figure 26: 1G EVPL - 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:

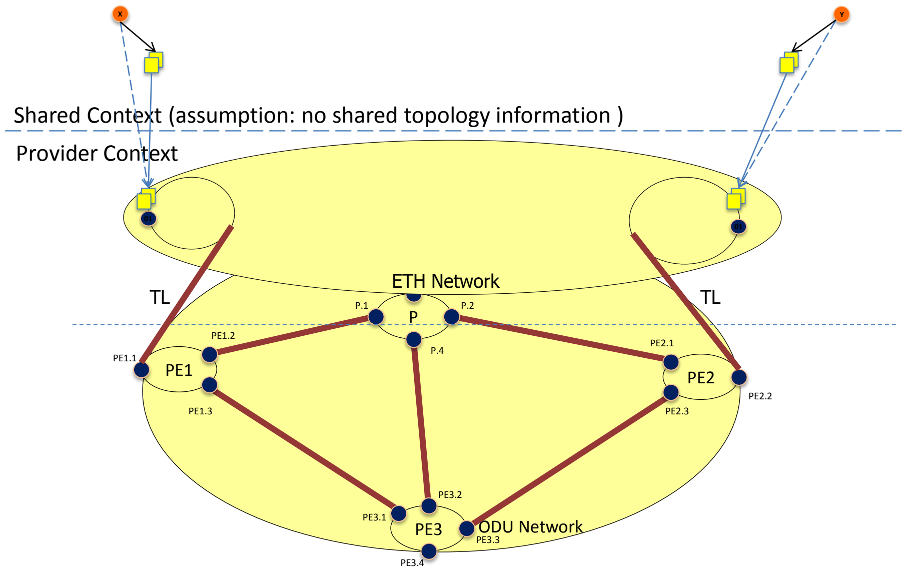


Figure 27: 1G EVPL - Provider's View of Topology exported to the Customer

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:

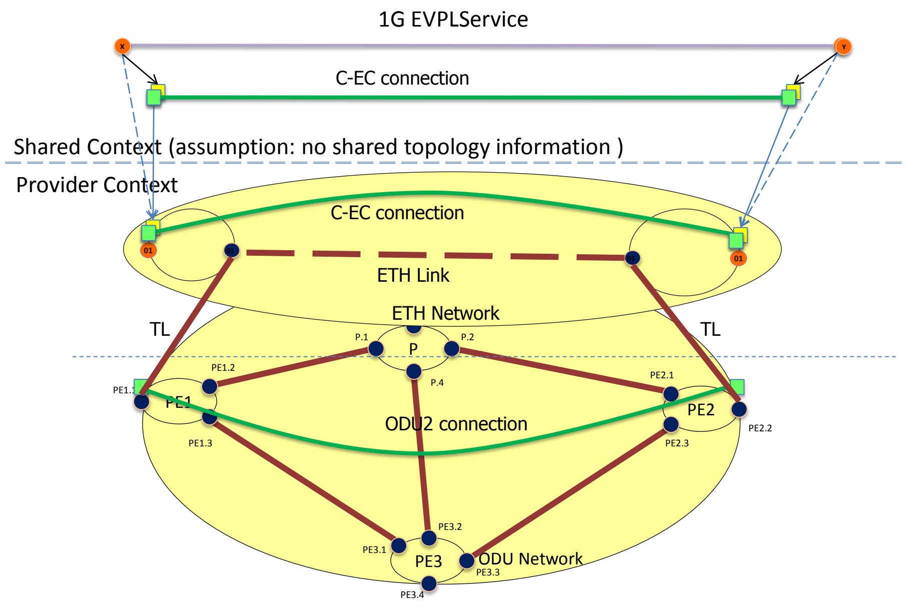


Figure 28: 1G EVPL - Provider's View of Service/Connections exported to the Customer

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. Alternately, if the Customer wants to force (for whatever reason) a specific C-VLAN value to be used inside the pool (top-down choice), the intended C-VLAN can be specified as part of the constraints related to the setup of the C-EC connection.

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.

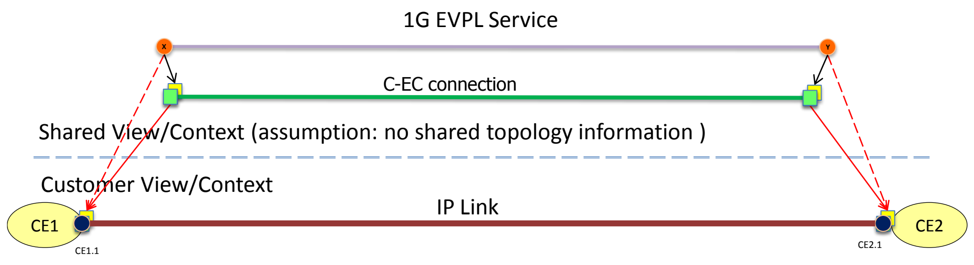


Figure 29: 1G EVPL - Customer's view of Service/Connectivity

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:

