B 5 G OPEN

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016608.

BEYOND 5G – OPTICAL NETWORK CONTINUUM
(H2020 – Grant Agreement Nº 101016663)

Deliverable D4.2

Specification of Interfaces, Protocols Component design and implementation

Editor R. Casellas (CTTC)

Contributors CTTC, TID, UC3M, TIM, INF-P, ADTRAN, NBLF, CNIT, CNR, UPC, OLC-E, ELIG, TuE, PLF

Version 2.0

Date October 31, 2023

Distribution PUBLIC (PU)

DISCLAIMER

This document contains information, which is proprietary to the B5G-OPEN (Beyond 5G – OPtical nEtwork coNtinuum) consortium members that is subject to the rights and obligations and to the terms and conditions applicable to the Grant Agreement number 101016663. The action of the B5G-OPEN consortium members is funded by the European Commission.

Neither this document nor the information contained herein shall be used, copied, duplicated, reproduced, modified, or communicated by any means to any third party, in whole or in parts, except with prior written consent of the B5G-OPEN consortium members. In such case, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced. In the event of infringement, the consortium members reserve the right to take any legal action they deem appropriate.

This document reflects only the authors' view and does not necessarily reflect the view of the European Commission. Neither the B5G-OPEN consortium members as a whole, nor a certain B5G-OPEN consortium member warrant that the information contained in this document is suitable for use, nor that the use of the information is accurate or free from risk and accepts no liability for loss or damage suffered by any person using this information.

The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

REVISION HISTORY

Revision	*Date*	*Responsible*	*Comment*
1.0	*November 3, 2023*	*ALL*	*Final version with reviewed sections.*
2.0	*December 12, 2023*	*Lutz Rapp*	*Quality check*

LIST OF AUTHORS

Partner ACRONYM	*Partner FULL NAME*	*Name & Surname*
TID	*Telefonica I+D*	*Oscar González de Dios*
UC3M	*Universidad Carlos III de Madrid*	*José Alberto Hernández, Alfonso Sánchez-Macián, Gonzalo Martínez*
TIM	*Telecom Italia*	*Roberto Morro, Marco Quagliotti, Emilio Riccardi*
CTTC	*Centre Tecnològic de Telecommunicacions de Catalunya*	*Ramon Casellas, Laia Nadal, Michela Svaluto Moreolo, Fco. Javier Vilchez, Ricardo Martinez*
Adtran	*Adtran Networks SE*	*Achim Autenrieth, Stefan Zimmermann, Vignesh Karunakaran, Jasper Müller*
CNIT	*CNIT*	*Davide Scano, Filippo Cugini*
CNR	*CNR*	*Alessio Giorgetti*
ELIG	*E-lighthouse Network Solution*	*Francisco-Javier Moreno-Muro, Pablo Pavón Mariño, José-Manuel Martínez-Caro*
NBL	*Nokia Bell Labs*	*Fabien Boitier, Patricia Layec, Vinod Bajaj*
UPC	*Universitat Politecnica de Catalunya*	*Luis Velasco, Marc Ruiz, Jaume Comellas, Salvatore Spadaro, Davide Careglio, Josep Prat*
OLC-E	*OpenLightComm Europe s.r.o.*	*Alexandros Stavdas, Evangelos Kosmatos, Christos Matrakidis*
PLF	*pureLiFi Ltd*	*Rui Bian*
INF-P	*Infinera Unipessoal Lda*	*João Pedro*

GLOSSARY

Acronym	Expansion	Acronym	Expansion
AI	Artificial Intelligence	NOS	Node Operating System
ANN	Artificial Neural Network	NS	Network Service
AP	Access Point	NSS	Network Slice Subnet
API	Application Programming Interface	OA	Optical Amplifier
ASIC	Application Specific Integrated Circuit	OADM	Optical Add/Drop Multiplexer
BF	Bloom Filter	OCh	Optical Channel
BGP	Border Gateway Protocol	ODTN	Open and Disaggregated Transport Network
CD	Chromatic Dispersion	OIF	Optical Internetworking Forum
CD	Colorless/Directionless	OLS	Optical Line System
CDB	Configuration Data Base	OLT	Optical Line Terminal
CEP	Connection End Point	ONF	Open Networking Foundation
CLI	Command Line Interface	ONOS	Open Network Operating System
CMIS	Content Management Interoperability Services	ONT	Optical Network Termination
CMS	Count-Min Sketch	ONU	Optical Network Unit
CPU	Central Processing Unit	OOPT	Open Optical Packet Transport
CRUD	Create, Read, Update and Delete	OPCE	Optical Path Computation Element
CS	Connectivity Service	OPM	Optical Packet Metro
CSEP	Connectivity Service End Point	OSNIR	Optical Signal-To-Noise plus Interference Ratio
DB	Database	OSNR	Optical Signal-To-Noise Ratio
DGD	Differential Group Delay	OSPF	Open Shortest Path First
DNN	Deep Neural Network	OTN	Optical Transport Network
DPSM	Data Path State Machine	OTSi	Optical Tributary Signal
DSR	Digital Signal Rate	P2MP	Point-To-MultiPoint
EDFA	Erbium Doped Fiber Amplifier	PCE	Path Computational Engine
EEPROM	Electrically Erasable Programmable Read-Only Memory	PLA	Physical Layer impairment Aware
FEC	Forward Error Correction	PLI	Physical Layer impairment
gNMI	gPRC Network Management Interface	PMD	Polarization Mode Dispersion
gPRC	gPRC Remote Call Procedure	PON	Passive Optical Network
GUI	Graphical User Interface	PSD	Power Spectrum Density
HLL	Hyper-LogLog	PSU	Power Supply Unit
HTTP	Hypertext Transfer Protocol	RAM	Random Access Memory
HW	Hardware	RF	Remote Feeder

IBN	Intent-Based Networking
IETF	Internet Engineering Task Force
ILA	In-Line Amplifier
IP	Internet Protocol
IPoWDM	IP over WDM
IT	Information Technology
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LI	Linear Impairments
LSP	Label Switched Path
LTS	Long Term Support
MB-PCE	MultiBand PCE
ML	Machine Learning
MPLS	Multiprotocol Label Switching
MSM	Module State Machine
MUST	Mandatory Uses Cases for Transport SDN
NBI	North Bound Interface
NCF	Nominal Central Frequency
NE	Network Element
NEP	Node Edge Point
NF	Noise Figure
NIC	Network Interface Card
NLI	Nonlinear Interference
NLI	Non Linear Impairments
NMC	Network Media Channel
NMS	Network Management System
RMSA	Routing, Modulation and Spectrum Assignment
ROADM	Reconfigurable Optical Add/Drop Multiplexer
RPC	Remote Procedure Call
RSA	Routing and Spectrum Assignment
SBI	Southbound Interface
S-BVT	Sliceable Bandwidth/bitrate Variable Transceiver
SDN	Software Defined Networking
SIP	Service Interface Point
SLA	Service Level Agreement
SMA	Spectral and Modulation Assignment
SNR	Signal To Noise Ratio
SONiC	Software for Open Networking in the Cloud
SRG	Shared Risk Group
SSID	Service Set Identifier
TAI	Transponder Abstraction Interface
TAPI	Transport API
TR	Techical Recommendation
UUID	Universallu Unique Identifier
VDM	Versatile Diagnostics Monitoring
VLAN	Virtual Local Area Network
VM	Virtual Machine
VRF	Virtual Routing and Forwarding
VSI	Virtual Switch Instance
WPA	WiFi Protected Access
WSS	Wavelength Selective Switch
YANG	Yet Another Next Generation

EXECUTIVE SUMMARY

Deliverable D4.1 summarized the main Year 1 contributions of W4 and provided B5G-OPEN key requirements, targeted services along with existing frameworks and functional elements and defined the *service orchestration and infrastructure control system* (commonly referred to as the Control Plane) which has been used as the reference architecture for work done in Year 2 and upcoming Year 3.

This document summarizes the interfaces and components required for the service orchestration and infrastructure control system, as well as the implementation of the different components developed in T4.1, T4.2 and T4.3. For each identified component, the deliverable reports component functional tests, and individual KPIs in view of their subsequent integration within WP5 demos. In this sense D4.2 complements the previous deliverable by specifying the interfaces for such a modular architecture, which must rely on standard and open interface definitions between the control plane functions and towards the devices.

After a short introduction of the B5G-OPEN control plane, Section 2 details the set of reference points and interfaces that have been adopted and extended to allow the different functional components to interwork. The interfaces covered are:

- **OpenConfig and OpenROADM data models** for the configuration of packet/optical nodes, as well as Open ROADM, including aspects related to device discovery and configuration.
- **The usage of P4 and P4 runtime** for the configuration of packet switching in the packet optical whiteboxes, enabling packet controllers to either provision packet flows in the nodes, or delegate to IP networking by configuring aspects of the IGP/EGP protocols.
- **ONF Transport Application Programming Interface** (T-API) for the purposes of enabling an end user or network orchestrator to perform topology management and service provisioning as well as for enabling externalized path computation in a multiband optical network.
- **The ONOS™ SDN controller north bound interface**, which is open and has been extended for specific purposes related to multi-band networking.
- **Standards relevant to the configuration of pluggable elements** as well as the usage of adaptors to enable remote programmability and configuration.
- **Interfaces for the configuration and control of PON and LiFi access networks**, enabling an end to end network orchestration from an end-to-end perspective, spanning the segments of optical access, metro, and long haul.
- **Interfaces from the point of view of the user or operator, enabling the provisioning of cloud-based services such a slicing**, by means of integration with the Kubernetes container orchestration system.

The interfaces are briefly introduced, and key usage paths are detailed, which have been used by implementors to ensure system interworking. Similarly, B5G-OPEN has defined a generic telemetry platform enabling straightforward adaptations of devices or systems as data sources:

- Interfaces allowing the production of time series or events data (e.g., timestamped events data) having a common collector and processor system. Examples of telemetry sources are the TAPI network orchestrator, LiFi access points or spectrum monitoring devices.

Starting from Section 3, each component developed in WP4 is detailed. For each component, the relevant sections provide a short introduction and description as well as the set of interfaces it relies on and implements -- based on what has been detailed in Section 2 – as well as functional validations that have been carried out, integrations between different components, and updated roadmap compared to what it was defined in D4.1 and a set of individual component Key Performance Indicators (KPIs) that have been evaluated or shall be evaluated in the scope of WP4/WP5.

The following table provides the list of components that have been or are in the progress of being integrated.

Component	Partner	Purpose
B5G-ONP network planner	ELIG	The B5G-OPEN Optical Network Planner (B5G-ONP) component orchestrates both IT and network resources. Within the B5G-OPEN project, the B5G-ONP serves as the hub and provides design, optimization, and planning tools to deploying, managing, and configuring services and resources.
TAPI enabled Network Orchestrator	CTTC	The TAPI-enabled Optical Network Orchestrator is responsible for: i) providing a uniform, open and standard view and interface to the higher levels and components; ii) Composing a complete Context to be consumed by B5G-OPEN network planner and additional consumers combining information retrieved from subsystems and sub-controllers; iii) Enabling a single entry point for provisioning DSR and Photonic Media services, including externalized path computation and iv) providing an event telemetry data source that reports events that happen asynchronously in the network.
Path Computation Element	OLC-E	The Multi-Band Path Computation Engine (MB-PCE) is based on a multi-band routing engine which ensures that: i) routing is implemented by means of an efficient spectrum and modulation-format assignment; and ii) the impact of physical layer effects over the selected optical paths is estimated and the results are benchmarked against QoT target values (BER, OSNIR, OSNR, etc).
ONOS Optical Controller	CNIT	The optical controller is based on the ONOS open-source project [ONOS] that, besides the control of optical devices, also provides a suitable environment for the control of packet devices (e.g., based on OpenFlow or P4Runtime protocols). The main roles of the optical controller in the B5G-OPEN project are: (i) retrieve device descriptions from data plane and abstract them toward the upper control layers; (ii) receive the service configuration requests by the upper control layers and translate such requests in

		a set of configuration messages to be forwarded to each involved device.
OLS Controller	Adtran CTTC	The OLS controller used in B5G-OPEN for an Adtran OLS is based on the Adtran Ensemble Network Controller software solution and is offering a northbound ONF Transport-API (TAPI) towards the Optical Controller.
PON Controller	OLC-E	The Access Controller is responsible for: a) monitor the PON network and receive any requests for PON reconfiguration; b) translate these requests into high level traffic requests that will be reported to the B5G-ONP App; c) execute the appropriate actions in the PON Controller in order to support the new requests. In addition, the Access Controller will communicate with the LiFi Controller for retrieving any connection/traffic requests
LiFi Controller	PLF	The LiFi controller serves as the central component responsible for managing LiFi APs in the network. It is strategically positioned between the PON controller and the LiFi AP. This specific positioning ensures seamless communication and enhanced coordination between the optical network layer and the wireless LiFi communication layer
LiFi agent	PLF	The LiFi Agent acts as a central hub in the architecture of the LiFi AP. With the advancement of NETCONF capabilities, the agent provides a seamless way for the AP to interact with other components, offering a structured interface for configurations and management
OpenROADM agent	TIM	The OpenROADM agent is an implementation of a NETCONF server controlling optical network elements using OpenROADM device models
OpenConfig agent	CTTC CNIT	OpenConfig agent is an implementation of an SDN agent using NETCONF/YANG with the OpenConfig data models. It implements a subset of the data models, namely the OpenConfig platform and optical transport as well as some extensions devised in the context of B5G-OPEN to report details about the transceiver operational mode
SONiC based packet optical node	CNIT	The Software for Open Networking in the Cloud, i.e., SONiC, [SONIC] is considered as the Network Operating System (NOS) to be deployed on packet-optical IPoWDM nodes operated in metro/aggregation networks. Within the B5G-OPEN project SONiC has been extended with several components provided in the form of docker containers.
AI/ML models for PSD and Power Management	NOKIA	Machine learning application towards augmented optical networks and is called “Automatic power correction”
Telemetry System	UPC CTTC	B5G/OPEN distributed telemetry system integrates measurements and event data collection and supports intelligent data aggregation nearby data collection, so

	Adtran Nokia	agents receive and analyze measurements before sending to a centralized manager.
Mesarthim – Failure management Using a SNR Digital Twin	UPC	MESARTHIM compares the QoT measured in the transponders with the one estimated using a QoT tool. Deviations can be explained by changes in the value of input parameters of the QoT model representing the optical devices, like noise figure in optical amplifiers and reduced Optical Signal to Noise Ratio in the Wavelength Selective Switches. By applying reverse engineering, MESARTHIM estimates the value of those modelling parameters as a function of the observed QoT of the lightpaths
Ocata - Digital Twin for the Optical Time Domain	UPC	OCATA is a deep learning-based digital twin for the optical time domain that is based on the concatenation of deep neural networks (DNN) modelling optical links and nodes, which facilitates representing lightpaths. The DNNs model linear and nonlinear noise, as well as optical filtering

As a summary, WP4 has completed the second year with key objectives fulfilled. This year has been mostly focused on developing the different software components; specifying the interfaces and protocols to enable the interconnection of such components, making sure that the components present neither functional gaps, nor regressions and evaluating their KPIs.

All the tasks, T4.1 - T4.3, are progressing well, covering the implementation of SDN-based technologies and solutions to operate on packet-optical nodes based on SONiC and using coherent pluggable modules, addressing the design, development and validation of a generalized orchestration and control plane system and closed the hierarchical and distributed telemetry system including realistic data available for testing different algorithms and the optical time domain digital twin.

The deliverable has set the foundations for the final WP4 year, which will focus on missing aspects related to further integrating the components in view of the demonstrations, addressing transparent multi-domain scenarios, and covering network autonomy with closed-loop operations involving the SDN control plane and the telemetry system.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLE

1 INTRODUCTION

D4.1 detailed the Infrastructure Control and Service Management architecture to be implemented in the B5G-OPEN Control, Orchestration and Telemetry System (control plane for short), along with a set of initial requirements, and existing frameworks. After outlining a set of initial requirements and use cases, carefully developed with WP2, and extended to the control plane, D4.1 listed the services that the control plane of B5G-OPEN must support such as point-to-point optical connectivity, IP link provisioning, or telemetry. Next, D4.1 contained thorough review of existing frameworks in the literature, different tools for implementing network telemetry systems, main orchestration frameworks and Quality of Transmission (QoT) estimation tools. The core of the document was the definition of different parts of the control plane architecture, including service orchestration and planning, optical-packet integration systems and telemetry and intent-based networking, covering the SDN control of optical multi-band networks, the control of different access technologies (PON, LiFi, etc) and IT and network resource orchestration platform for autonomous network operations.

The main innovations for the control plane of B5G-OPEN are:

- [**multiband control**] Control of an optical multi-band network, this means being able to exploit the multiband capabilities of optical devices such as transmission (Tx) elements (transceivers) or switching elements (multi-band ROADMs).
- [**transparent multi-domain**, *domain-less*] The ability to set up connections in a transparent manner, across multiple domains and network segments. This is exemplified in the "*multi-OLS*" scenario, in which different optical line systems are interconnected without a O/E/O conversion. There is a systematic need to extend SDN principles to networks composed of multiple domains and technological layers.
- [**Packet/optical integration**] The evolution from discrete optics towards pluggable interfaces is challenging the design of the control plane which, to a large extent, has considered the control plane of the IP/MPLS layer largely decoupled from the control plane of the optical layer. Current architectures for the SDN control plane of the transport network consider the scope of the control limited to transceiver to transceiver and the tunability of the transceiver was directly under the control of the optical SDN controller. Multi-layer networking was commonly accomplished typically with a hierarchical arrangement of controllers (a packet controller and an optical controller under the orchestration of a parent controller). This is addressed in B5G-OPEN, considering multiple options including *exclusive* or *concurrent* control.
- [**physical layer impairments, PLI**] Accounting for PLI is critical to efficiently plan and operate optical networks and high data rates, with increasing non-linear effects. When considering the extension to wide-band, such parameters can be specific to certain frequency bands and one can no longer assume uniform channel behaviour.
- [**telemetry**] The scope of the SDN no longer covers exclusively device / system control and configuration aspects but extends to optical monitoring and telemetry, a key enabled for advanced functions such as autonomous/autonomic networking via hierarchical and coordinated closed loops. Streaming Telemetry protocols and architectures such as gRPC/gNMI are increasingly being used to export telemetry data from devices, providing flexibility in the definition of streams, filtering, and use cases.
- [**external planning tools**] Planning tools, including QoT estimators or path computation and validation systems, need efficient access (in terms of retrieval, storage and

processing) to collected and managed data. Algorithm inputs need to be modelled in an efficient and scalable way, defining dynamic workflows with controlled and minimized impact on service provisioning latency. Algorithmically, functional elements dedicated to generalized Routing and Spectrum Assignment (RSA) or function placement are needed and are expected to operate in hybrid off-line/on-line modes, e.g., dynamically, used to compute/validate e.g

- [**network automation**] Aspects related to automation, zero touch networking and Intent Based Networking (IBN) are developed in the areas of service deployment, network planning and overall network operation. Outcomes related to automation in single domains and later cross-domain automation (across technology layers or network segments).