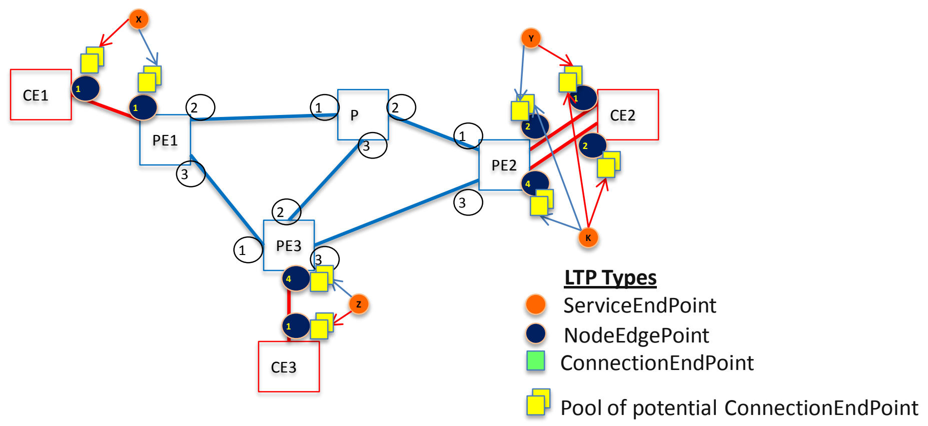


Figure 30: EVPL Service with Load Balancing

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:

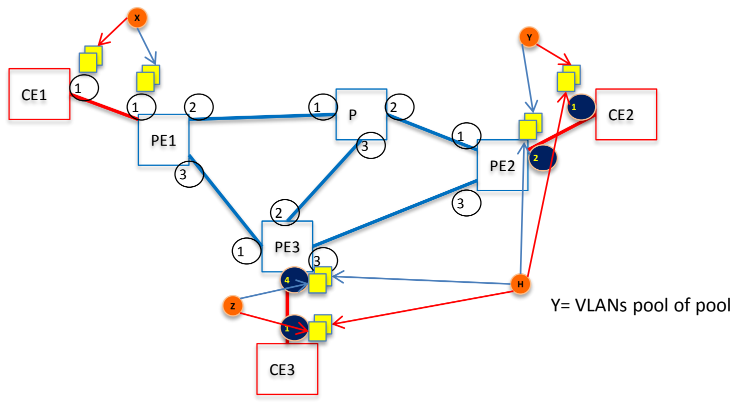


Figure 31: Example Anycast EVPL Service

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.

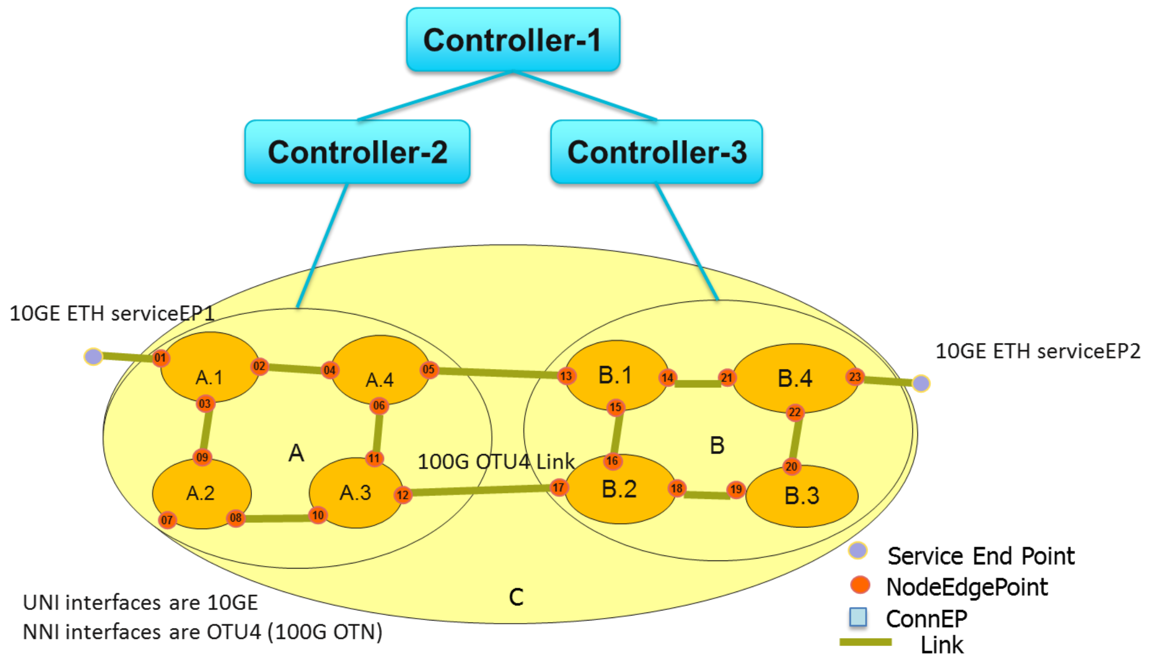


Figure 32: Multi-layer and Multi-domain Example Network Configuration

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.

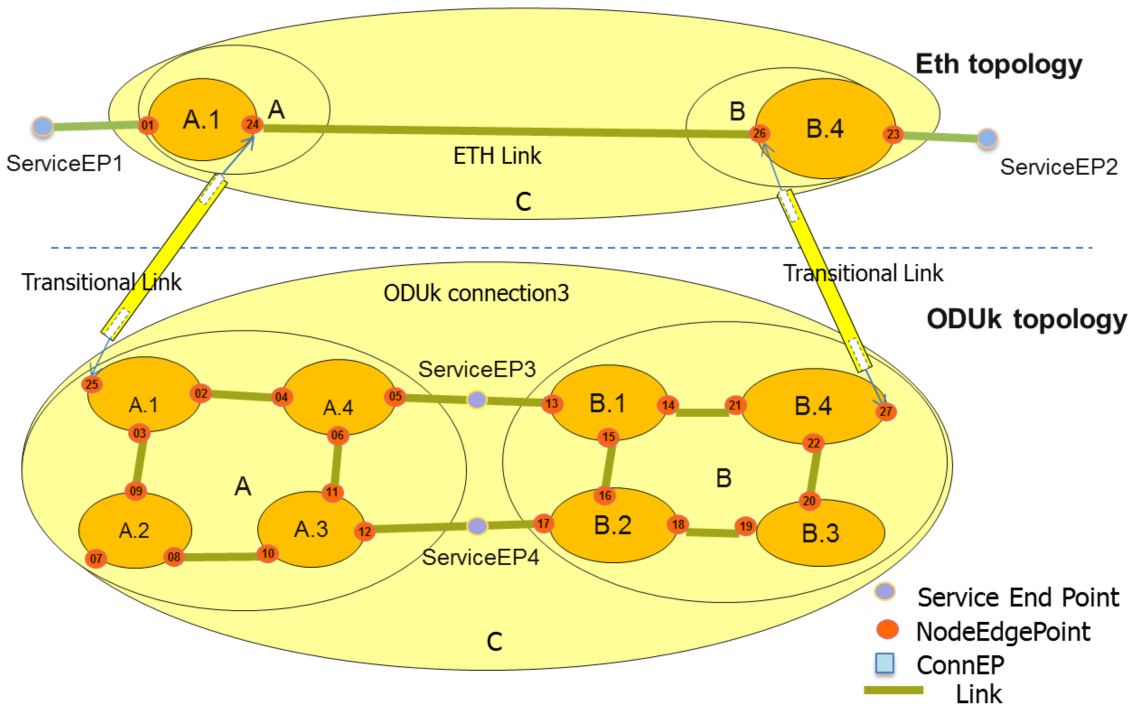


Figure 33: Network Topology in Controller 1

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 connection 1, 2 and ETH link internally.
15. Controller 1 returns ETH service to User.

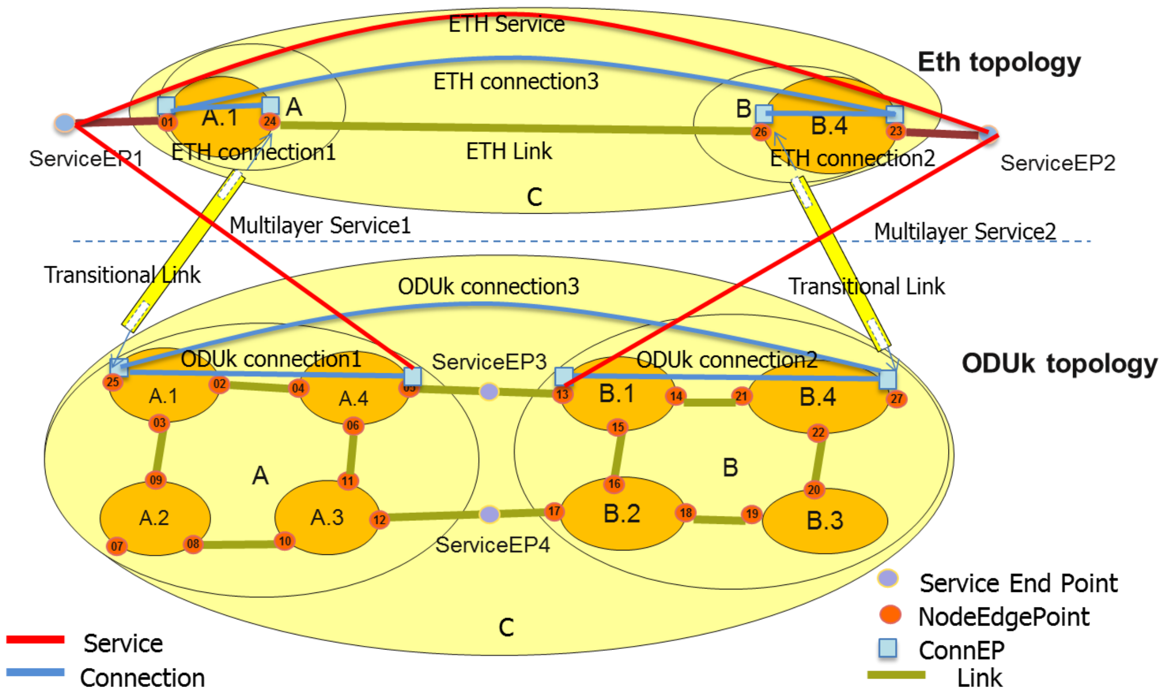


Figure 34: Multi-layer and Multi-domain service/connection setup (Option A)

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 connection 1, 2 and ETH link internally.
13. Controller 1 returns ETH service to User.