D4.2 complements the previous deliverable by specifying the Interfaces for such a modular architecture, which must rely on standard and open interfaces between the control plane functions and towards the devices.

After a short introduction regarding B5G-OPEN control plane, Section 2 details the set of reference points and interfaces that have been adopted and extended to allow the different functional components to interwork. Such interfaces are briefly introduced, and key usage paths detailed, which have been used by implementors to ensure system interworking. Similarly, B5G-OPEN has defined a generic telemetry platform enabling straightforward adaptations of devices or systems as data sources.

Starting from Section 3, each component developed in WP4 is detailed. For each component, the relevant sections provide a short introduction and description as well as the set of interfaces it relies on and implements -- based on what has been detailed in Section 2 – as well as functional validations that have been carried out, integrations between different components , and updated roadmap compared to what it was defined in D4.1 and a set of individual component Key Performance Indicators (KPIs) that have been evaluated or shall be evaluated in the scope of WP4/WP5.

2 B5G-OPEN INTERFACES AND PROTOCOLS

This section provides an overview of the main interfaces and protocols that are considered for the interactions between B5G-OPEN control plane components and towards external systems (such as Operational Support Systems) and Network devices (including the prototypes developed by WP3).

2.1 OPENCONFIG FOR PACKET AND OPTICAL DEVICES

The OpenConfig project [OpenConfig] is a collaborative effort by network operators to develop interfaces and tools for managing networks in a dynamic, vendor-neutral way. Thus, its models are periodically updated. All OpenConfig models are available on github [OpenConfig]. The OpenConfig information model is composed by a set of abstract modules. Each one is composed by a set of YANG models and represents a specific capability and features of a network device, such as HW components hierarchy, interfaces, OSPF configuration, QoS, among others.

The main modules used for Packet and Optical SDN operations are the following:

Platform:

- Platform - openconfig-platform.yang – It constitutes the main model to define the **hardware components** of a network device.
- CPU - openconfig-platform-cpu.yang – It augments the platform model to add specific parameters of a CPU component.
- FAN - openconfig-platform-fan.yang – It augments the platform model to add specific parameters of a FAN component.
- LINECARD - openconfig-platform-linecard.yang – It augments the platform model to add specific parameters of a Linecard component.
- PORT - openconfig-platform-port.yang – It augments the platform model to add specific parameters of a port component.
- PSU - openconfig-platform-psu.yang – It augments the platform model to add specific parameters of a PSU (Power Suply Unit) component.
- TRANSCEIVER - openconfig-platform-transceiver.yang – It augments the platform model to add specific parameters of a Transceiver component.
- Platform Types - openconfig-platform-types.yang – It defines the types used to define the parameters in the platform module parameters.

Optical-Transport:

- Terminal Device - openconfig-terminal-device.yang – It defines the main model to define a terminal optics device.
- Optical Transport Types - openconfig-transport-types.yang – It defines the types used to define the parameters in the optical transport module parameters.

Terminal device manifest

OpenConfig has defined the manifest files, a special type of model which is not configuration nor operation. A remote controller requires some data from the transceiver in order to perform optical planning and impairment validation of the end-to-end transmission across an Optical Line System (OLS). When a pluggable module is recognized by a terminal device (which can be a transponder or a packet-optical box), the operational mode datastore is updated.

- **Operational-mode-capabilities:** this set of attributes contains all characteristic information of the signal (modulation format, FEC, bit rate...), relevant information for the physical impairment validation (OSNR Rx sensitivity, CD/PMD tolerance and penalties).
- **Optical-channel-config-value-constrains:** Contains the transmission configuration constrains/ranges of the optical-channel's attributes characterized by the operational-mode, i.e., the central frequency range, the frequency grid and the configurable transmitted power.

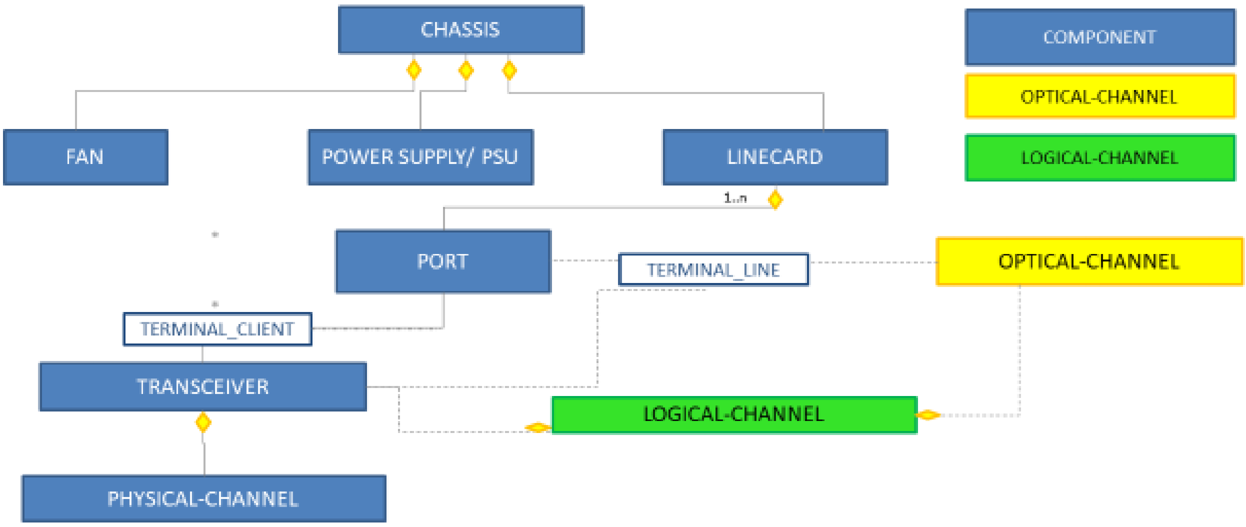


Figure 2-1 Hierarchy of components of an open terminal device

The structure of the OpenConfig component hierarchy is shown in Figure 2-1. The main modules that are required for packet (IP/MPLS) control are:

bgp: This set of modules describe the BGP protocol configuration. They are used in the service-related use cases to handle the BGP protocol and to support IP Connectivity.

interfaces: Model for managing network interfaces and sub interfaces. For the use cases that are currently defined, it is used to configure the line side interfaces after setting up the optical connectivity.

local-routing: This module describes configuration and operational state data for routes that are locally generated, i.e., not created by dynamic routing protocols. It can be used with network-instances to configure the static routes.

network-instance: The network instance is an abstraction of a packet forwarding device. It may be a Layer 3 forwarding construct such as a virtual routing and forwarding (VRF) instance or the Global routing instance. A Layer 2 instance such as a virtual switch instance (VSI) and Mixed Layer 2 and Layer 3 instances are also supported. The network instance works in conjunction with other modules such as:

- Interfaces
- VLANs
- Potocols

2.1.1 Device discovery Workflow

2.1.1.1 Retrieve Components

The first operation that the controller needs to perform over an open terminal device or a packet/optical device with coherent pluggable modules is to retrieve the list of components. In principle, the structure of both types of devices need to follow the OpenConfig platform model (explained in the previous section).

```
<get>
   <filter type="subtree">
     <components xmlns="http://openconfig.net/yang/platform">
     </components>
   </filter>
</get>
```

The retrieve operation is implemented with the get NETCONF message as reported above. Obtained reply message allows to discover all generic details of the device such as the device model, the manufacturer, the software version and so on. Moreover, the same reply also contains all the details regarding the device interfaces. Thus, parsing such information is possible, at the SDN controller, to fill-up the list of ports supported by the device, including their details, for instance the supported bitrate for the packet ports and the tunability range for optical ports.

A component of type PORT with a sub-component of type TRANSCEIVER is identified as a client port of the terminal device. A component of type PORT with a sub-component of type OPTICAL_CHANNEL is identified as a line port of the terminal device.

2.1.1.2 Listing Operational Modes

```
<get>
   <filter type="subtree">
     <operational-modes xmlns="http://openconfig.net/yang/platform">
     </components>
   </filter>
</get>
```

In the current OpenConfig standard, the supported operational modes are defined at the level of the device and are not associated with specific interfaces. The get NETCONF message reported above can be used to retrieve the details regarding all operational modes supported by a device. Each operational mode is identified by an ID. In WP3, B5G-OPEN has defined a way of characterizing the operational modes, as detailed later in the different components of this document.

2.1.2 Configuration of the Optical Channel

After discovery of the device details, the configuration of an optical channel can be executed by the SDN controller. Specifically, this phase involves the configuration of source and destination transponders and all intermediate ROADMs. In case the end-points of the optical channel are coherent pluggables located in IPoWDM nodes, the control plane implementation considered by B5G-OPEN assumes that the Optical SDN controller also performs the configuration of the coherent pluggable parameters.

The request of a new optical channel is received by the ONOS SDN controller on its NBI interfaces that has been mainly inherited from the METRO-HAUL project, but it has been extended in B5G-OPEN to support the utilization of flexi-grid and multi-band optical channels. The request is typically submitted including a suggested path and frequency slot (e.g., previously computed by QoT evaluation tools).

The configuration is performed in three subsequent phases. 1) The configuration of the OPTICAL_CHANNEL component. Where three are the parameters to be configured, i.e., the central frequency, the target output power and the operational mode.

```
<edit-config>
 <components xmlns='http://openconfig.net/yang/platform’>”
  <component>
   <name>channel-12</name>
    <optical-channel xmlns:oc-opt-term='http://openconfig.net/yang/terminal-device’>
     <config>
      <frequency>193100</oc-opt-term:frequency>
      <target-output-power>1.0</oc-opt-term:target-output-power>
      <operational-mode>100</oc-opt-term:operational-mode>
     <line-port>port-12</oc-opt-term:line-port>
    </config>
   </optical-channel>
  </component>
 </components>
</edit-config>
```

2) The configuration of the LOGICAL_CHANNEL, mainly due to activate the interface.

```
<edit-config>
 <terminal-device xmlns='http://openconfig.net/yang/terminal-device’>”
  <logical-channels>
   <channel>
    <index>12</index>
    <config>
     <admin-state>ENABLED</admin-state>
    </config>
   </channel>
  </logical-channels>
 <terminal-device>
</edit-config>
```

3) The configuration of the assignment between a client port and a line port.

```
<edit-config>
  <terminal-device xmlns='http://openconfig.net/yang/terminal-device’>”
    <logical-channels>
      <channel>
        <index>client-port-index</index>
        <config>
          <admin-state>ENABLED</admin-state>
        </config>
        <logical-channel-assignments>
          <assignment>
            <index>1</index>
            <config>
              <logical-channel>logical-channel-id</logical-channel>
              <allocation>1</allocation>
            </config>
          <assignment>
        <logical-channel-assignment>
      </channel>
    </logical-channels>
  <terminal-device>
</edit-config>
```

2.1.3 Operational Mode Extensions for Physical Layer Characterization

In cooperation with B5G-OPEN WP3, WP4 has defined a set of attributes and properties that extend an operational mode in such a way that clients may retrieve the specific physical layer information associated with that transmission mode. This has been accomplished by extending the OpenConfig Yang modules in two different files:

```
openconfig-terminal-device-properties.yang
openconfig-terminal-device-property-types.yang
```

Mainly, these files contain a module to extend OpenConfig terminal device's operational modes' data. It supports operational modes for one Optical Channel, with a single OTSi. The operational mode includes key attributes such modulation format, symbol rate, nominal central frequency (NFC) tunability constraints (grid, min/max NCF), FEC gain, minimum and maximum output power of the transmitter or minimum OSNR at the receiver as well as the spectrum width of the OTSi (OTSiMC). It also includes (optional) aspects such as filter characterization, CD and DGD tolerance. The module is captured in the following Yang fragment:

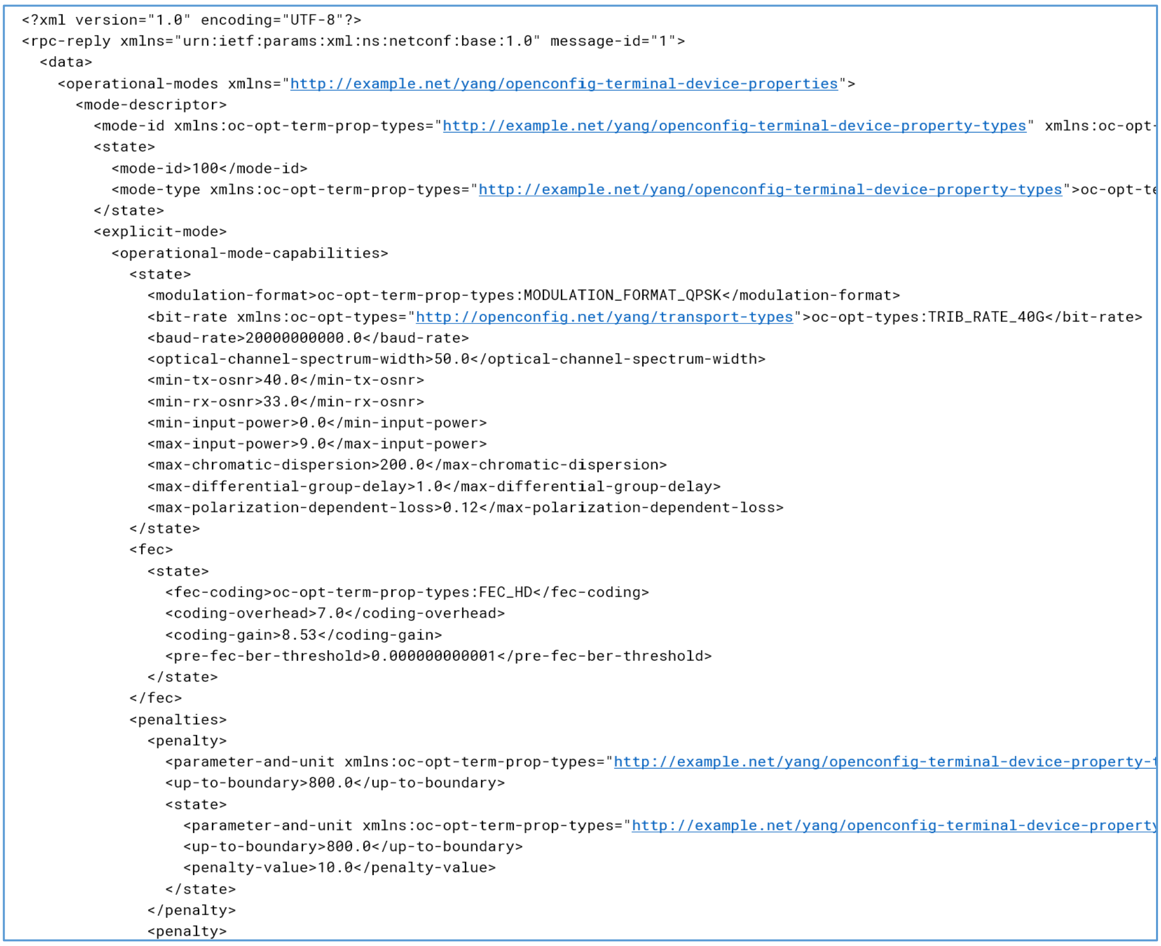
```
module: openconfig-terminal-device-properties
  +--ro operational-modes
     +--ro mode-descriptor* [mode-id]
        +--ro mode-id          -> ../state/mode-id
        +--ro state
        |  +--ro mode-id?     uint16
        |  +--ro mode-type?   identityref
        +--ro G.698.2
        |  +--ro state
        |     +--ro standard-mode?   oc-opt-term-prop-types:standard-mode
        +--ro explicit-mode
           +--ro operational-mode-capabilities
           |  +--ro state
           |  |  +--ro modulation-format?                union
           |  |  +--ro bit-rate?                         oc-opt-term-prop-types:bit-
```

```
rate
           |  |  +--ro baud-rate?                          decimal64
           |  |  +--ro optical-channel-spectrum-width?     decimal64
           |  |  +--ro min-tx-osnr?                        decimal64
           |  |  +--ro min-rx-osnr?                        decimal64
           |  |  +--ro min-input-power?                    decimal64
           |  |  +--ro max-input-power?                    decimal64
           |  |  +--ro max-chromatic-dispersion?           decimal64
           |  |  +--ro max-differential-group-delay?       decimal64
           |  |  +--ro max-polarization-dependent-loss?    decimal64
           |  +--ro fec
           |  |  +--ro state
           |  |     +--ro fec-coding?               union
           |  |     +--ro coding-overhead?          decimal64
           |  |     +--ro coding-gain?              decimal64
           |  |     +--ro pre-fec-ber-threshold?    decimal64
           |  +--ro penalties
           |  |  +--ro penalty* [parameter-and-unit up-to-boundary]
           |  |     +--ro parameter-and-unit     -> ../state/parameter-and-unit
           |  |     +--ro up-to-boundary         -> ../state/up-to-boundary
           |  |     +--ro state
           |  |        +--ro parameter-and-unit?    oc-opt-term-prop-types:impairment-
type
           |  |        +--ro up-to-boundary?        decimal64
           |  |        +--ro penalty-value?         decimal64
           |  +--ro filter
           |     +--ro state
           |        +--ro pulse-shaping-type?    union
           |        +--ro roll-off?              decimal64
           +--ro optical-channel-config-value-constraints
              +--ro state
                 +--ro min-central-frequency?     oc-opt-types:frequency-type
                 +--ro max-central-frequency?     oc-opt-types:frequency-type
                 +--ro grid-type?                 oc-opt-term-prop-types:grid-type
                 +--ro adjustment-granularity?    oc-opt-term-prop-types:adjustment-
granularity
                 +--ro min-channel-spacing?       decimal64
                 +--ro min-output-power?          decimal64
                 +--ro max-output-power?          decimal64
```