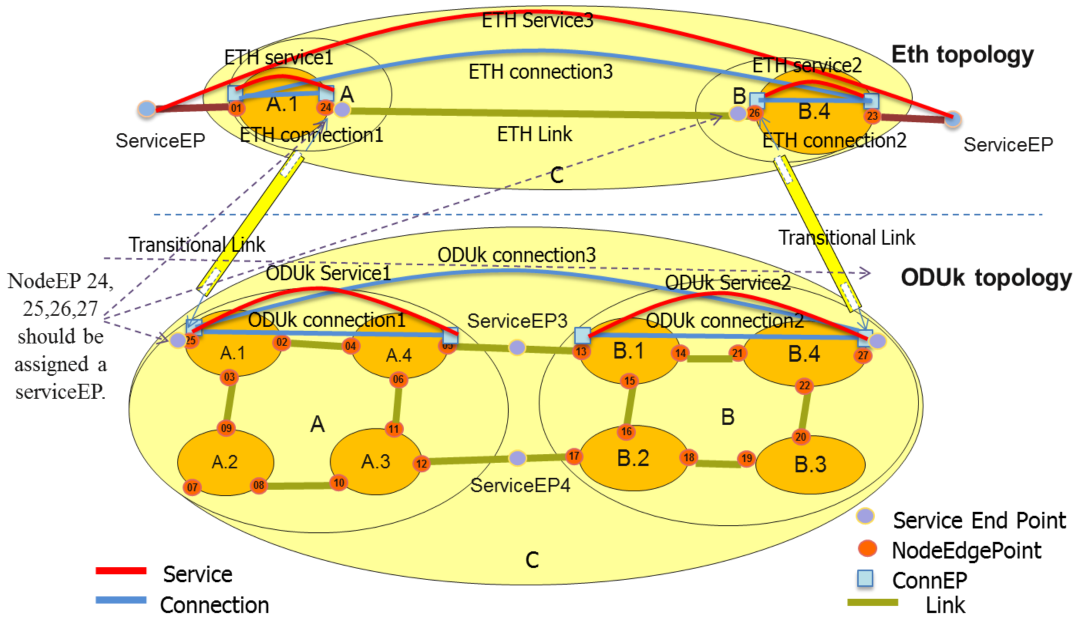


Figure 35: Multi-layer and Multi-domain service/connection setup (Option B)

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.

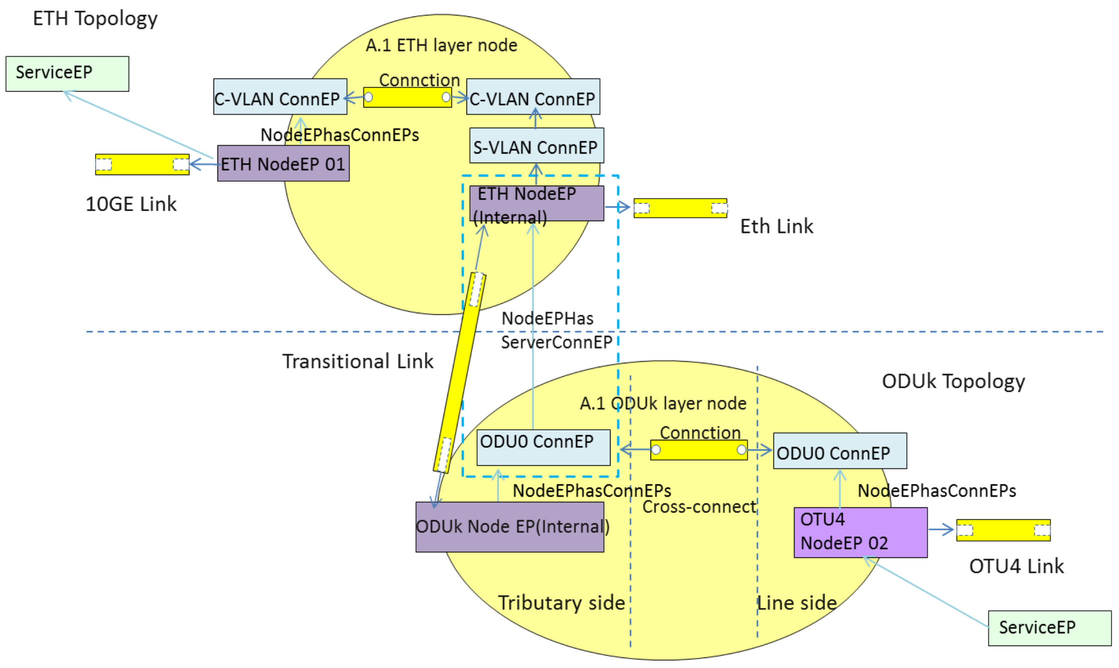


Figure 36: Topology instance diagram after service/connection setup

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

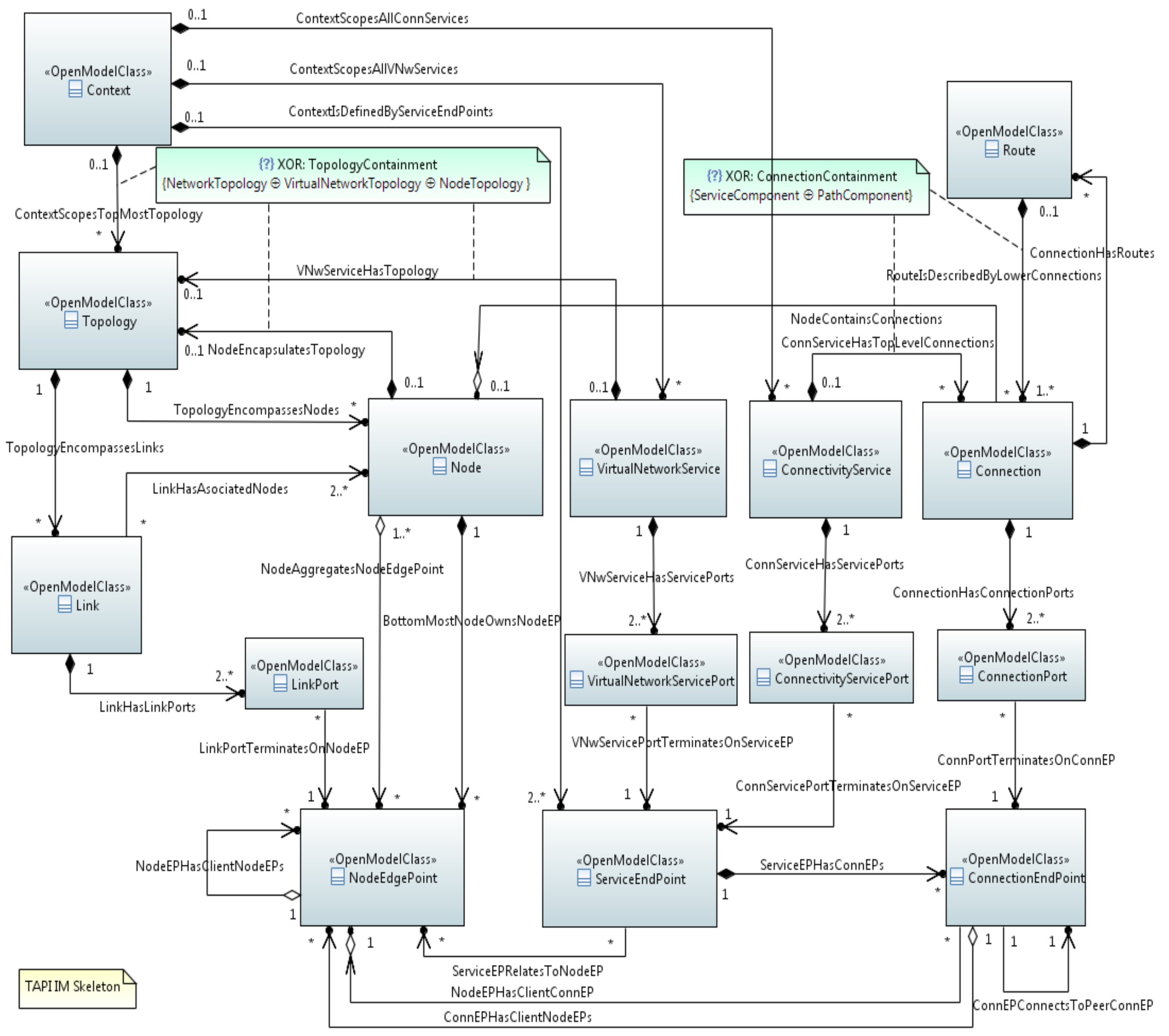

«OpenModelClass» Context
«OpenModelClass» Topology
«OpenModelClass» Node
«OpenModelClass» Link
«OpenModelClass» LinkPort
«OpenModelClass» NodeEdgePoint
«OpenModelClass» VirtualNetworkService
«OpenModelClass» ConnectivityService
«OpenModelClass» Connection
«OpenModelClass» Route
«OpenModelClass» VirtualNetworkServicePort
«OpenModelClass» ConnectivityServicePort
«OpenModelClass» ConnectionPort
«OpenModelClass» ServiceEndPoint
«OpenModelClass» ConnectionEndPoint
{?} XOR: TopologyContainment {NetworkTopology ⊕ VirtualNetworkTopology ⊕ NodeTopology}
{?} XOR: ConnectionContainment {ServiceComponent ⊕ PathComponent}
ContextScopesAllConnServices
ContextScopesAllVNwServices
ContextIsDefinedByServiceEndPoints
ContextScopesTopMostTopology
VNwServiceHasTopology
NodeEncapsulatesTopology
TopologyEncompassesNodes
TopologyEncompassesLinks
LinkHasAsociatedNodes
NodeAggregatesNodeEdgePoint
BottomMostNodeOwnsNodeEP
LinkHasLinkPorts
LinkPortTerminatesOnNodeEP
NodeEPHasClientNodeEPs
VNwServiceHasServicePorts
ConnServiceHasServicePorts
ConnectionHasConnectionPorts
VNwServicePortTerminatesOnServiceEP
ConnServicePortTerminatesOnServiceEP
ConnPortTerminatesOnConnEP
ServiceEPHasConnEPs
ServiceEPRelatesToNodeEP
NodeEPHasClientConnEP
ConnEPHasClientNodeEPs
ConnEPConnectsToPeerConnEP
NodeContainsConnections
ConnServiceHasTopLevelConnections
RouteIsDescribedByLowerConnections
ConnectionHasRoutes
TAPI IM Skeleton