For example, a client may request the details of a given Operational mode by asking for the mode descriptor providing the mode-id (e.g., 100) using standard NETCONF/Yang as seen next:

```
<get>
    <filter type="subtree">
         <operational-modes xmlns="http://example.net/yang/openconfig-terminal-device-properties" xmlns:oc-opt-term-prop-
types="http://example.net/yang/openconfig-terminal-device-property-types" xmlns:oc-opt-types="http://opencon-
fig.net/yang/transport-types">
            <mode-descriptor>
                <mode-id>100</mode-id>
            </mode-descriptor>
        </operational-modes>
    </filter>
</get>
```

The OpenConfig Terminal device is able to reply with the detailed properties of the requested operational mode:



```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <operational-modes xmlns="http://example.net/yang/openconfig-terminal-device-properties">
      <mode-descriptor>
        <mode-id xmlns:oc-opt-term-prop-types="http://example.net/yang/openconfig-terminal-device-property-types" xmlns:oc-opt-
        <state>
          <mode-id>100</mode-id>
          <mode-type xmlns:oc-opt-term-prop-types="http://example.net/yang/openconfig-terminal-device-property-types">oc-opt-te
        </state>
        <explicit-mode>
          <operational-mode-capabilities>
            <state>
              <modulation-format>oc-opt-term-prop-types:MODULATION_FORMAT_QPSK</modulation-format>
              <bit-rate xmlns:oc-opt-types="http://openconfig.net/yang/transport-types">oc-opt-types:TRIB_RATE_40G</bit-rate>
              <baud-rate>20000000000.0</baud-rate>
              <optical-channel-spectrum-width>50.0</optical-channel-spectrum-width>
              <min-tx-osnr>40.0</min-tx-osnr>
              <min-rx-osnr>33.0</min-rx-osnr>
              <min-input-power>0.0</min-input-power>
              <max-input-power>9.0</max-input-power>
              <max-chromatic-dispersion>200.0</max-chromatic-dispersion>
              <max-differential-group-delay>1.0</max-differential-group-delay>
              <max-polarization-dependent-loss>0.12</max-polarization-dependent-loss>
            </state>
            <fec>
              <state>
                <fec-coding>oc-opt-term-prop-types:FEC_HD</fec-coding>
                <coding-overhead>7.0</coding-overhead>
                <coding-gain>8.53</coding-gain>
                <pre-fec-ber-threshold>0.000000000001</pre-fec-ber-threshold>
              </state>
            </fec>
            <penalties>
              <penalty>
                <parameter-and-unit xmlns:oc-opt-term-prop-types="http://example.net/yang/openconfig-terminal-device-property-t
                <up-to-boundary>800.0</up-to-boundary>
                <state>
                  <parameter-and-unit xmlns:oc-opt-term-prop-types="http://example.net/yang/openconfig-terminal-device-property
                  <up-to-boundary>800.0</up-to-boundary>
                  <penalty-value>10.0</penalty-value>
                </state>
              </penalty>
              <penalty>
```

This module extension has been added to the B5G-OPEN SBVT agent component as will be detailed later in the document. The ONOS SDN controller is responsible for mapping the information about Operational modes into its NBI to be consumed and queried by the TAPI Network Orchestrator.

2.2 P4 AND P4 RUNTIME

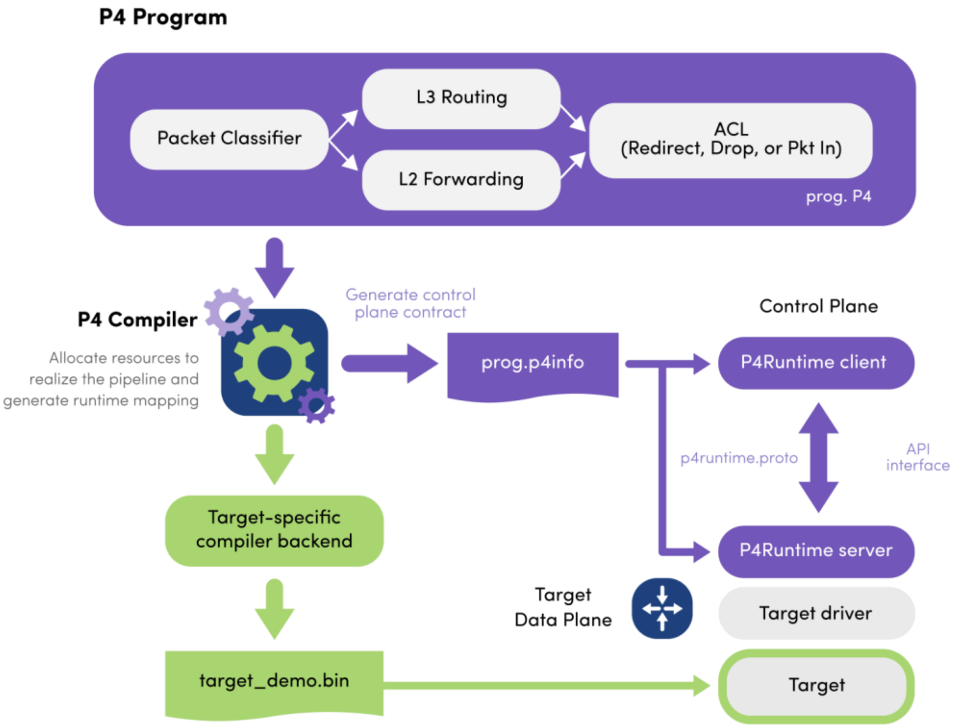


Figure 2-2 P4 development workflow [https://p4.org/]

P4 is a domain-specific language for network devices that defines how data plane devices (switches, NICs, routers, filters, etc.) process packets. P4 [P4] is an open-source project whose goal is to develop and define the tools needed to work with P4 forwarding concepts (e.g., specifications, compiler, interfaces, etc.) in order to enable next-generation SDN. The tools/applications developed in the project are maintained in [P4lang] GitHub repository. Figure 2-2 shows the P4 development workflow required to program and install a P4 pipeline and control it via an SDN controller. More in detail, a P4 program allows to implement a custom pipeline supporting configurable match-action tables and packet headers, metadata extraction, programmable actions, and stateful data structures. The P4 compiler generates an executable file for the target data plane and the runtime mapping metadata to allow the communication among control and data planes.

The P4Runtime API is a remote procedure call (RPC) interface used by the control plane for managing a P4 device where a custom pipeline is installed. After a detailed analysis, P4 language and P4Runtime perfectly fit the requirements for managing and control the packet optical node in the BG5-OPEN project. Indeed, the possibility to perform in-network operations opens the way to new applications and functionalities to be operated at wire-speed. The B5G-OPEN deliverable D3.1 reports an example of optical monitoring parameters included within telemetry packets that are processed by the P4 ASIC for fast recovery.

P4 Runtime commonly uses gRPC [gRPC] as its underlying transport protocol. gRPC uses Protocol Buffers (Protobuf) for serialization, making it efficient and compact. The P4 Runtime Protocol defines a set of standardized operations that the controller can perform on the P4 data plane. These operations typically include:

- *Read*: The controller can request information from the P4 node, such as the current state of tables, counters, and meters.
- *Write*: The controller can send configuration commands to the P4 node to add, modify, or delete table entries, change pipeline stages, update counters, and more.
- *Asynchronous Notifications*: P4 Runtime supports asynchronous notifications, allowing the P4 node to inform the controller of events or changes in real-time, such as packet drops, updates to counters, or hardware-specific events.
- *Fine-Grained Control*: P4 Runtime provides fine-grained control over the configuration of P4 devices. This means the controller can specify detailed rules for packet processing, define how tables should be populated, and manage the pipeline stages to achieve specific networking behaviours.
- *Versioning*: P4 Runtime includes versioning support to ensure compatibility between the controller and the P4 device. This helps prevent issues that might arise when the controller and device have different understandings of the protocol.
- *Authentication and Security*: Like any network protocol, security is a concern. P4 Runtime can incorporate authentication and security mechanisms to ensure that only authorized controllers can configure and control the P4 data plane.
- *Vendor Neutrality*: P4 Runtime aims to be vendor-neutral, allowing controllers to work with a variety of P4-enabled devices from different manufacturers without needing specific adaptations for each vendor's hardware.
- *Dynamic Updates*: P4 Runtime is designed to support dynamic updates of the data plane configuration. This means that the controller can adjust configurations in real-time to adapt to network changes, traffic patterns, and security requirements.

2.3 Transport API (TAPI) for Topology Management and Service Provisioning

The TAPI Optical Network Orchestrator, as an SDN controller, provides Network Topology and Connectivity Request services to a parent SDN Controller or another T-API-able user. It is mainly responsible for the offering of DSR connectivity services between optical transponders that are connected to the ROADMs. The transport protocol used for all operations on the NBI is RESTCONF [RFC8040]. It is an HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, which is the language T-API is defined in. The key YANG models composing the T-API information models are to be based either in the current version 2.1.3 [TR-547], including the following modules:

- *tapi-common.yang ,*
- *tapi-connectivity.yang ,*
- *tapi-dsr.yang*
- *tapi-topology.yang*
- *tapi-connectivity.yang*
- *tapi-path-computation.yang*

2.3.1 Generic Aspects

T-API is based on a context relationship between a server and a client. 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. 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 T-API server *tapi-common:context* includes the following information: The set of Service Interface Points (SIP) exposed to the TAPI client applications representing the available customer-facing access points for requesting network services.

This set must allow Connectivity Service (CS) creation at the DSR Layer, a topology-context which includes one or more top-level Topology objects which are dynamic representations of the network, and connectivity-context which includes the list of Connectivity-Service and Connection objects created within the TAPI Context.

Adopting TAPI allows a standard and mature way to interact with SDN controllers for optical networks, as specified in OOPT MUST [MUST]. In particular, the figure below (Figure 2-3) shows a common representation of an optical network using TAPI terminology and convention.

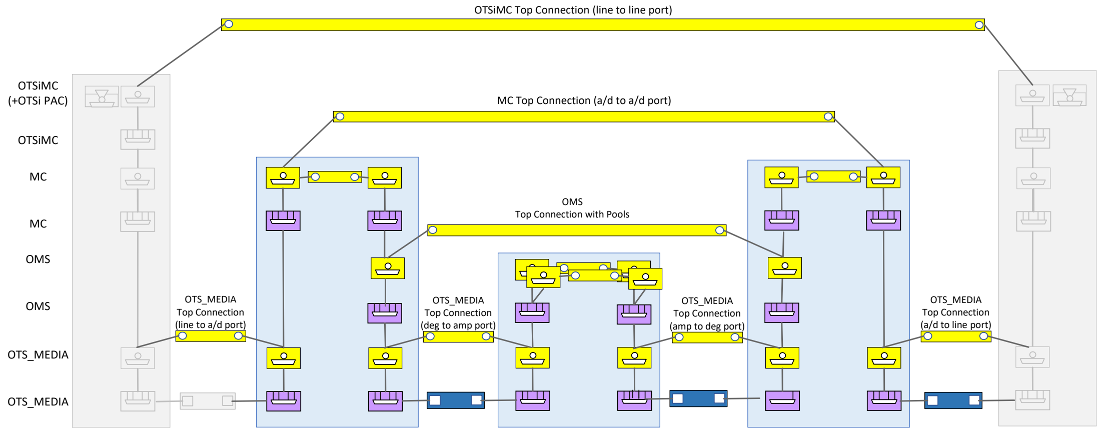


Figure 2-3 TAPI representation of a digital service between pluggables across an optical network

2.3.2 General Remarks

This document does not report or mandate direct access to all data nodes defined by the YANG models. This section captures a minimal set of objects which shall provide full CRUD support according to the TAPI YANG model's specification (e.g., configurable objects should support all operations while non configurable objects shall support retrieval).

API Entry	RESTCONF Operations allowed	Notes
/tapi-common:context Used to retrieve the whole TAPI Context May present scalability issues.	GET	
/tapi-common:context/service-interface-point={uuid} Retrieve the list of Service Interface Points	GET	
/tapi-common:context/tapi-topology:topology-context/topology={uuid} Retrieve a full topology, including the details in terms of links and nodes.	GET	
/tapi-common:context/tapi-topology:topology-context/topology={uuid}/node={uuid} Retrieve a specific node.	GET	
/tapi-common:context/tapi-topology:topology-context/topology={uuid}/node={uuid}?fields=owned-node-edge-point(uuid) Retrieve all NEPs (node edge point) of a given node.	GET	
/tapi-common:context/tapi-topology:topology-context/topology={uuid}/link={uuid}	GET	

Retrieve a specific link.		
/tapi-common:context/tapi-topology:topology-context/topology={uuid}/node={uuid}/owned-node-edge-point={uuid} Retrieve a Specific NEP	GET	
/tapi-common:context/tapi-topology:topology-context/topology={uuid}/node={uuid}/owned-node-edge-point={uuid}/tapi-connectivity:cep-list Retrieve the list of CEPs over a given NEP.	GET	
/tapi-common:context/tapi-topology:topology-context/topology={uuid}/node={uuid}/owned-node-edge-point={uuid}/tapi-connectivity:cep-list/connection-end-point={uuid} Retrieve a specific CEP**.**	GET	
/tapi-common:context/tapi-connectivity:connectivity-context Retrieve the whole connectivity Context. May present scalability issues.	GET/POST	GET for retrieval. POST for Provisioning
/tapi-common:context/tapi-connectivity:connectivity-context/connectivity-service={uuid} Retrieve a specific Connectivity Service	GET	
/tapi-common:context/tapi-connectivity:connectivity-context/connection={uuid} Retrieve a Specific Connection	GET	

2.3.3 Context & Service Interface Points discovery

The TAPI Context and Service Interface Points (SIPs) are the relevant network service information required before any connectivity-service creation operation.

The discovery of this information is intended to be requested periodically and/or on-demand basis, proactively from the TAPI client role, to synchronize the context information. It is possible to retrieve the whole set of SIPs, or the specific details of a given SIP. As it can be seen in the example below, the SIPs are defined for the Digital Signal Rate (DSR) network layer, so a client such as the B5G-ONP can ask a DSR for K Gb/s between two of such SIPs.

2.3.3.1 Example

GET `http://localhost:4900/restconf/data/tapi-common:context/service-interface-point`"

GET `http://localhost:4900/restconf/data/tapi-common:context/service-interface-point=b71052b2-fc83-5cb5-b39e-9fd226a2042d`

Response:

```
{
    "tapi-common:service-interface-point": [
        {
            "administrative-state": "UNLOCKED",
            "direction": "BIDIRECTIONAL",
            "layer-protocol-name": "DSR",
            "name": [
                {
                    "value": "Transp-4_port_11003-floating",
                    "value-name": "cttc.ols.port"
                },
                {
                    "value": "4294956292",
                    "value-name": "cttc.ols.ifid"
                },
                {
                    "value": "10.100.101.24:4294956292",
                    "value-name": "cttc.gmpls.ifid"
                },
                {
                    "value": "Transp-4_port_11003-sip",
                    "value-name": "service-port-name"
                },
                {
                    "value": "Transp-4_port_4294956292-sip",
                    "value-name": "local-name"
                }
            ],
            "operational-state": "ENABLED",
            "supported-layer-protocol-qualifier": [
                "tapi-dsr:DIGITAL_SIGNAL_TYPE_UNSPECIFIED"
            ],
            "uuid": "b71052b2-fc83-5cb5-b39e-9fd226a2042d"
        },
```

2.3.4 Topology Discovery

The TAPI Topology is the relevant network logical representation information required for inventory, traffic-engineering, or provisioning purposes. The discovery of this information is intended to be requested periodically and/or on-demand basis, proactively from the TAPI client role, to synchronize the context information.

topology	**/tapi-common:context/tapi-topology:topology-context/topology**
Attribute	**Allowed Values/Format**
uuid	As per RFC 4122
layer-protocol-name	Leaf-List including the present Layer Protocol Names in the topology. They MUST be elements from {"DSR", "DIGITAL_OTN", "PHOTONIC_MEDIA"}
link	List of {link} objects, as defined in [TR-547]
node	List of {node} objects as defined in [TR-547]

2.3.4.1 Example

GET http://localhost:4900/restconf/data/tapi-common:context/tapi-topology:topology-context/topology

GET http://localhost:4900/restconf/data/tapi-common:context/tapi-topology:topology-context/topology=d457ae3f-56f0-5abe-8d5b-1139dc46e9df

Response:

```
{
   "tapi-topology:topology" : [
      {
         "layer-protocol-name" : [
            "PHOTONIC_MEDIA",
            "DSR"
         ],
         "link" : [
            {
               "cost-characteristic" : [
                  {
                     "cost-name" : "te-metric",
                     "cost-value" : "1.000000"
                  }
               ],
               "direction" : "UNIDIRECTIONAL",
               "layer-protocol-name" : [
                  "PHOTONIC_MEDIA"
               ],
               "name" : [
                  {
                     "value" : "link_10.100.101.13:31-10.100.101.14:32",
                     "value-name" : "local-name"
```

(...)

2.3.5 Retrieve a Link

This all aims at retrieving a Link's characteristics provided its uuid in the request (within the topology that contains such link).

2.3.5.1 Example

GET `http://localhost:4900/restconf/data/tapi-common:context/tapi-topology:topology-context/topology=d457ae3f-56f0-5abe-8d5b-1139dc46e9df/link=ce850f8e-b521-517a-8074-60546bde5b92`

Response :

```
{
    "cost-characteristic": [
        {
            "cost-name": "te-metric",
            "cost-value": "1.000000"
        }
    ],
    "direction": "UNIDIRECTIONAL",
    "layer-protocol-name": [
        "PHOTONIC_MEDIA"
    ],
    "name": [
        {
            "value": "netconf:10.100.101.14:2022",
            "value-name": "dst.device"
        },
        {
            "value": "32",
            "value-name": "dst.port"
        },
        …
    ],
    "node-edge-point": [
        {
            "node-edge-point-uuid": "9dced988-a521-5a00-a9ee-2c80756df9ad",
            "node-uuid": "1479f1cf-f22f-51e6-a531-66fa98af719b",
            "topology-uuid": "d457ae3f-56f0-5abe-8d5b-1139dc46e9df"
        },
        {
            "node-edge-point-uuid": "4182743d-0b0f-51ea-b0db-3636b855d9f7",
            "node-uuid": "92679803-da09-50a5-baef-5f41eafe0405",
            "topology-uuid": "d457ae3f-56f0-5abe-8d5b-1139dc46e9df"
        }
    ],
    "uuid": "ce850f8e-b521-517a-8074-60546bde5b92"
}
```

2.3.6 Retrieve a Node

This all aims at retrieving a Node's characteristics provided its UUID.