Figure 37: Transport API Information Model Skeleton

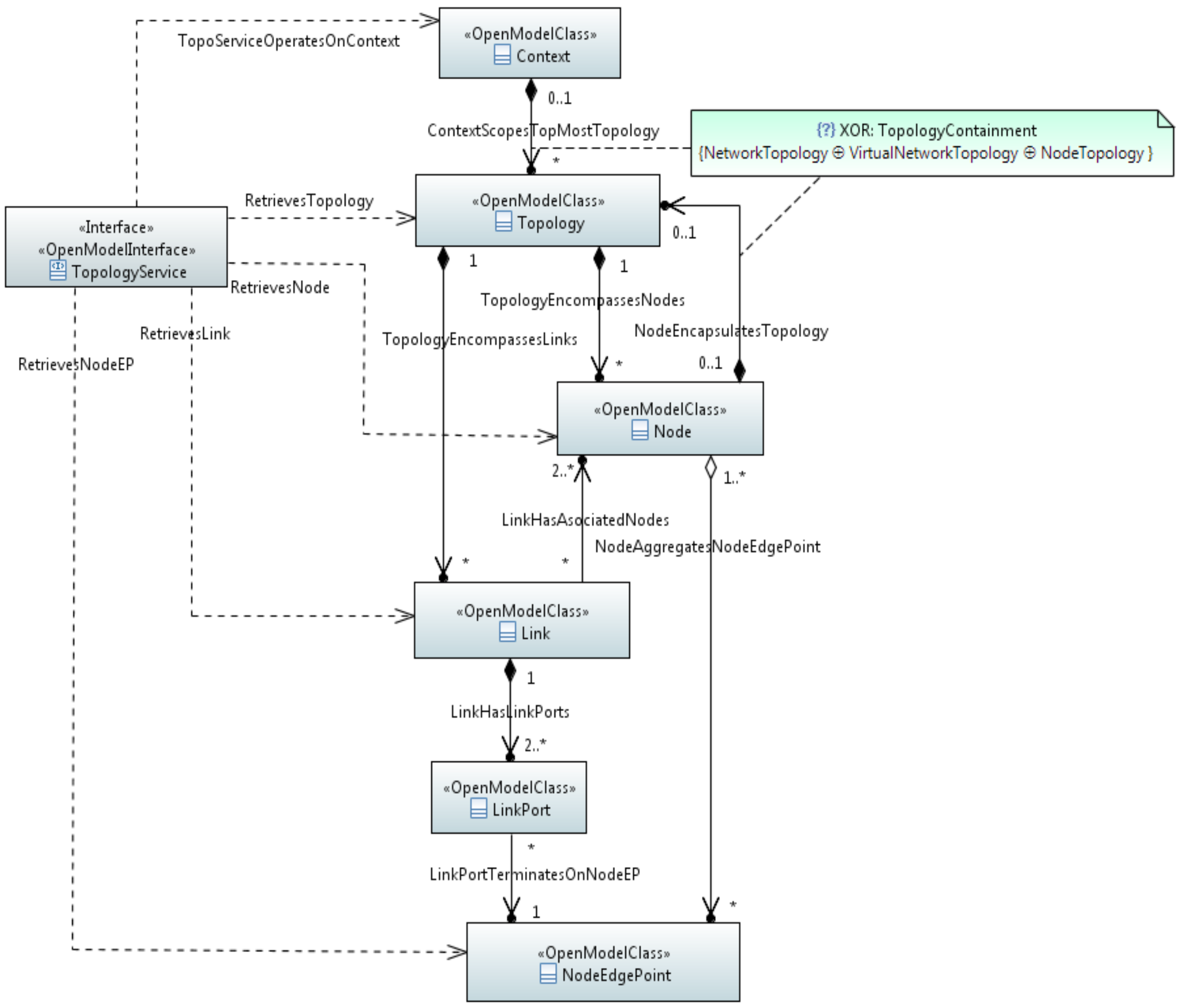


Figure 38: Topology Service Skeleton

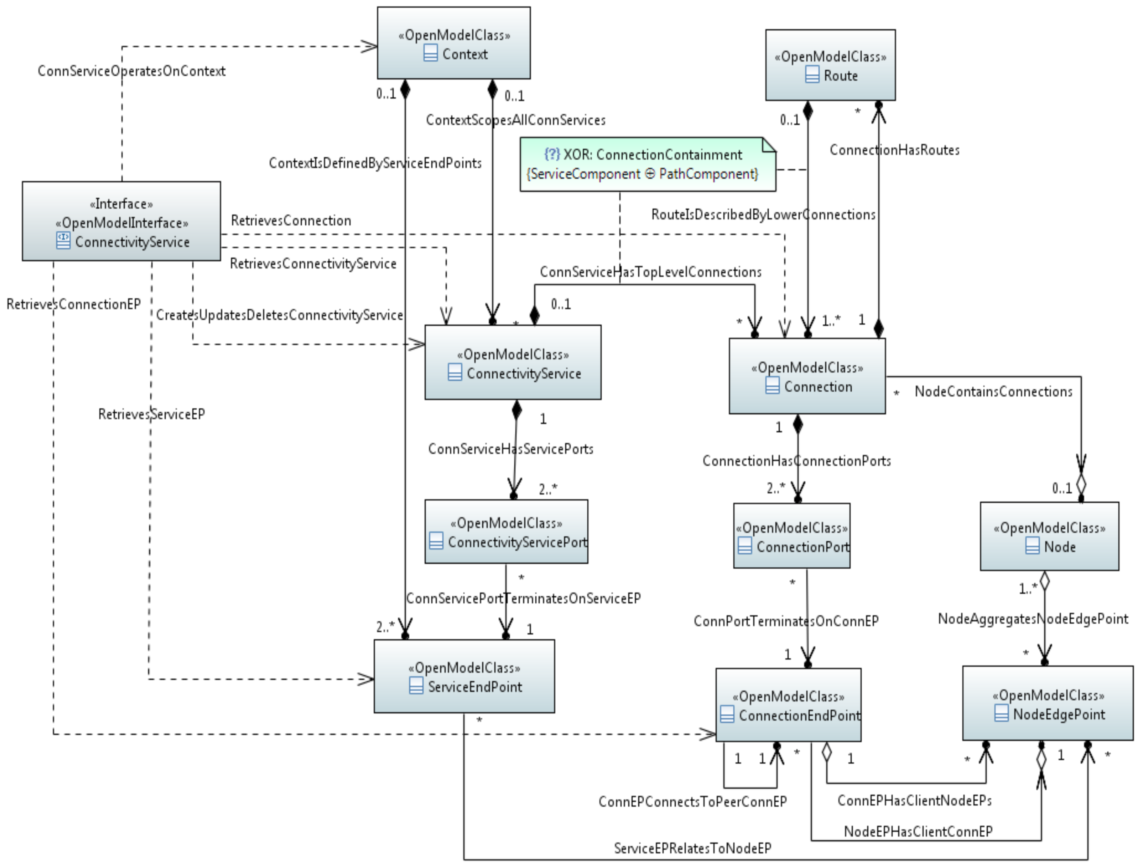

«OpenModelClass» Context
«OpenModelClass» Route
ConnServiceOperatesOnContext
0..1
0..1
ContextScopesAllConnServices
0..1
*
ConnectionHasRoutes
{?} XOR: ConnectionContainment
{ServiceComponent ⊕ PathComponent}
ContextIsDefinedByServiceEndPoints
«Interface» «OpenModelInterface» ConnectivityService
RetrievesConnection
RouteIsDescribedByLowerConnections
RetrievesConnectivityService
ConnServiceHasTopLevelConnections
RetrievesConnectionEP
CreatesUpdatesDeletesConnectivityService
0..1
*
1..*
1
«OpenModelClass» ConnectivityService
«OpenModelClass» Connection
NodeContainsConnections
RetrievesServiceEP
1
1
ConnServiceHasServicePorts
ConnectionHasConnectionPorts
2..*
2..*
0..1
«OpenModelClass» ConnectivityServicePort
«OpenModelClass» ConnectionPort
«OpenModelClass» Node
ConnServicePortTerminatesOnServiceEP
ConnPortTerminatesOnConnEP
1..*
NodeAggregatesNodeEdgePoint
2..*
1
1
«OpenModelClass» ServiceEndPoint