2.3.6.1 Example

GET http://localhost:4900/restconf/data/tapi-common:context/tapi-topology:topology-context/topology=d457ae3f-56f0-5abe-8d5b-1139dc46e9df/node=1479f1cf-f22f-51e6-a531-66fa98af719b

Response:

```
{
    "name": [
        {
            "value": "ROADM-3",
            "value-name": "clli"
        },
       ...    ],
    "owned-node-edge-point": [
        {
            "direction": "SOURCE",
            "layer-protocol-name": "PHOTONIC_MEDIA",
            "name": [
                {
                    "value": "ROADM-3_port_21",
                    "value-name": "local-name"
                },
                ...
```

2.3.7 Creating a Service

To create a connectivity service, the client sends a POST providing the UUID and the Attributes

Example:

POST http://127.0.0.1:4900/restconf/data/tapi-common:context/tapi-connectivity:connectivity-context

```
{
   "tapi-connectivity:connectivity-service" : [
      {
         "connectivity-constraint" : {
            "connectivity-direction" : "BIDIRECTIONAL",
            "requested-capacity" : {
               "total-size" : {
                  "unit" : "GBPS",
                  "value" : 400
               }
            }
         },
         "direction" : "BIDIRECTIONAL",
         "end-point" : [
            {
               "direction:" : "BIDIRECTIONAL",
               "layer-protocol-name" : "DSR",
               "layer-protocol-qualifier" : "LAYER_PROTOCOL_QUALIFIER_UNSPECIFIED",
               "local-id" : "8eb843ff-ea74-5b5d-a8c0-661673b6c823",
               "service-interface-point" : {
                  "service-interface-point-uuid" : "8eb843ff-ea74-5b5d-a8c0-661673b6c823"
               }
            },
            {
               "direction:" : "BIDIRECTIONAL",
               "layer-protocol-name" : "DSR",
               "layer-protocol-qualifier" : "LAYER_PROTOCOL_QUALIFIER_UNSPECIFIED",
               "local-id" : "7c09f268-6b42-5c30-b02a-b8b54a0ff7c5",
               "service-interface-point" : {
                  "service-interface-point-uuid" : "7c09f268-6b42-5c30-b02a-b8b54a0ff7c5"
               }
            }
         ],
         "layer-protocol-name" : "DSR",
         "layer-protocol-qualifier" : "LAYER_PROTOCOL_QUALIFIER_UNSPECIFIED",
         "requested-capacity" : {
            "total-size" : {
               "unit" : "GBPS",
               "value" : 400
            }
         },
         "route-objective-function" : "26000",
         "uuid" : "d71187bc-2947-11e8-b467-0ed5f89f718b"
      }
   ]
}
```

```
Response: 201 Created
Location
```

2.3.8 Retrieval of Connectivity Service and Connections

These calls are used to retrieve the data of existing connectivity services and connections.

Examples:

```
GET http://localhost:4900/restconf/data/tapi-common:context/tapi-connectivity:connectivity-context/connectivity-service
```

2.3.9 Service Deletion

To proceed with a service deletion, the client sends a delete request specifying the UUID of the Connectivity service to delete.

Example:

```
DELETE http://127.0.0.1:4900/restconf/data/tapi-common:context/tapi-connectivity:connectivity-context/connectivity-service=aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa
```

2.4 Path Computation (TAPI) for Externalized Path Computation

B5G OPEN has adopted the TAPI (Transport API) architecture [TR-547], and as such, the OPCE element is a module that can assist the TAPI Optical Network Orchestrator for computing the optical path e.g., in the provisioning process. More than one OPCE implementations can be used, by different partners, e.g., during the EUCNC2023 and ECOC2023 the Multi-Band Path Computation Engine (MB-PCE) was used as OPCE. In addition, other implementations can be realised which can be included inside the B5G-ONP, or external to it, with different capabilities.

The interaction between the OPCE and the TAPI Optical Network Orchestrator follows TAPI standards (TAPI version 2). Figure 2-4 shows an exemplified sequence of messages in a typical interaction based on [TR-547].

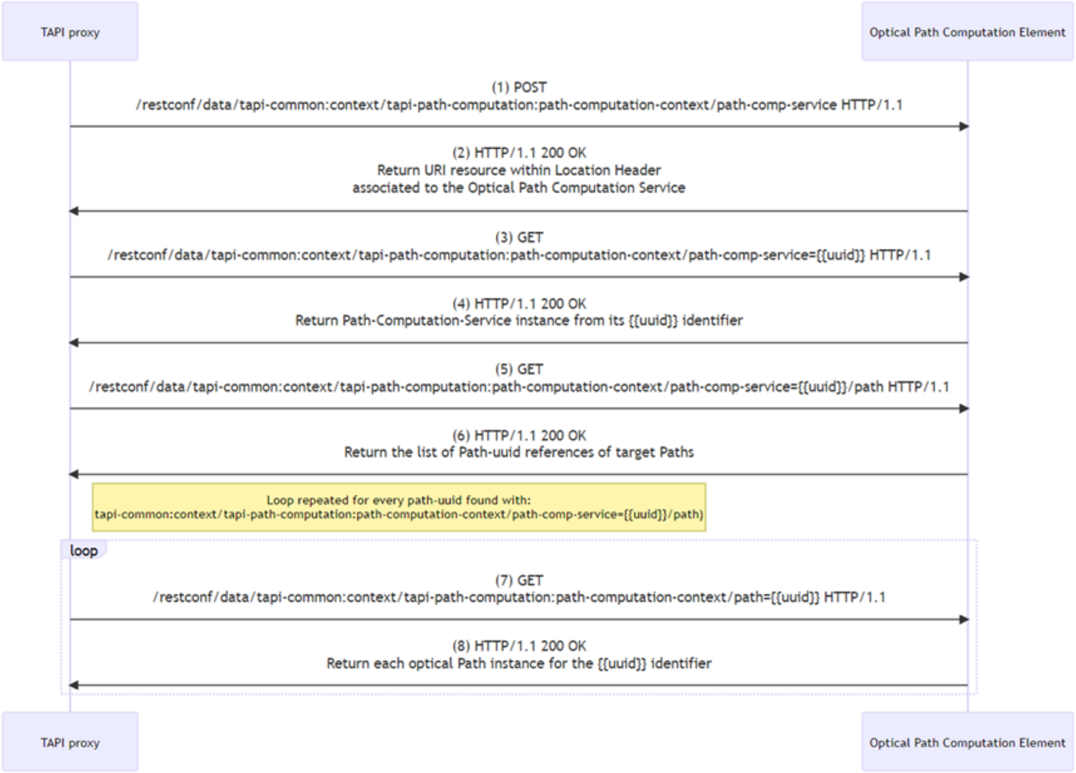

TAPI proxy
Optical Path Computation Element
(1) POST
/restconf/data/tapi-common:context/tapi-path-computation:path-computation-context/path-comp-service HTTP/1.1
(2) HTTP/1.1 200 OK
Return URI resource within Location Header
associated to the Optical Path Computation Service
(3) GET
/restconf/data/tapi-common:context/tapi-path-computation:path-computation-context/path-comp-service={{uuid}} HTTP/1.1
(4) HTTP/1.1 200 OK
Return Path-Computation-Service instance from its {{uuid}} identifier
(5) GET
/restconf/data/tapi-common:context/tapi-path-computation:path-computation-context/path-comp-service={{uuid}}/path HTTP/1.1
(6) HTTP/1.1 200 OK
Return the list of Path-uuid references of target Paths
Loop repeated for every path-uuid found with:
tapi-common:context/tapi-path-computation:path-computation-context/path-comp-service={{uuid}}/path)
loop
(7) GET
/restconf/data/tapi-common:context/tapi-path-computation:path-computation-context/path={{uuid}} HTTP/1.1
(8) HTTP/1.1 200 OK
Return each optical Path instance for the {{uuid}} identifier
TAPI proxy
Optical Path Computation Element

Figure 2-4 Exemplified TAPI Optical Network Orchestrator -.OPCE interaction.

Alternatively, to ease integration between components, we have opted for the RPC version, as shown below. The TAPI Network Orchestrator requests a Path Computation:

URI http://127.0.0.1:4901/restconf/operations/tapi-path-computation:compute-p-2-p-path/

Input:

```
{
   "end-point" : [
      {
         "capacity" : {
            "total-size" : {
               "unit" : "GBPS",
               "value" : 50
            }
         },
         "direction" : "BIDIRECTIONAL",
         "layer-protocol-name" : "DSR",
         "layer-protocol-qualifier" : "tapi-dsr:DIGITAL_SIGNAL_TYPE",
         "local-id" : "Z",
         "service-interface-point" : {
            "service-interface-point-uuid" : "8eb843ff-ea74-5b5d-a8c0-661673b6c823"
         }
      },
      {
         "capacity" : {
            "total-size" : {
               "unit" : "GBPS",
               "value" : 50
            }
         },
         "direction" : "UNIDIRECTIONAL",
         "layer-protocol-name" : "DSR",
         "layer-protocol-qualifier" : "tapi-dsr:DIGITAL_SIGNAL_TYPE",
         "service-interface-point" : {
            "service-interface-point-uuid" : "7c09f268-6b42-5c30-b02a-b8b54a0ff7c5"
         }
      }
   ],
   "layer-protocol-name" : "DSR",
   "objective-function" : {
      "local-id" : "26000"
   }
}
```

Output:

```
{
   "tapi-path-computation:output" : {
      "service" : {
         "end-point" : [
            {
               "capacity" : {
                  "total-size" : {
                     "unit" : "GBPS",
                     "value" : 50
                  }
               },
               "direction" : "BIDIRECTIONAL",
               "layer-protocol-name" : "DSR",
               "layer-protocol-qualifier" : "tapi-dsr:DIGITAL_SIGNAL_TYPE",
               "local-id" : "Z",
               "service-interface-point" : {
                  "service-interface-point-uuid" : "8eb843ff-ea74-5b5d-a8c0-661673b6c823"
               }
            },
            {
               "capacity" : {
                  "total-size" : {
                     "unit" : "GBPS",
                     "value" : 50
                  }
               },
               "direction" : "UNIDIRECTIONAL",
               "layer-protocol-name" : "DSR",
               "layer-protocol-qualifier" : "tapi-dsr:DIGITAL_SIGNAL_TYPE",
               "service-interface-point" : {
                  "service-interface-point-uuid" : "7c09f268-6b42-5c30-b02a-b8b54a0ff7c5"
               }
            }
         ],
         "layer-protocol-name" : "DSR",
         "layer-protocol-qualifier" : "tapi-dsr:DIGITAL_SIGNAL_TYPE",
         "path" : {
            "link" : [
               {
                  "link-uuid" : "5b9faf19-a756-57da-8482-44dcf29d3acd",
                  "topology-uuid" : "d457ae3f-56f0-5abe-8d5b-1139dc46e9df"
               },
               {
                  "link-uuid" : "0c519695-ab02-5ffd-b0d4-0a5cc005bad3",
                  "topology-uuid" : "d457ae3f-56f0-5abe-8d5b-1139dc46e9df"
               },
               {
                  "link-uuid" : "8c2a8c52-10c8-508f-979f-7e543437347c",
                  "topology-uuid" : "d457ae3f-56f0-5abe-8d5b-1139dc46e9df"
               }
            ],
            "mc-pool" : {
               "available-spectrum" : [
                  {
                     "frequency-constraint" : {
                        "adjustment-granularity" : "G_6_25GHZ",
                        "grid-type" : "FLEX"
                     },
                     "lower-frequency" : 184525000,
                     "upper-frequency" : 184625000
                  }
               ]
            },
            "path-uuid" : "0e47cccf-2fb0-48c5-a3be-a921c3df65db"
         }
      }
   }
}
```

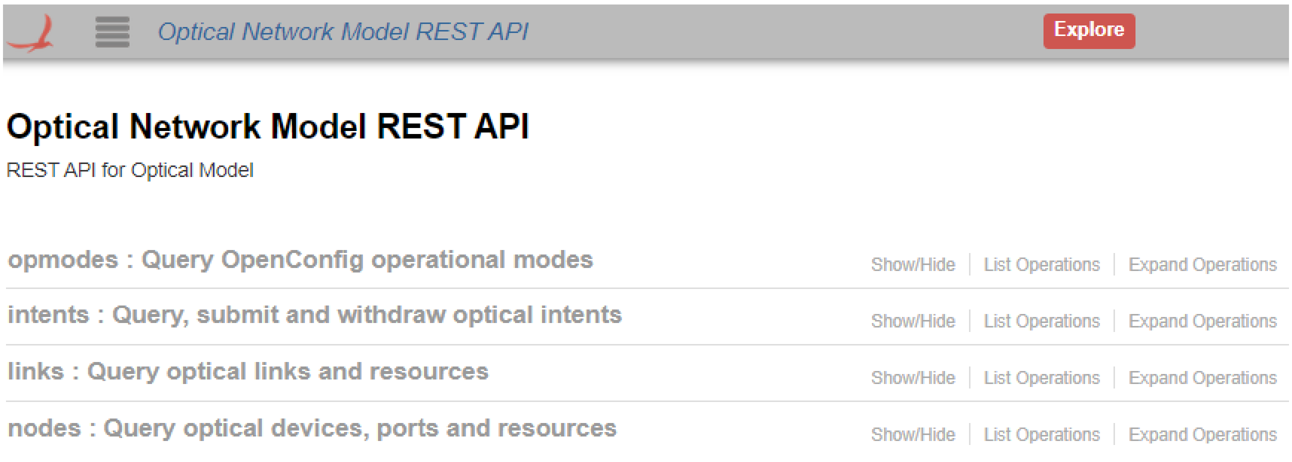
2.5 ONOS REST-BASED APIS

The ONOS controller [ONOS, ONOS-metro] includes a wide set of northbound REST APIs providing GET/POST/DELETE methods towards the network [ONOSREST]. For example, GET methods can be used to retrieve information about the network topology or about the current configuration of ONOS applications (NetApps) running on the controller. Similarly, POST and DELETE methods can be used to interact with the network devices and the network applications, e.g., sending new configuration to the network devices or modifying actual values of NetApps parameters.

These interfaces have been extended to integrate the optical controller within the B5G-OPEN control plane, specifically the interfaces will be consumed by TAPI orchestrator. Such interfaces have been extended for enabling retrieval of optical resource utilization (e.g., supported and available frequency slots) and optical physical parameters to be used by upper control plane components.

The Optical Network Model REST APIs is a NetApp included in the master distribution on the ONOS controller that, before B5G-OPEN, was only providing a GET/POST/DELETE methods for the management of optical intents. In particular, the GET method is used to retrieve the list of established optical intents, the POST method is used to submit an intent request to the network, and the DELETE method is used to withdraw an intent.

Such NetApp has been extended and now provides four different menus for the management of: optical intents, optical links, optical nodes and operational modes. Such extended version of the NetApp will be released to the ONOS community before the end of the B5G-OPEN project.



Optical Network Model REST API

REST API for Optical Model

opmodes : Query OpenConfig operational modes — Show/Hide | List Operations | Expand Operations

intents : Query, submit and withdraw optical intents — Show/Hide | List Operations | Expand Operations

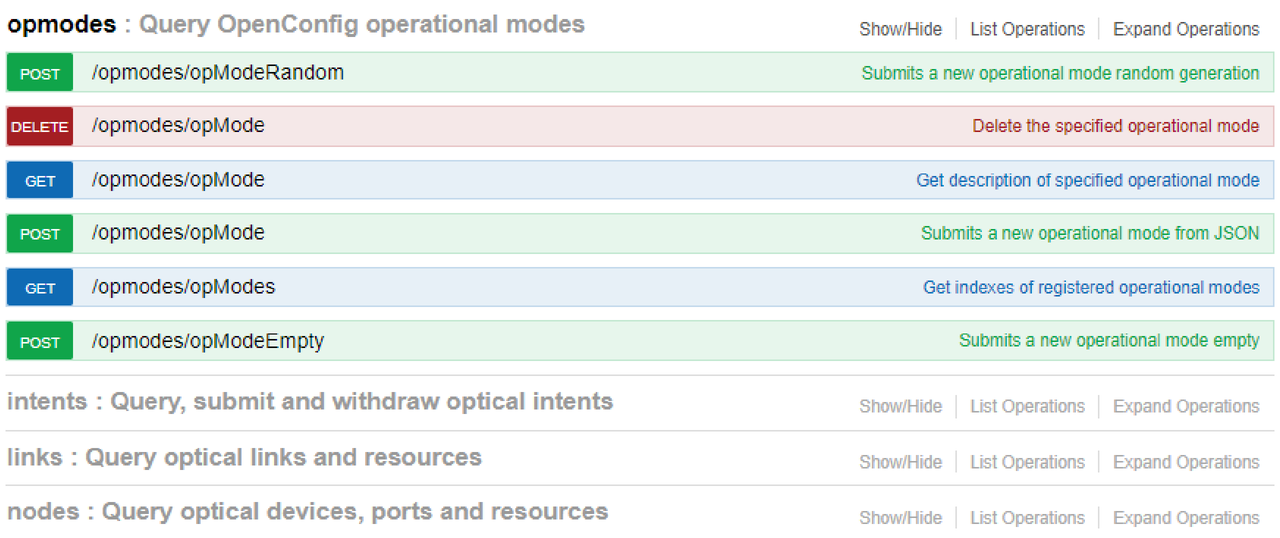
links : Query optical links and resources — Show/Hide | List Operations | Expand Operations

nodes : Query optical devices, ports and resources — Show/Hide | List Operations | Expand Operations

Figure 2-5: main menu of NetApp Optical Network Model REST APIs

2.5.1 Operational Modes

The operational modes menu has been implemented to export the available operational modes toward the T-API orchestrator. At the same time, the ONOS controller has been extended in the SBI to properly acquire the supported operational modes, and the related description, from the connected devices.



opmodes : Query OpenConfig operational modes

Show/Hide | List Operations | Expand Operations

POST	/opmodes/opModeRandom	Submits a new operational mode random generation
DELETE	/opmodes/opMode	Delete the specified operational mode
GET	/opmodes/opMode	Get description of specified operational mode
POST	/opmodes/opMode	Submits a new operational mode from JSON
GET	/opmodes/opModes	Get indexes of registered operational modes
POST	/opmodes/opModeEmpty	Submits a new operational mode empty

intents : Query, submit and withdraw optical intents

Show/Hide | List Operations | Expand Operations

links : Query optical links and resources

Show/Hide | List Operations | Expand Operations

nodes : Query optical devices, ports and resources

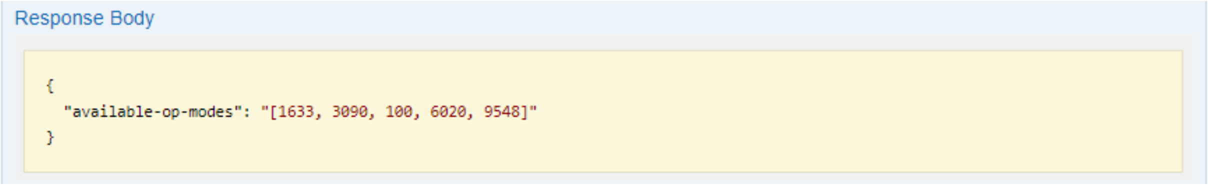
Show/Hide | List Operations | Expand Operations

Figure 2-6: Operational Modes REST APIs.