«OpenModelClass» ConnectionEndPoint
«OpenModelClass» NodeEdgePoint
ConnEPConnectsToPeerConnEP
ConnEPHasClientNodeEPs
NodeEPHasClientConnEP
ServiceEPRelatesToNodeEP

Figure 39: Connectivity Service Skeleton

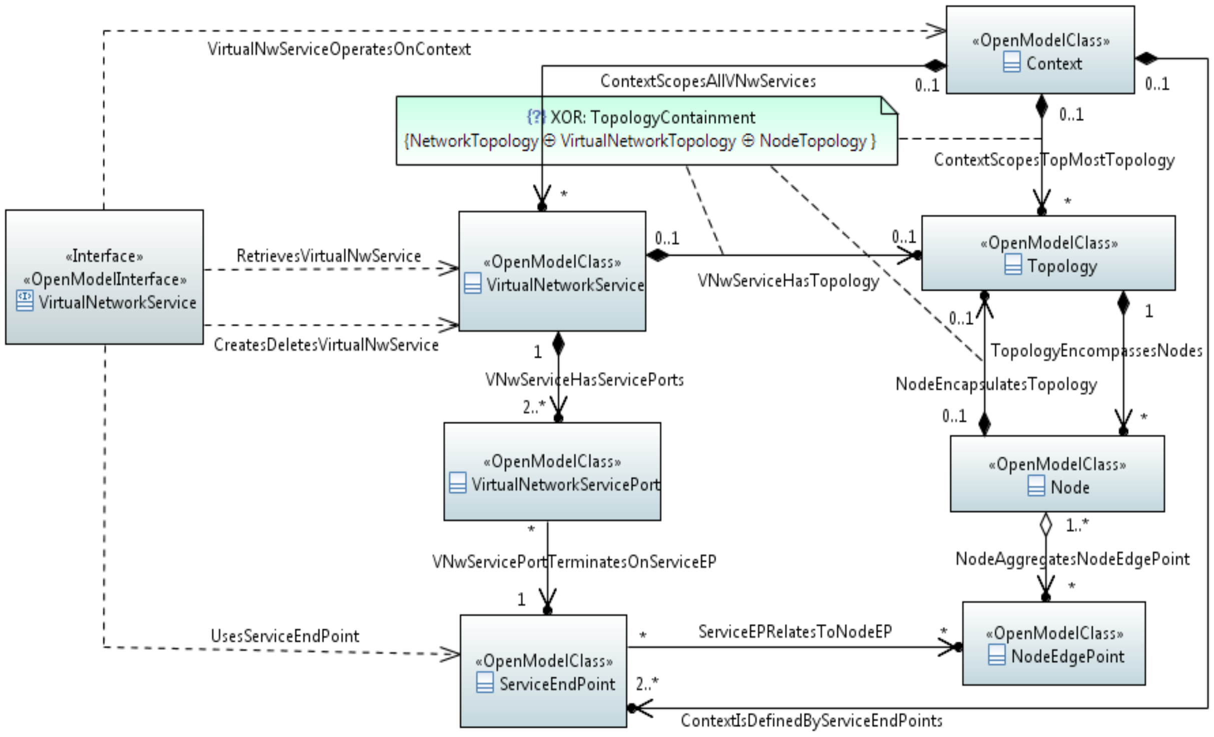

VirtualNwServiceOperatesOnContext
«OpenModelClass» Context
ContextScopesAllVNwServices
0..1
0..1
0..1
{?} XOR: TopologyContainment
{NetworkTopology ⊕ VirtualNetworkTopology ⊕ NodeTopology }
ContextScopesTopMostTopology
*
*
«Interface» «OpenModelInterface» VirtualNetworkService
RetrievesVirtualNwService
CreatesDeletesVirtualNwService
«OpenModelClass» VirtualNetworkService
0..1
VNwServiceHasTopology
0..1
«OpenModelClass» Topology
0..1
1
TopologyEncompassesNodes
NodeEncapsulatesTopology
0..1
*
1
VNwServiceHasServicePorts
2..*
«OpenModelClass» VirtualNetworkServicePort
«OpenModelClass» Node
1..*
NodeAggregatesNodeEdgePoint
*
VNwServicePortTerminatesOnServiceEP
1
UsesServiceEndPoint
«OpenModelClass» ServiceEndPoint
*
ServiceEPRelatesToNodeEP
*
«OpenModelClass» NodeEdgePoint
2..*
ContextIsDefinedByServiceEndPoints

Figure 40: Virtual Network Service Skeleton

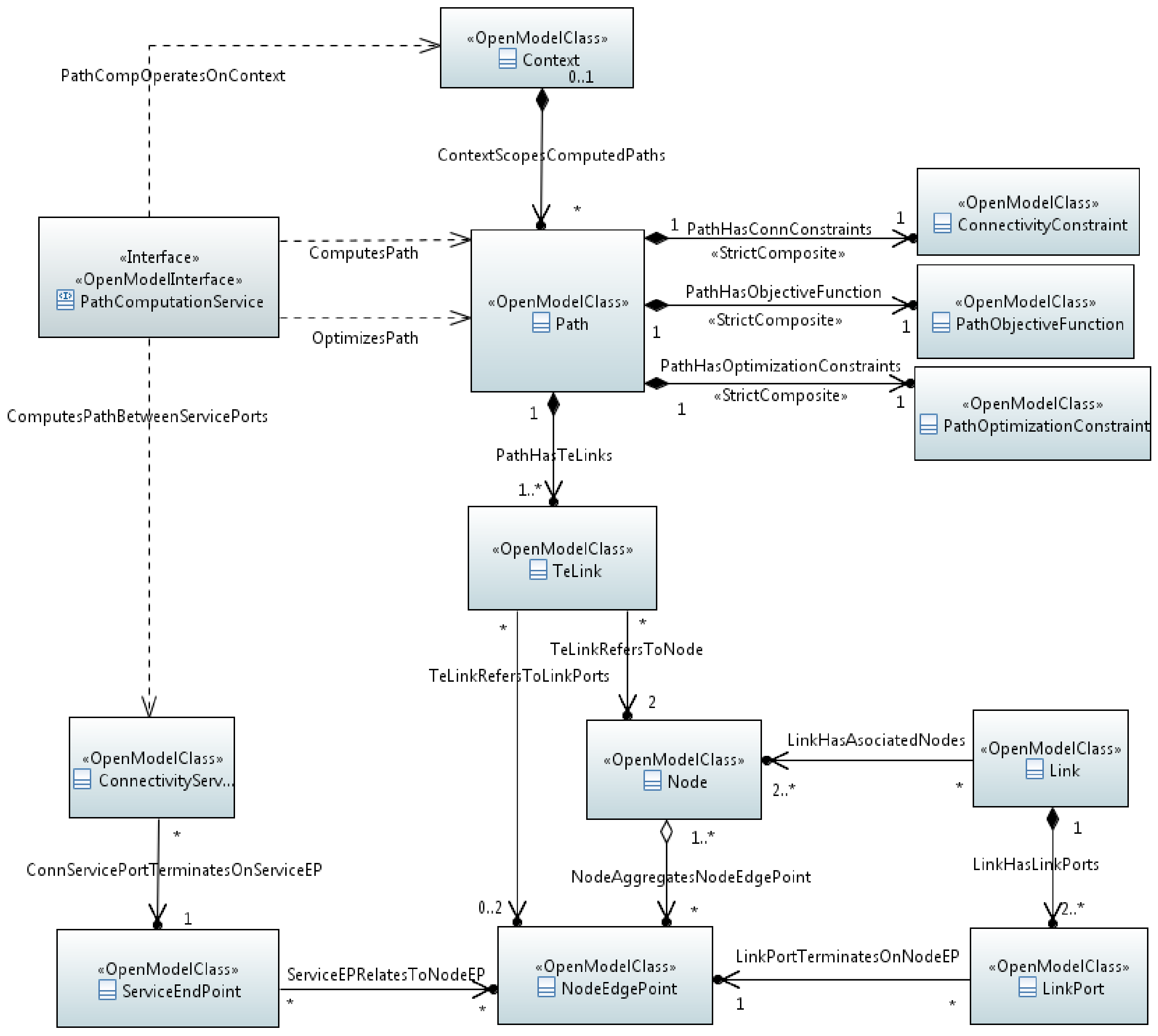

«OpenModelClass» Context
0..1
PathCompOperatesOnContext
ContextScopesComputedPaths
*
«Interface» «OpenModelInterface» PathComputationService
ComputesPath
OptimizesPath
«OpenModelClass» Path
1 PathHasConnConstraints 1
«StrictComposite»
«OpenModelClass» ConnectivityConstraint
PathHasObjectiveFunction
«StrictComposite»
1
1
«OpenModelClass» PathObjectiveFunction
PathHasOptimizationConstraints
«StrictComposite»
1
1
«OpenModelClass» PathOptimizationConstraint
ComputesPathBetweenServicePorts
1
PathHasTeLinks
1..*
«OpenModelClass» TeLink
*
*
TeLinkRefersToNode
TeLinkRefersToLinkPorts
2
«OpenModelClass» Node
LinkHasAsociatedNodes
2..*
*
«OpenModelClass» Link
«OpenModelClass» ConnectivityServ...
*
ConnServicePortTerminatesOnServiceEP
1
1..*
NodeAggregatesNodeEdgePoint
1
LinkHasLinkPorts
0..2
*
2..*
«OpenModelClass» ServiceEndPoint
ServiceEPRelatesToNodeEP
*
*
«OpenModelClass» NodeEdgePoint
LinkPortTerminatesOnNodeEP
1
*
«OpenModelClass» LinkPort

Figure 41: Path Computation Service Skeleton

8 Contributors

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

- Chen Qiaogang, ZTE
- Erez Segev, ECI
- Eve Varma, ALU
- Guoying Zhang, CATR
- Hui Ding, RITT
- Italo Busi, Huawei
- Jia He, Huawei
- Karthik Sethuraman, NEC (Editor)
- Lyndon Ong, Ciena
- Nigel Davis, Ciena
- Ricard Vilalta, CTTC
- Sergio Bellotti, ALU
- Victor Lopez, Telefonica

Special thanks to everyone who provided their input and comments to make this a better document.