Among the implemented methods the ones that are relevant to the network operation are the two GET methods, while other methods have been implemented to facilitate debugging operation during development.

2.5.1.1 Operational Modes – GET details

This method provides the list of operational modes supported in the network.



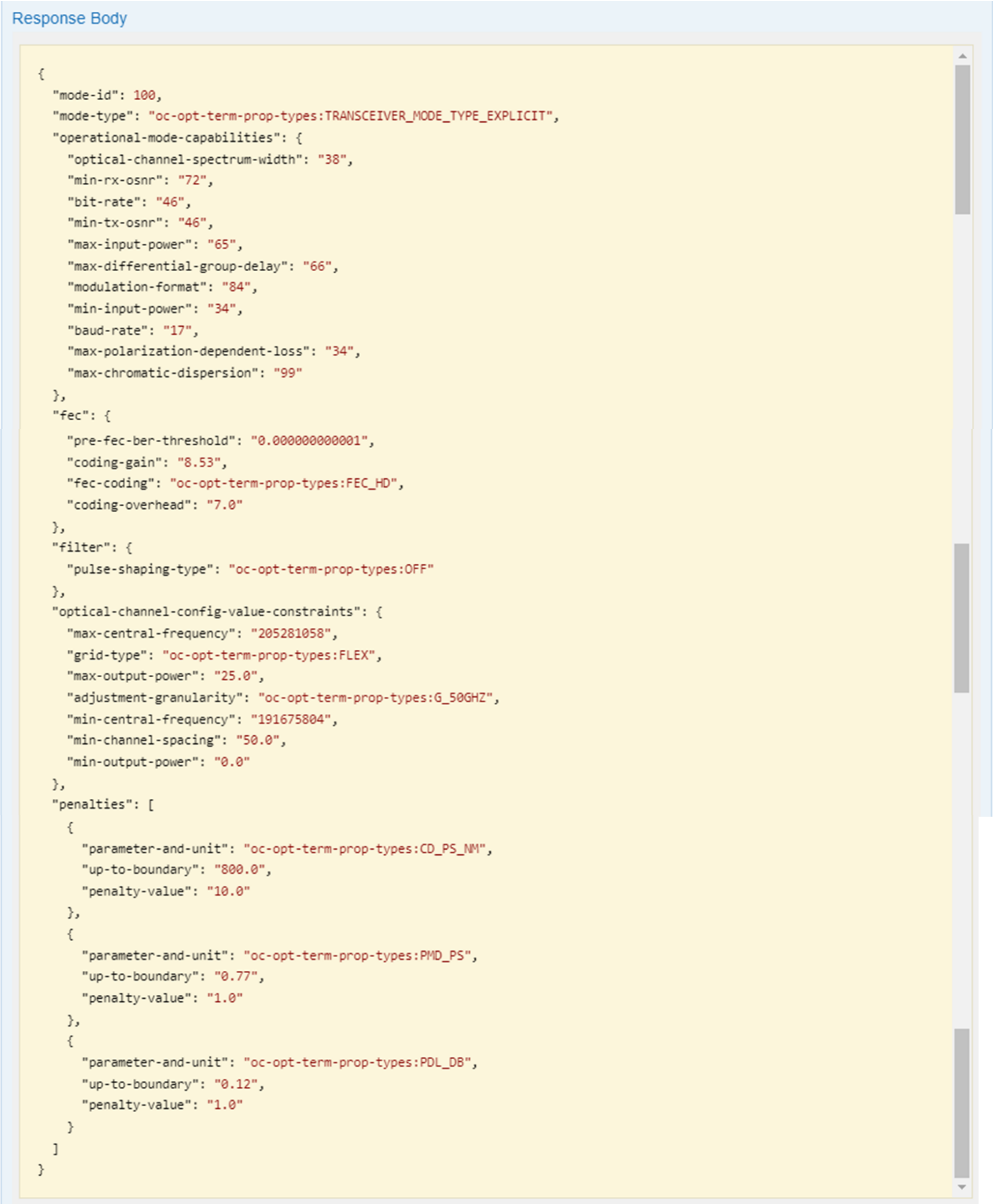
Response Body

```
{
  "available-op-modes": "[1633, 3090, 100, 6020, 9548]"
}
```

Figure 2-7: Response to GET opmodes

2.5.1.2 Operational Mode – GET details

This method provides all the details regarding the operational mode that is specified as input.

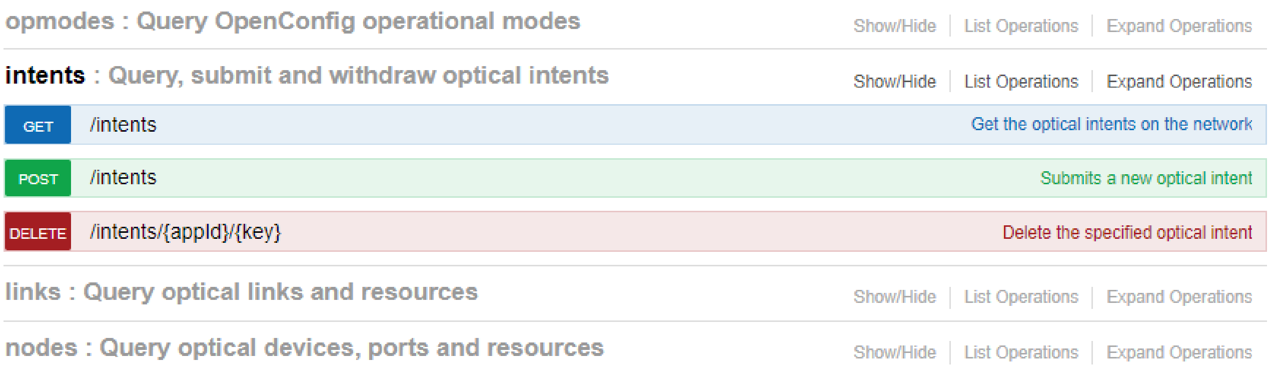


```
{
  "mode-id": 100,
  "mode-type": "oc-opt-term-prop-types:TRANSCEIVER_MODE_TYPE_EXPLICIT",
  "operational-mode-capabilities": {
    "optical-channel-spectrum-width": "38",
    "min-rx-osnr": "72",
    "bit-rate": "46",
    "min-tx-osnr": "46",
    "max-input-power": "65",
    "max-differential-group-delay": "66",
    "modulation-format": "84",
    "min-input-power": "34",
    "baud-rate": "17",
    "max-polarization-dependent-loss": "34",
    "max-chromatic-dispersion": "99"
  },
  "fec": {
    "pre-fec-ber-threshold": "0.000000000001",
    "coding-gain": "8.53",
    "fec-coding": "oc-opt-term-prop-types:FEC_HD",
    "coding-overhead": "7.0"
  },
  "filter": {
    "pulse-shaping-type": "oc-opt-term-prop-types:OFF"
  },
  "optical-channel-config-value-constraints": {
    "max-central-frequency": "205281058",
    "grid-type": "oc-opt-term-prop-types:FLEX",
    "max-output-power": "25.0",
    "adjustment-granularity": "oc-opt-term-prop-types:G_50GHZ",
    "min-central-frequency": "191675804",
    "min-channel-spacing": "50.0",
    "min-output-power": "0.0"
  },
  "penalties": [
    {
      "parameter-and-unit": "oc-opt-term-prop-types:CD_PS_NM",
      "up-to-boundary": "800.0",
      "penalty-value": "10.0"
    },
    {
      "parameter-and-unit": "oc-opt-term-prop-types:PMD_PS",
      "up-to-boundary": "0.77",
      "penalty-value": "1.0"
    },
    {
      "parameter-and-unit": "oc-opt-term-prop-types:PDL_DB",
      "up-to-boundary": "0.12",
      "penalty-value": "1.0"
    }
  ]
}
```

The details are the ones loaded from the OpenConfig model of the device at the time of device discovery and are compliant with the OpenConfig version 0.1.0.

2.5.2 Intents

The intents menu was already available before the starting of the project (it has been developed during the METRO-HAUL project) [ONOS-metro]. However, some further extensions to these interfaces have been introduced during B5G-OPEN.



opmodes : Query OpenConfig operational modes — Show/Hide | List Operations | Expand Operations

intents : Query, submit and withdraw optical intents — Show/Hide | List Operations | Expand Operations

GET	/intents	Get the optical intents on the network
POST	/intents	Submits a new optical intent
DELETE	/intents/{appId}/{key}	Delete the specified optical intent

links : Query optical links and resources — Show/Hide | List Operations | Expand Operations

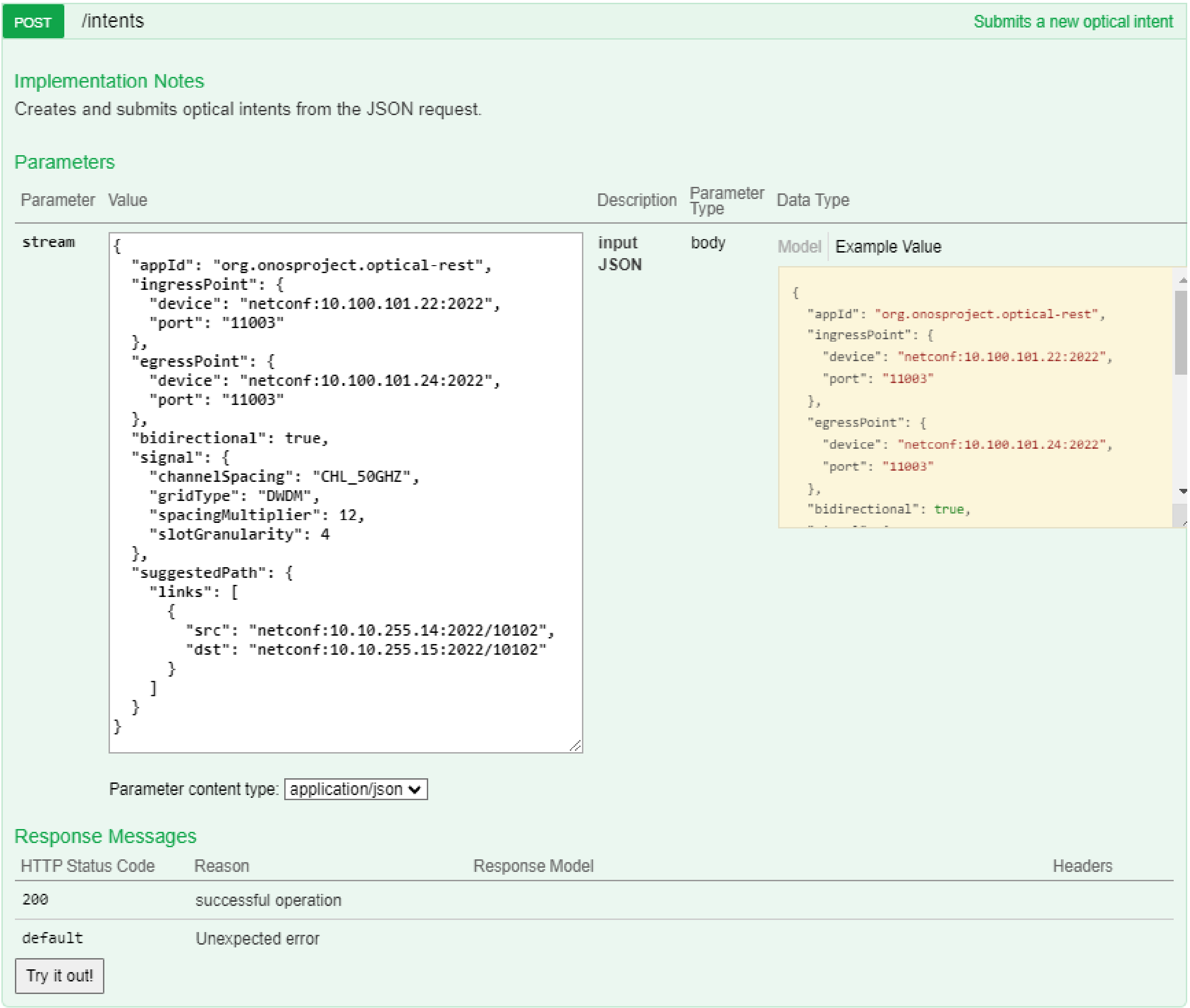
nodes : Query optical devices, ports and resources — Show/Hide | List Operations | Expand Operations

Figure 2-8: intents REST APIs.

2.5.2.1 Intents – POST details

The POST method is used to submit an intent request to the network. It allows to specify the following parameters:

- **End-points of the intent:** these are specified through the device id and the port id. The specified device should be of type TERMINAL_DEVICE (i.e., a transponder), while the specified ports should be of type OCH (i.e., the line port of a transponder) or of type ODU_CLT (i.e., the client port of a transponder). During B5g-OPEN, an extension to this point has been introduced for enabling the activation of optical intents initiating/terminating at OCH ports of a ROADM devices.
- **Bidirectional:** it is a boolean flag that is set to true if the required intent has to be established bidirectionally.
- **Signal:** this field allows to specify the optical channel to be used for the intent. During B5G-OPEN this field has been extended to support the specification of channels using flex-grid, also considering multi-bands (i.e., L, C and S bands).
- **Suggested path:** this field allows to specify the path that should be followed by the intent. This is specified as a sequence of links. In case of bidirectional intents, the reverted path is applied in the opposite direction.



POST /intents Submits a new optical intent

Implementation Notes

Creates and submits optical intents from the JSON request.

Parameters

Parameter	Value	Description	Parameter Type	Data Type
stream	(see below)	input JSON	body	Model Example Value

```
{
  "appId": "org.onosproject.optical-rest",
  "ingressPoint": {
    "device": "netconf:10.100.101.22:2022",
    "port": "11003"
  },
  "egressPoint": {
    "device": "netconf:10.100.101.24:2022",
    "port": "11003"
  },
  "bidirectional": true,
  "signal": {
    "channelSpacing": "CHL_50GHZ",
    "gridType": "DWDM",
    "spacingMultiplier": 12,
    "slotGranularity": 4
  },
  "suggestedPath": {
    "links": [
      {
        "src": "netconf:10.10.255.14:2022/10102",
        "dst": "netconf:10.10.255.15:2022/10102"
      }
    ]
  }
}
```

Example Value:

```
{
  "appId": "org.onosproject.optical-rest",
  "ingressPoint": {
    "device": "netconf:10.100.101.22:2022",
    "port": "11003"
  },
  "egressPoint": {
    "device": "netconf:10.100.101.24:2022",
    "port": "11003"
  },
  "bidirectional": true,
```

Parameter content type: application/json

Response Messages

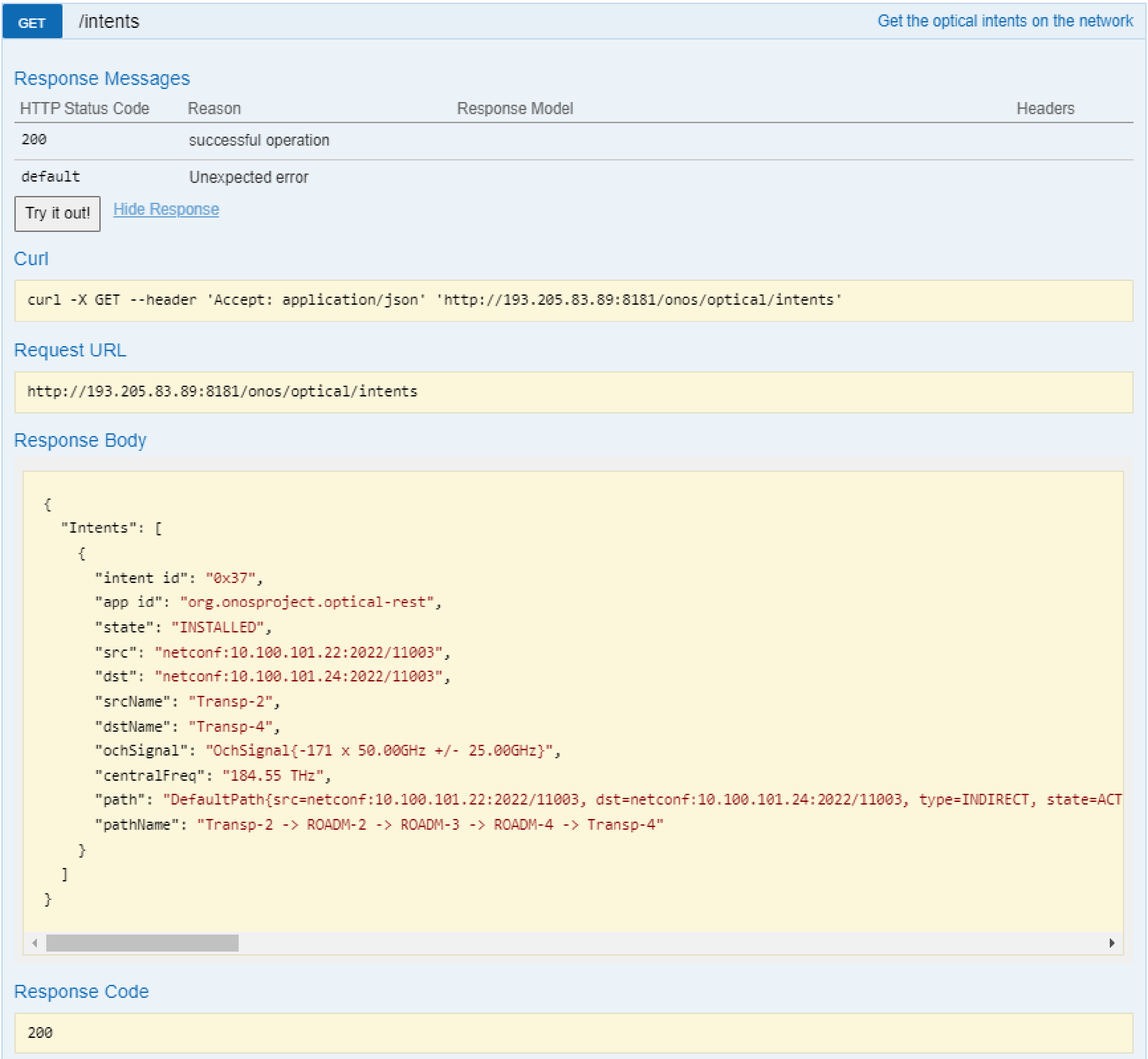
HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

Figure 2-9 Example of POST intent method.

2.5.2.2 Intents – GET details

The GET method is used to retrieve information regarding the intent requests that have been submitted by the controller. An example of the response reply is provided in the following figure.

Among the information that are provide as a response it important to mention the state that is related to the Finite State Machine modelling the intents in ONOS. It can assume the following self-explaining values: REQUESTED, INSTALLING, INSTALLED, FAILED, WITHDRAWING, WITHDRAWN, CORRUPTED. When an intent is in the state FAILED, the controller keep periodically trying to recover the intent.



Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

Curl

```
curl -X GET --header 'Accept: application/json' 'http://193.205.83.89:8181/onos/optical/intents'
```

Request URL

```
http://193.205.83.89:8181/onos/optical/intents
```

Response Body

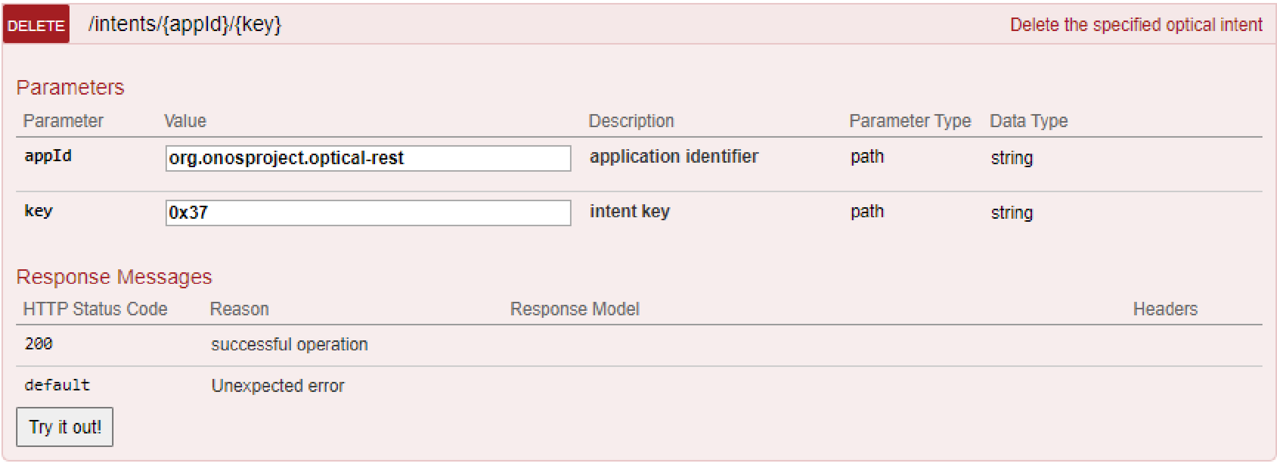
```
{
  "Intents": [
    {
      "intent id": "0x37",
      "app id": "org.onosproject.optical-rest",
      "state": "INSTALLED",
      "src": "netconf:10.100.101.22:2022/11003",
      "dst": "netconf:10.100.101.24:2022/11003",
      "srcName": "Transp-2",
      "dstName": "Transp-4",
      "ochSignal": "OchSignal{-171 x 50.00GHz +/- 25.00GHz}",
      "centralFreq": "184.55 THz",
      "path": "DefaultPath{src=netconf:10.100.101.22:2022/11003, dst=netconf:10.100.101.24:2022/11003, type=INDIRECT, state=ACT
      "pathName": "Transp-2 -> ROADM-2 -> ROADM-3 -> ROADM-4 -> Transp-4"
    }
  ]
}
```

Response Code

```
200
```

2.5.2.3 *Intents – DELETE details*

The GET method is used to withdraw an intent. It requires to specify the intent id and the application that originally installed the intent.



Parameters

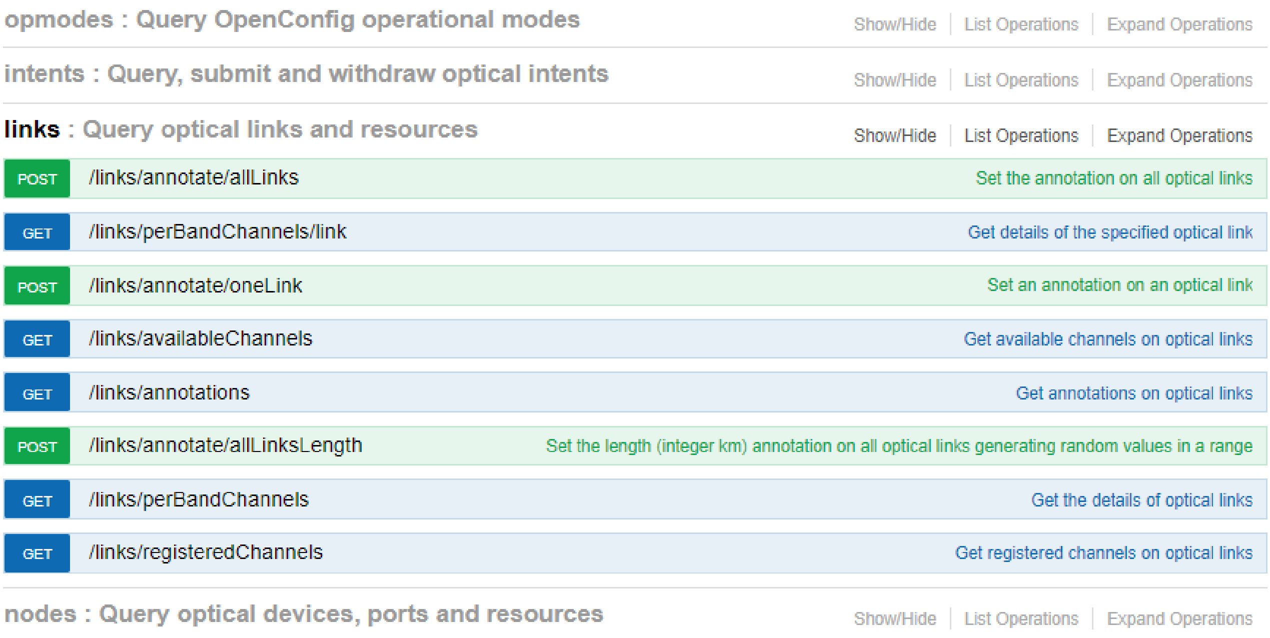
Parameter	Value	Description	Parameter Type	Data Type
appId	org.onosproject.optical-rest	application identifier	path	string
key	0x37	intent key	path	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	successful operation		
default	Unexpected error		

2.5.3 Links

The links menu provides information regarding the characteristics of fibre links (e.g., fibre length) that can be relevant for the computation of physical impairments executed by upper layers tools. Also, this interface provides details regarding the frequency slots that are supported and available/unavailable by each fibre link.



opmodes : Query OpenConfig operational modes — Show/Hide | List Operations | Expand Operations

intents : Query, submit and withdraw optical intents — Show/Hide | List Operations | Expand Operations

links : Query optical links and resources — Show/Hide | List Operations | Expand Operations

POST /links/annotate/allLinks — Set the annotation on all optical links

GET /links/perBandChannels/link — Get details of the specified optical link

POST /links/annotate/oneLink — Set an annotation on an optical link

GET /links/availableChannels — Get available channels on optical links

GET /links/annotations — Get annotations on optical links

POST /links/annotate/allLinksLength — Set the length (integer km) annotation on all optical links generating random values in a range

GET /links/perBandChannels — Get the details of optical links

GET /links/registeredChannels — Get registered channels on optical links

nodes : Query optical devices, ports and resources — Show/Hide | List Operations | Expand Operations

Figure 2-10: links REST APIs.

Among the implemented methods, the relevant one for the network operation is the GET methods: /links/perBandChannels. Other methods have been implemented to improve presentation and facilitate debugging operation during development.

2.5.3.1 Links perBandChannels – GET details

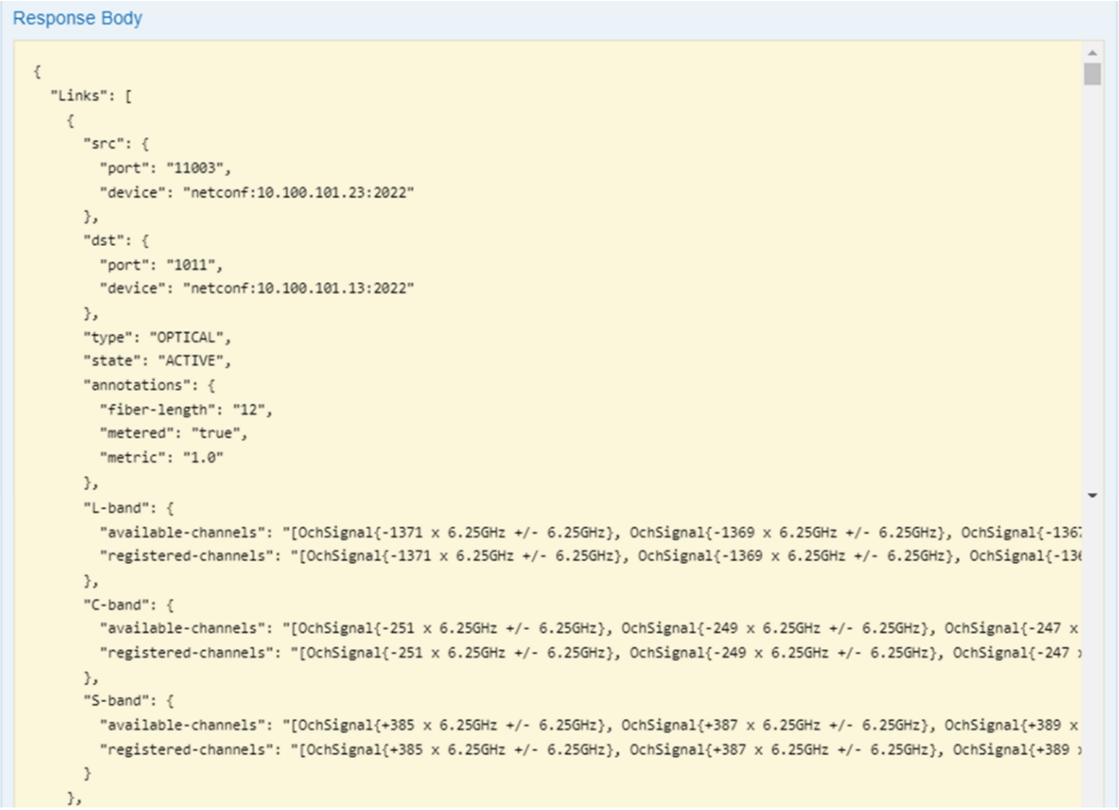
This interface provides the basic topological references for each link, such as the end-points, the type and the current state. Moreover, it provides the list of annotations where all the details relevant for the physical impairment computation will be annotated. In the current version only the length of the fibre is exposed. Finally, the interface provides, for each band the list of registered and currently available channels.

In the lists of registered and available channels, each channel is described using the ONOS encoding on the flexible grid as specified in G.694.1 (10/20) published by ITU-T. Thus, for example the `OchSignal{-251 x 6.25GHz +/- 6.25GHz}` refers to a frequency slot:

central frequency: 193100 – 251*6.25 = 193100 – 1568.75 = 191531.25 GHz

width: 6.25 * 2 = 12.5 GHz

The list of available channels is updated every time a new lightpath is established on such link.



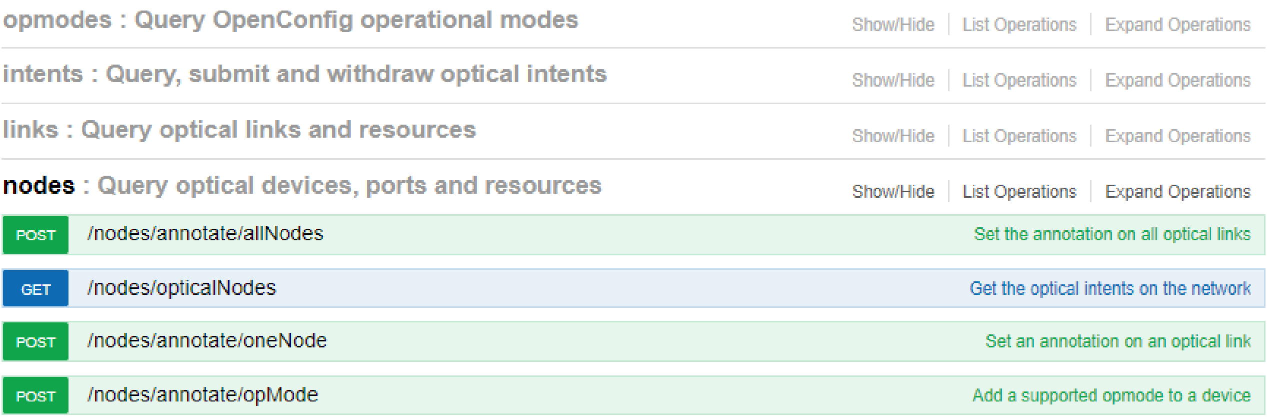
Response Body

```
{
  "Links": [
    {
      "src": {
        "port": "11003",
        "device": "netconf:10.100.101.23:2022"
      },
      "dst": {
        "port": "1011",
        "device": "netconf:10.100.101.13:2022"
      },
      "type": "OPTICAL",
      "state": "ACTIVE",
      "annotations": {
        "fiber-length": "12",
        "metered": "true",
        "metric": "1.0"
      },
      "L-band": {
        "available-channels": "[OchSignal{-1371 x 6.25GHz +/- 6.25GHz}, OchSignal{-1369 x 6.25GHz +/- 6.25GHz}, OchSignal{-136
        "registered-channels": "[OchSignal{-1371 x 6.25GHz +/- 6.25GHz}, OchSignal{-1369 x 6.25GHz +/- 6.25GHz}, OchSignal{-13
      },
      "C-band": {
        "available-channels": "[OchSignal{-251 x 6.25GHz +/- 6.25GHz}, OchSignal{-249 x 6.25GHz +/- 6.25GHz}, OchSignal{-247 x
        "registered-channels": "[OchSignal{-251 x 6.25GHz +/- 6.25GHz}, OchSignal{-249 x 6.25GHz +/- 6.25GHz}, OchSignal{-247
      },
      "S-band": {
        "available-channels": "[OchSignal{+385 x 6.25GHz +/- 6.25GHz}, OchSignal{+387 x 6.25GHz +/- 6.25GHz}, OchSignal{+389 x
        "registered-channels": "[OchSignal{+385 x 6.25GHz +/- 6.25GHz}, OchSignal{+387 x 6.25GHz +/- 6.25GHz}, OchSignal{+389
      }
    },
```

Figure 2-11: response body of GET perBandChahnnels

2.5.4 Nodes

The nodes menu provides information regarding the characteristics of optical nodes. This interface is mainly devoted to export toward the T-API orchestrator the list of devices building up the network.



opmodes : Query OpenConfig operational modes — Show/Hide | List Operations | Expand Operations

intents : Query, submit and withdraw optical intents — Show/Hide | List Operations | Expand Operations

links : Query optical links and resources — Show/Hide | List Operations | Expand Operations

nodes : Query optical devices, ports and resources — Show/Hide | List Operations | Expand Operations

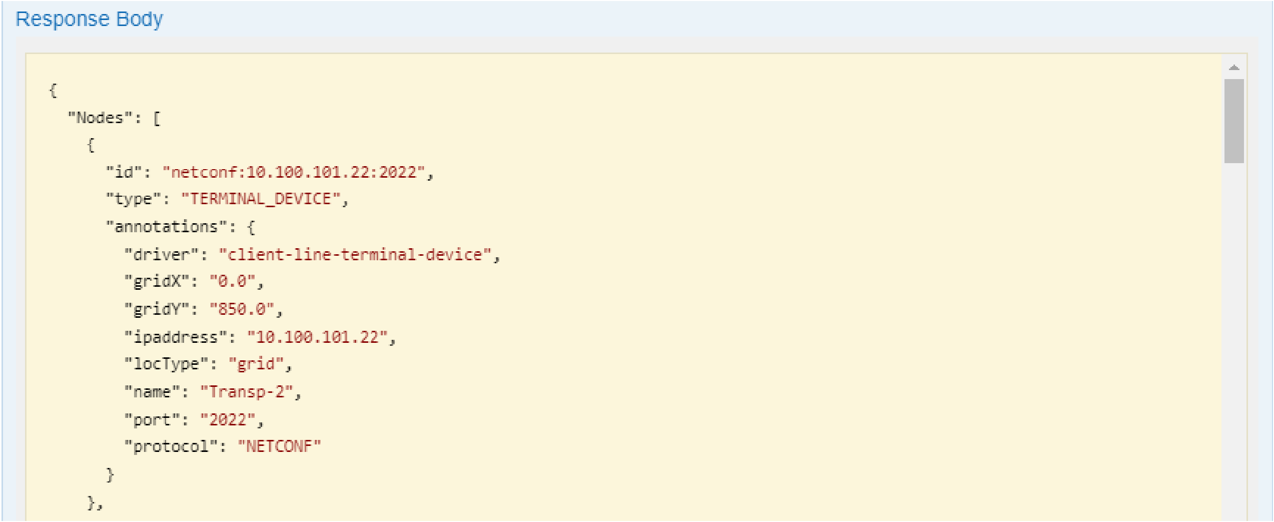
POST	/nodes/annotate/allNodes	Set the annotation on all optical links
GET	/nodes/opticalNodes	Get the optical intents on the network
POST	/nodes/annotate/oneNode	Set an annotation on an optical link
POST	/nodes/annotate/opMode	Add a supported opmode to a device

Figure 2-12: nodes REST APIs.

Among the implemented methods the relevant one for the network operation is the GET method. Other methods have been implemented to facilitate debugging operation during development.

2.5.4.1 Nodes – GET details

The GET method provides the list of nodes including the node type (i.e., TERMINAL_DEVICE for transponders and ROADM).



Response Body

```
{
  "Nodes": [
    {
      "id": "netconf:10.100.101.22:2022",
      "type": "TERMINAL_DEVICE",
      "annotations": {
        "driver": "client-line-terminal-device",
        "gridX": "0.0",
        "gridY": "850.0",
        "ipaddress": "10.100.101.22",
        "locType": "grid",
        "name": "Transp-2",
        "port": "2022",
        "protocol": "NETCONF"
      }
    },
```

Figure 2-13: Response body for GET nodes.

2.6 OpenROADM

[OpenROADM] is a Multi-Source Agreement initiative, active since 2015 and comprising several network operators and optical system and component vendors. From the control plane perspective, OpenROADM defines data models for device, network, and service modelling, targeting the fully disaggregated network model. The device model covers detailed configuration information, alarm, and performance monitoring and, as such, was chosen by the METRO-HAUL project as the reference interface for ROADM devices. Recently, device models have been extended opening to the “partial disaggregated” solution covering also trans-, mux- and switch-ponders.

The METRO-HAUL project developed an OpenROADM driver for the ONOS SDN controller to control ROADM devices. The driver is currently downloadable from the official ONOS repository and available under the ODTN-driver section [ONOS]. During device discovery, ONOS retrieves the number and type of ports together with their capabilities to feed its internal device database. More specifically, the current driver collects the spectral feature of the ports reading the *<mc-capabilities>* branch of the device datastore, available both for degrees and Shared Risk Group (SRG, i.e., add/drop modules), performing a NETCONF *<get>* operation. However, *<mc-capabilities>* allows modelling a single spectral band and cannot be instantiated more than once, so that multi-band devices cannot be covered. Recent updates of the device model (starting from v.7.0.0) address multi-band devices by a new top-level list named *<mc-capabilities-profile>*, very similar to the old *<mc-capability>,* but that can be instantiated several times to describe the different bands and can be referenced by ports, degrees and SRGs, as can be seen in the following tree, extracted from the OpenROADM device model.

```
+--rw circuit-packs* [circuit-pack-name]
|   +--rw circuit-pack-type
|   +--
....
|   +--rw ports* [port-name]
|       +--rw port-name
|       +--
|       +--ro mc-capability-profile-name*
....
+--rw degree* [degree-number]
|   +--rw degree-number
|   +--
|   +--ro mc-capability-profile-name*
```

```
....
+--rw shared-risk-group* [srg-number]
|  +--rw srg-number
|  +--
|  +--ro mc-capability-profile-name*
....
+--ro mc-capability-profile* [profile-name]
   |  +--ro profile-name
   |  +--ro center-freq-granularity?
   |  +--ro min-edge-freq?
   |  +--ro max-edge-freq?
   |  +--ro slot-width-granularity?
   |  +--ro min-slots?
   |  +--ro max-slots?
```

Figure 2-14 Extract of OpenROADM tree for multiband support

All the leaves of the *<mc-capability-profile>* have clear meaning. Only *min-slots* and *max-slots* deserve a short explanation: they are used to advertise the minimum and maximum number of slots (as defined by the *slot-width-granularity* leaf) that can be used to form a spectral window.

For what concerns point-to-multipoint connections, as those needed by XR-optics, current device model doesn't cover them. Coverage for point-to-multipoint optical connections is not yet in the scope of the MSA. Currently, the *<roadm-connection>* container allows creation of both express and add-drop connections uniquely between one source and one destination Network Media Channel (NMC) interface. Generally speaking, nothing prevents to have more than one *<roadm-connection>* referencing the same NMC. However, this is not a shared solution and may create interoperability issues. Alternatively, the container *<roadm-connection>* could be extended to cover more source and destination NMC interfaces. However, this is not allowed by the current device model.

2.7 Interfaces for the Telemetry Platform

The telemetry system includes three different interfaces:

REDIS

We rely on Redis as a demarcation point between data sources and the telemetry system. Telemetry data generated by data sources are encapsulated as JSON objects and published in a Redis database. In particular, the RedisJSON module that provides JSON support for Redis is used. RedisJSON lets store, update, and retrieve JSON values in a Redis database, similar to any other Redis data type. The publish/subscribe paradigm is used to decouple data sources (publishers) from subscribers aimed at reaching good scalability and achieving dynamic messaging routing. The publish/subscribe implementation of Redis ensures the delivery to all the subscribers interested in the topic and offers four models of communication: one-to-one, one-to-many, many-to-one and many-to-many. In particular, the Telemetry system relies on the one-to-many publish/subscribe model.

An example of JSON object exchanged among components is presented next.

```
{
"header": {
"sender": "dataSource-1",
"receiver": "telemetryProcessor"
},
"body": {
```

```
"timeStamp": 1695805907
"freq": [int]
"power_x": [float],
"power_y": [float],
"channel": [<central freq.: int, width: int>]
"location": string
}
}
```

Figure 2-15: Example of JSON object injected by a data source to the telemetry system

gRPC

A gRPC interface is used to transport telemetry data, both measurements and events, between telemetry agents and the telemetry manager. GRPC uses Protocol Buffers as the interface description language to serialize structured data. Protocol buffers have a strict specification, which needs to be defined for every type of data being transported. The Protocol buffers schema implemented in the gRPC interface is presented next.

```
syntax = "proto3";

package receiver;

// The Receiver service definition.
service Receiver
{
  // Sends Data
  rpc SendData(SendDataRequest) returns (SendDataReply) {}
}

// The request message containing the data
message SendDataRequest
{
  bytes message = 1;
}

// The response message containing the reply
message SendDataReply
{
  bytes response = 1;
}
```

Figure 2-16: Telemetry gRPC interface Protocol Buffers schema definition

The schema defines a single service that implements the method used to send the data through the interface. Two types of messages are implemented, one used to convey the data to another gRPC interface and the other to receive the reply. As both messages are of type bytes, data must be serialized before being sent, allowing generalization of the telemetry data to be conveyed. The data is binary serialized using MessagePack, an efficient binary serialization format. Such encoding format brings interoperability between different programming languages as implementations are available for most of them.

Note that, although such encoding could largely increase the volume of transported data, intelligent data aggregation performed by telemetry agents and compression performed by the gRPC interface could reduce such volume to a minimum.

REST API

A REST API has been developed for the remote management of the telemetry system. This interface exposes methods that can be used by external systems to retrieve information or perform management actions, including the reconfiguration of the different nodes in the system. The interface also exposes methods to control the components running inside of each of the telemetry nodes. The REST API authentication is implemented using a user/password verification, which issues a token that can be used for a certain period of time.

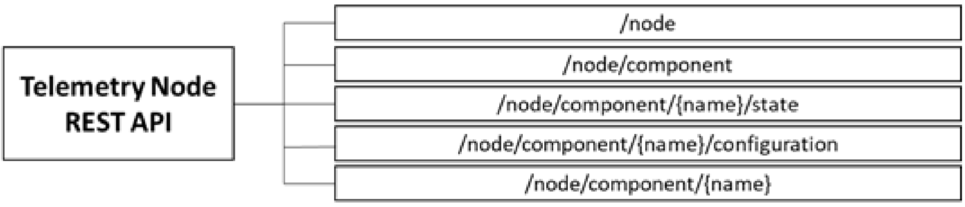


Figure 2-17: *Telemetry Node endpoint structure*

Telemetry Node Management	
HTTP Request	POST /node
Description	Perform an action on a telemetry node. This action is defined by the command key inside the body. The available commands are: *stop*. The command *stop* issues a petition to stop all the running components in the telemetry node and finally shutdown the node.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token

Body	`{`
	`  "command": "stop"`
	`}`

Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	`{`
	`  "result": "string"`
	`}`

Endpoint	GET /node
Description	Obtain the information of the node and the information of the running components
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	`{`
	`  "name": "string",`
	`  "type": "type",`
	`  "components": "dictionary"`
	`}`

Telemetry Component Management	
Endpoint	GET /node/component/{name}/state
Description	Get the state of the component identified by the *name.*
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	`{`
	`"state": "string"`
	`}`

Endpoint	GET /node/component/{name}/configuration
Description	Get the configuration of the component identified by the *name.*
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	`{`
	`"result": "string"`
	`}`

HTTP Request	POST /node/component
Description	Perform an action regarding the components in a telemetry node. This action is defined by the command key inside the body. The available commands are: *deploy*, *start* and *reconfigure*. The command *deploy* issues a petition to deploy a new component identified by *name*. The command *start* issues a petition to run an already deployed component identified by *name*. The command *reconfigure* issues a petition to reconfigure an already deployed or running component identified by *name*.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token

Body	`{`
	`"command": "string",`
	`"name": "string",`
	`"config": "dictionary",`
	`}`

Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	`{`
	`"result": "string"`
	`}`

Endpoint	DELETE /node/component/{name}
Description	Remove the component identified by the *name*.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	{
	  "result": "string"
	}

2.8 CONTROL OF PLUGGABLE MODULES

Common Management Interface Specification (CMIS) [CMIS], defines a generic management communication interface and protocol among the host (e.g., network switch) and modules (e.g., optical transceivers). This interface has been defined to provide a standard across a variety of module capabilities and form factors (QSFP-DD, OSFP, COBO) to foster the vendor agnostic management. The Coherent CMIS (C-CMIS) extends the CMIS interface to handle coherent optics pluggables, e.g., 400ZR(+) modules, that require additional calls to perform interface-related data processing, such as Forward Error Correction (FEC). The CMIS and C-CMIS specifications are defined within the Optical Internetworking Forum (OIF)[OIF] thanks to the joint collaboration of vendors of networks devices. These standards fulfil the needs of B5G-OPEN project for managing ZR and OpenXR pluggable modules within a packet-optical node.

In B5G-OPEN the CMIS/CCMIS implementation in SONiC IPoWDM white box has been deeply investigated, aiming at supporting dynamic configuration of pluggable modules via REST interface through the NETCONF agent.

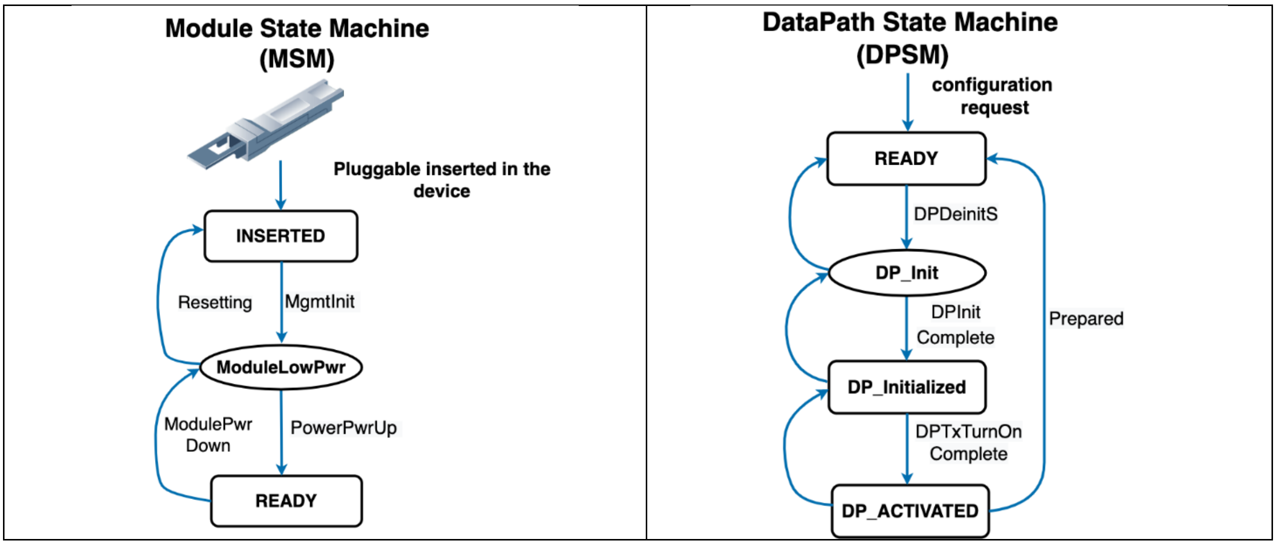
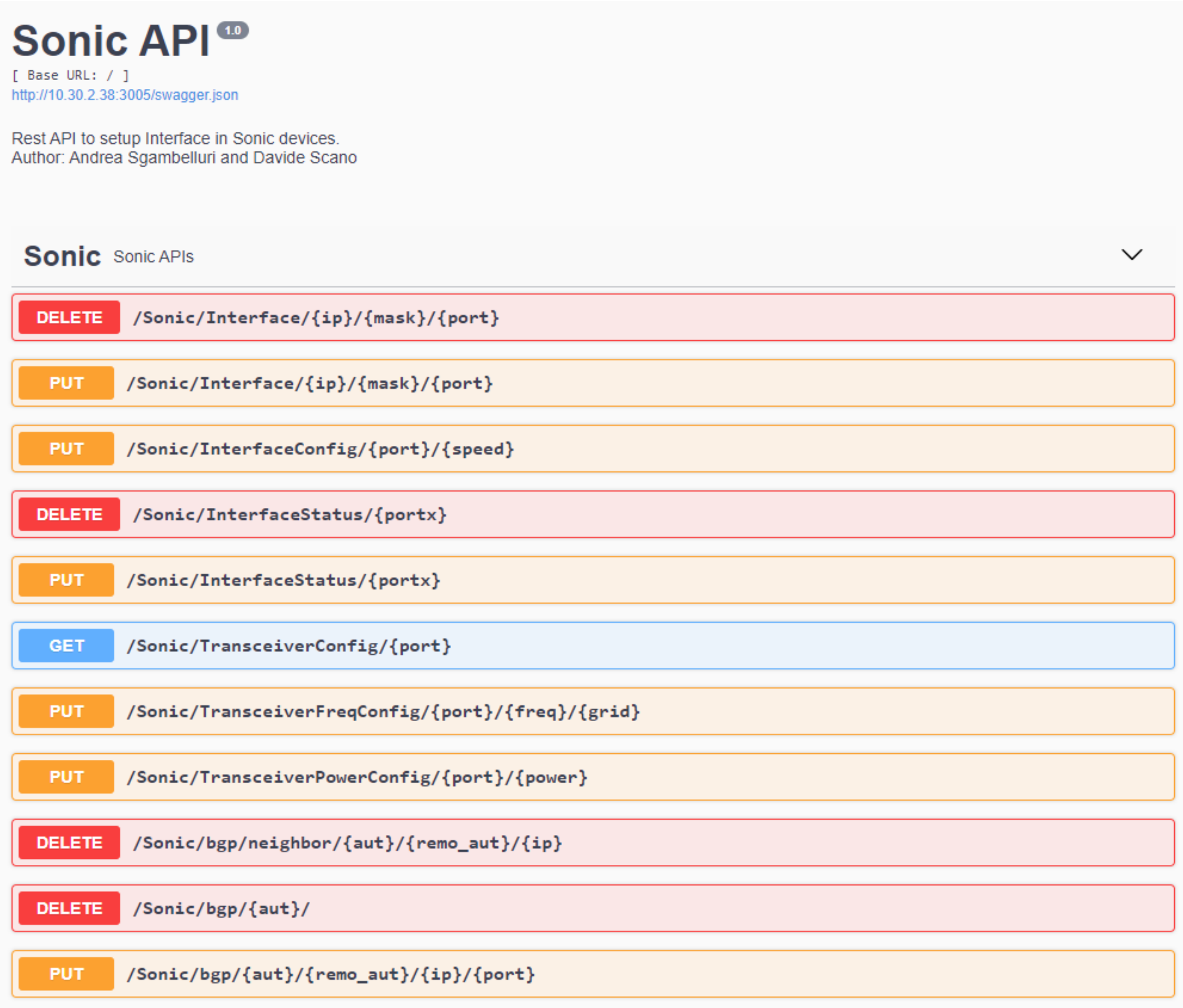


Figure 2-18: (left) CMIS Module State Machine (MSM); (right) Data Path State Machine (DPSM)

The CMIS electrically erasable programmable read-only memory (EEPROM) within the pluggable module is organized in pages, each of which can be read or written to. Each page contains a different set of information, such as: module information, Versatile Diagnostics Monitoring (VDM), configuration, and status. The initialization of the module is represented by the Module State Machine (MSM), while the configuration is represented by the Data Path State Machine (DPSM). The MSM defines the initialization process between the device that hosts the pluggable module and the module itself, as depicted in Figure 2-18(left). Once the pluggable module is inserted into the IPoWDM box, named 'host' in [CMIS], the MSM enters the 'INSERTED' state. Consequently, the module is powered, and the 'MgmtInit' transition phase is initiated. During this phase, the module initializes the Memory Map to default values and sets up the management communication interface, allowing the host to eventually manage the module. After that, the module enters the 'ModuleLowPwr' state, during which the host can configure the module using the management interface to read from and write to the management Memory Map. If the operation is successful, the 'ModulePwrUp' transition phase is triggered, and the host is informed that the module is in the process of powering up to High Power Mode. However, if it fails, the 'Resetting' transition phase is triggered, clearing the Memory Map, and transitioning the MSM to the 'INSERTED' state. When the MSM reaches the 'READY' state, the module is in High Power mode, and the host can initialize or deinitialize DPSM. If the "READY" state is not reached, the 'ModulePwrDown' transition phase is triggered, and the module shall return to the 'ModuleLowPwr' state. The DPSM defines the host-module interactions required to configure the parameters within the module, including transmission power, frequency, threshold alarms, and more. As illustrated in Figure 2-18(right), the module in the 'READY' state is awaiting a configuration request from the host, such as a change in frequency. When a configuration request is received, the 'DPDeinitS' transition phase is initiated. During this phase, the module performs all the necessary deinitialization activities on all resources associated with the current configuration within the module, and the transmission power is turned off. Once the 'DP_Init' state is reached, the 'DPInit Complete' transition phase is triggered, and the module performs all the necessary initialization activities on its internal resources to apply the new configuration. If the 'DPInit Complete' process fails, the transition phase will be restarted. Afterward, the module reaches the 'DP_Initialized' state in this configuration, fully operational, initialized, and ready to transmit traffic, but it is not powered. Subsequently, the 'DPTxTurnON' transition phase is triggered, and the module is powered. If this phase is completed, the module reaches the 'DP_ACTIVATED' state. This indicates that the new configuration has been successfully applied to the module, and traffic can be transmitted and received. Finally, the module is moved to the 'READY' state through the 'Prepared' transition phase, which informs the host that the module is ready to handle a new configuration.

In B5G-OPEN, the control of the above procedure has been successfully integrated in the NETCONF Agent and successfully validated on an EdgeCore 400Gb/s white box. Specific frequencies or power levels have been successfully enforced through the Agent in an automated way. Results show that a change in the configuration is completed in around 14s (9s as Prompt), the same as using the Common Line Interface (CLI), i.e., imposed by HW constraints of pluggable modules.

A REST APIs has been developed on top of the CMIS driver to facilitate the integration with the NETCONF agent. Such interface is used to GET the



Sonic API 1.0

[Base URL: /]
http://10.30.2.38:3005/swagger.json

Rest API to setup Interface in Sonic devices.
Author: Andrea Sgambelluri and Davide Scano

Sonic Sonic APIs

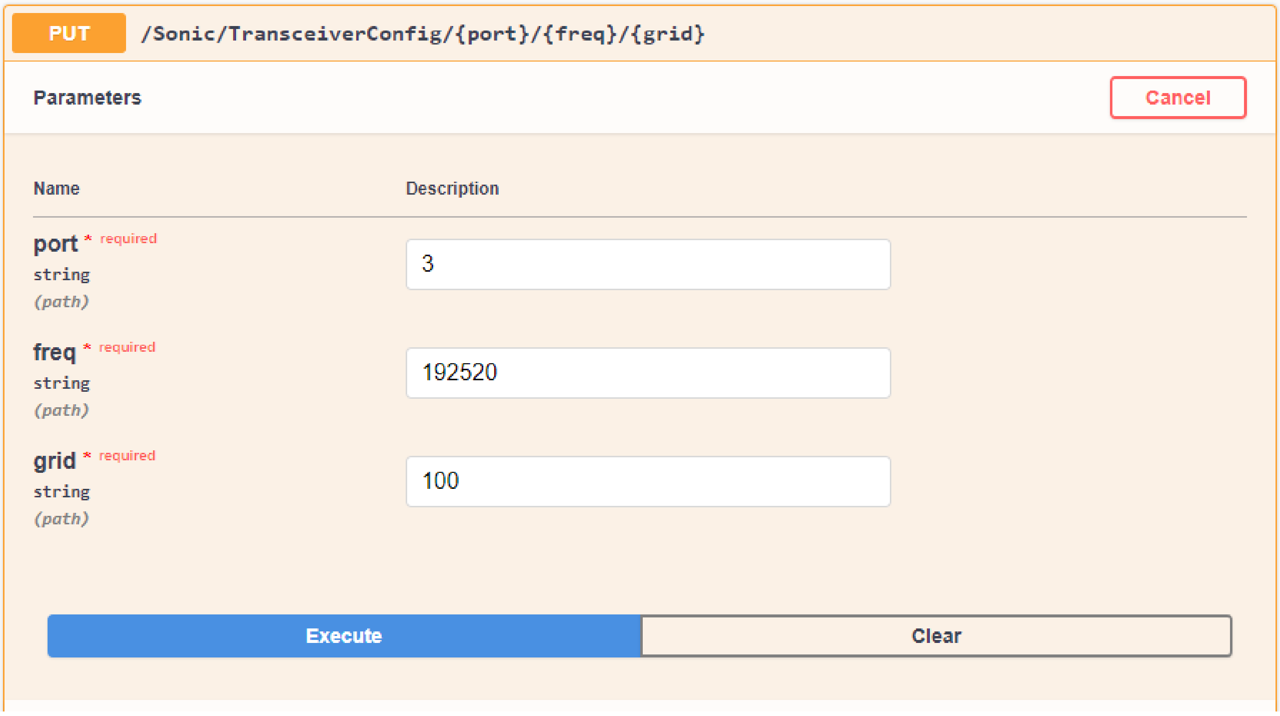
```
DELETE  /Sonic/Interface/{ip}/{mask}/{port}
PUT     /Sonic/Interface/{ip}/{mask}/{port}
PUT     /Sonic/InterfaceConfig/{port}/{speed}
DELETE  /Sonic/InterfaceStatus/{portx}
PUT     /Sonic/InterfaceStatus/{portx}
GET     /Sonic/TransceiverConfig/{port}
PUT     /Sonic/TransceiverFreqConfig/{port}/{freq}/{grid}
PUT     /Sonic/TransceiverPowerConfig/{port}/{power}
DELETE  /Sonic/bgp/neighbor/{aut}/{remo_aut}/{ip}
DELETE  /Sonic/bgp/{aut}/
PUT     /Sonic/bgp/{aut}/{remo_aut}/{ip}/{port}
```

Figure 2-19: REST APIs developed on top of SONIC.

2.8.1 Transceiver Configuration PUT methods details

2.8.1.1 Frequency

This method is used to set the central frequency in the transceiver that is inserted in a specific port. As shown, the method allows to specify the port number, frequency slot and grid. The grid parameter is used because the transceiver supports two grids 100 GHz spacing and 75 GHz spacing. If the specified central frequency does not exactly match on the specified grid the actual value to which the laser is tuned is rounded to the nearest value on the grid.


PUT /Sonic/TransceiverConfig/{port}/{freq}/{grid}

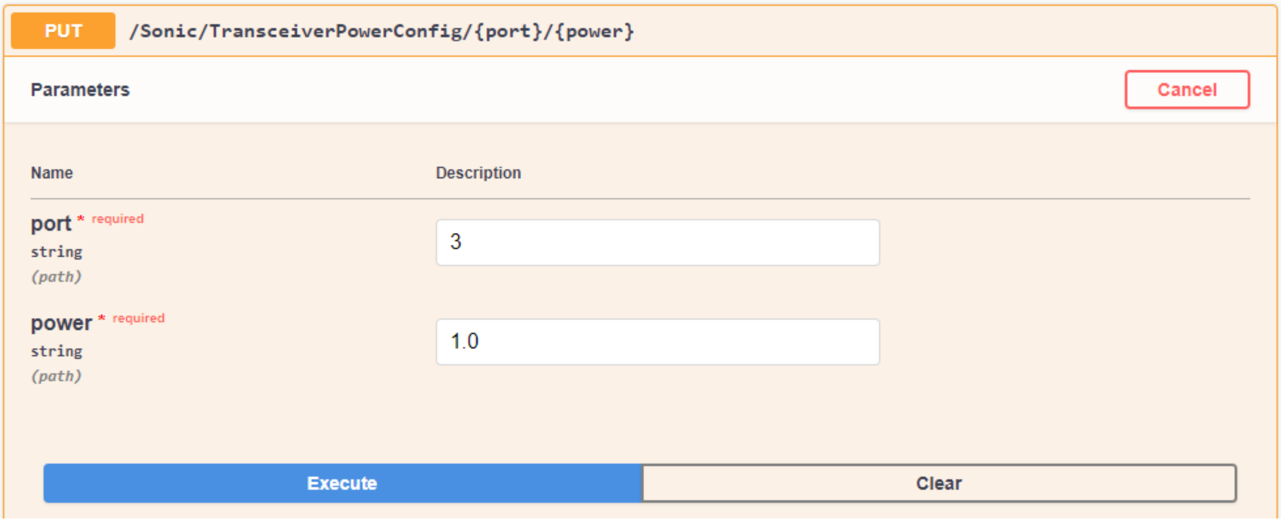
Parameters

Name	Description
port * required string *(path)*	3
freq * required string *(path)*	192520
grid * required string *(path)*	100

Figure 2-20: PUSH method of Transceiver Configuration

2.8.1.2 *Power*

This method is used to set the output power in the transceiver that is inserted in a specific port. As shown, the method allows to specify the port number and the output power.

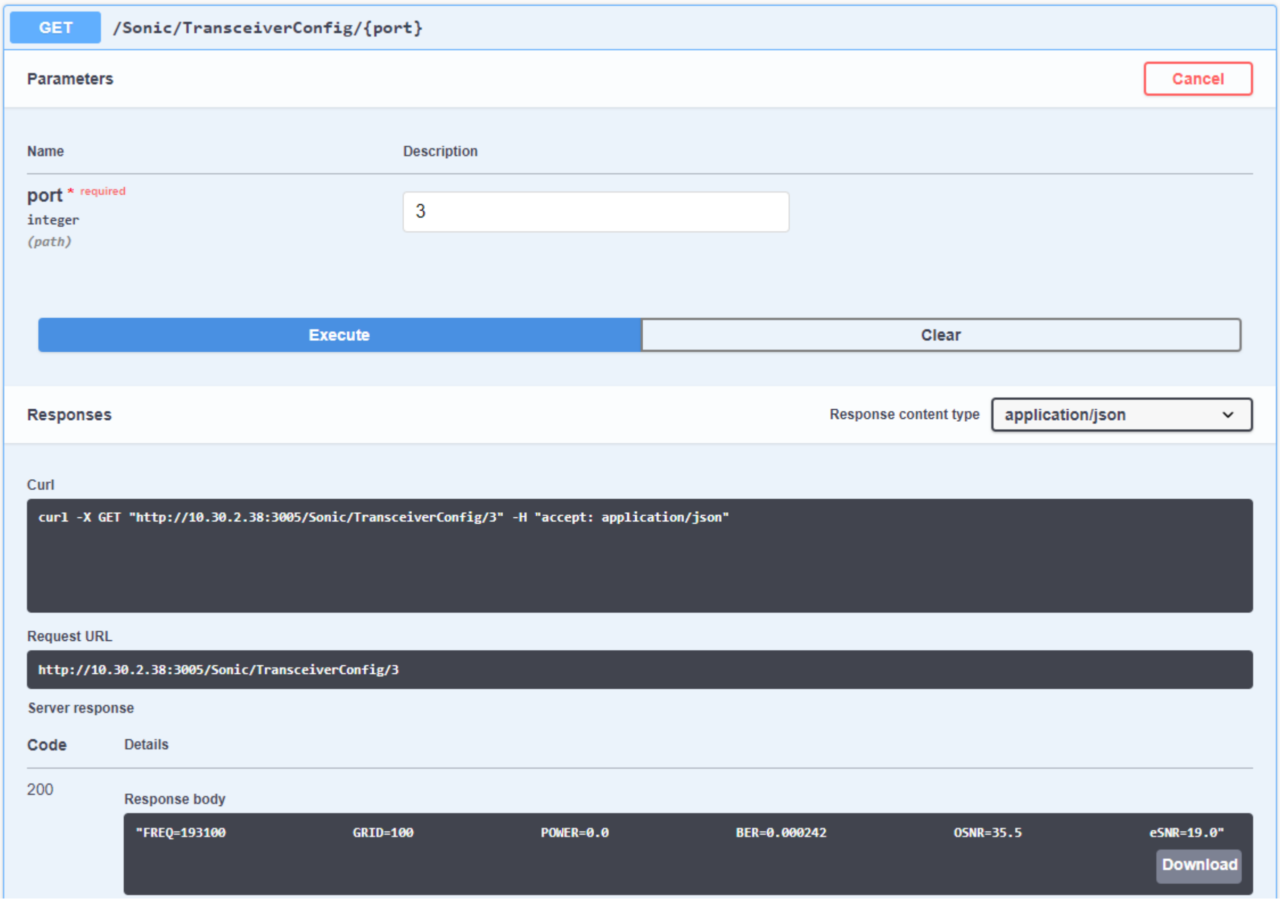

PUT /Sonic/TransceiverPowerConfig/{port}/{power}

Parameters

Name	Description
port * required string *(path)*	3
power * required string *(path)*	1.0

2.8.2 Transceiver Configuration GET method details

This method is used to retrieve information regarding the transceiver inserted in a specific port. This GET method returns the current values of Frequency, Grid, and Power specified through the aforementioned PUT methods, moreover it provides the values of BER, and OSNR measured on the receiver side.



port * required
integer
(path)

3

Curl

```
curl -X GET "http://10.30.2.38:3005/Sonic/TransceiverConfig/3" -H "accept: application/json"
```

Request URL

```
http://10.30.2.38:3005/Sonic/TransceiverConfig/3
```

Server response

Code	Details
200	Response body

```
"FREQ=193100          GRID=100          POWER=0.0          BER=0.000242          OSNR=35.5          eSNR=19.0"
```

Figure 2-21: Response of GET Transceiver Configuration method.

2.9 1.1 PON Control

1.1.1 PON Controller NBI User Authentication

User Authentication	
HTTP Request	POST /users/authenticate/
Description	Authenticate and login the user

Request Body JSON	{
	  "type": "object",
	  "properties": {
	    "email": {
	      "type": "string",
	      "format": "email"
	    },
	    "password": {
	      "type": "string"
	    }
	  },
	  "required": [
	    "email",
	    "password"
	  ]
	}

Response JSON	{
	"status": "success",
	"details": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	},
	"data": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	}
	}

Example Request	{
	"data": {
	"email": "user@example.com",
	"password": "string"
	}
	}

Examples Response	{
	"auth": "True",
	"fName": "Evangelos",
	"lName": "Kosmatos",
	"email": "vkosmatos@openlightcomm.uk",
	"lastLogin": "2023-10-03T17:35:51.623Z",
	"dateJoined": "2023-05-12T09:05:47.169Z",
	"roles": [
	"Administrators"
	],
	"permissions": [
	"other.can_delete_other_branding",
	"accounts.can_read_accounts_admin",
	....
	],
	"active_database": "Default"
	}

2.9.1 1.1.2 PON Controller NBI Retrieve Configuration

Controller Configuration	
HTTP Request	GET /controllers/configs/
Description	Retrieve Controller configuration
Request Body JSON	No body required

Response JSON	{
	"status": "success",
	"details": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"

	},
	"data": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	}
	}

Example Request	No body required

Examples Response	{
	"status": "success",
	"data": [
	{
	"_id": "f4:39:09:70:19:06",
	"CNTL": {
	"CFG Version": "R3.1.0",
	"CFG Change Count": 0,
	"CNTL-ALARM-CFG": "Default",
	"Name": "",
	"Picture": "",
	"Location": "",
	"Address": "",
	"Tag": "",
	"Loop Delay": 3,
	"Max Services": 8,
	"OLT Timeout": 300,
	"ONU Threads": 64,
	"Pause": false,
	"Shutdown": false,
	"Statistic Sample": 300,
	"TAPI Trace": [],
	"Max STATS Size": 1000000000,
	"Max SYSLOG Size": 1000000000,
	"Labels": [],
	"Alarm History Duration": 90,
	"UMT Discovery Timeout": 2,
	"Create Date": "2023-05-12 08:57:06.689787"
	},
	"MGMT LAN": {
	"Name": "Unnamed"
	},
	"OLTs": {
	"Secondary Limit": 65535,
	"Unspecified Limit": 65535,
	"Primary": [],
	"Secondary": [],
	"Excluded": []
	},
	"ONU": {
	"Max FW Upgrades": 16,
	"Service Config Policy": "Disabled",
	"Realtime Stats Sample": 1

	},
	"NETCONF": {
	"Name": "f4:39:09:70:19:06"
	},
	"Alarm History": {
	"Alarm IDs": [],
	"Ack Count": 0,
	"Ack Operator": "",
	"Purge Count": 0
	},
	"Cascading": {
	"Mode Selection": "Dynamic",
	"Groups": {}
	}
	}
	]
	}

2.9.2 1.1.3 PON Controller NBI Retrieve OLTs Configuration

OLT Configuration	
HTTP Request	GET /olts/configs/
Description	Retrieve OLTs configuration
Request Body JSON	No body required

Response JSON	{
	"status": "success",
	"details": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	},
	"data": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	}
	}

Example Request	No body required

Examples Response	{
	"status": "success",
	"data": [
	{
	"_id": "70:b3:d5:52:35:48",
	"CNTL": {
	"CFG Version": "R3.1.0"
	},
	"EPON": {
	"Discovery Period": 3000,

```
      "Encryption": "Downstream",
      "Encryption Key Time": 900,
      "FEC": true,
      "Grant Spacing": 16,
      "Laser OFF": 32,
      "Laser ON": 32,
      "Max Frame Size": 9600,
      "Sync Time": 16
    },
    "GPON": {
      "Discovery Period": 3000,
      "Downstream Fec": true,
      "Encryption": "Bidirectional",
      "Encryption Key Time": 600,
      "Error Det Min Sample": 100,
      "Error Det Max Ratio": 20,
      "Guard Time": 64,
      "Max Frame Size": 9600,
      "PON ID": 0,
      "Upstream FEC 0": true,
      "Upstream FEC 1": true,
      "Upstream FEC 2": true,
      "Upstream FEC 3": true,
      "Upstream Preamble 0": 64,
      "Upstream Preamble 1": 64,
      "Upstream Preamble 2": 64,
      "Upstream Preamble 3": 64
    },
    "LLDP": {
      "Transmit": false,
      "Receive": true
    },
    "NNI": {
      "Max Frame Size": 9600
    },
    "NNI Networks": [],
    "OLT": {
      "OLT-ALARM-CFG": "Default",
      "Auto Boot Mode": false,
      "CFG Change Count": 4,
      "Debug Dump Count": 0,
      "Debug Log Level": "NONE",
      "FW Bank Files": [],
      "FW Bank Ptr": 65535,
      "FW Bank Versions": [],
      "Name": "OLT01",
      "Location": "",
      "Address": "",
      "Picture": "",
      "Tag": "",
      "PON Enable": true,
      "PON Mode": "GPON",
      "Reg Allow Count": 0,
        "Reg Allow ONU": "ALL",
        "Reset Count": 1,
        "Max Round Trip Time": 250,
        "Labels": [],
        "Shared Downstream Policer": true,
        "Create Date": "2023-05-18 17:32:52.693648"
    },
    "ONU": {
        "Max FW Upgrades": 4,
        "Realtime Stats": false,
        "Enable All Serial Numbers Count": 0
    },
    "ONUs": {
        "AZRS00012230": {
            "ALLOC ID (OMCC)": 1,
            "Disable": false,
            "Enable Count": 0,
            "Disable Count": 0,
            "OLT-Service 0": 1154
        },
        "CMTD424cafdd": {
            "ALLOC ID (OMCC)": 2,
            "Disable": false,
            "Enable Count": 0,
            "Disable Count": 0,
            "OLT-Service 0": 1155
        }
    },
    "Protection": {
        "Peer": "",
        "Switchover Count": 0,
        "Inactive Periods": {
            "Active": 100,
            "Standby": 200
        },
        "Watch": "Disabled",
        "Watch Count": 0
    },
    "NETCONF": {
        "Name": "70:b3:d5:52:35:48"
    },
    "MAC Learning": {
        "Age Limit": 300,
        "Allow CPEs To Move": false
    },
    "Alarm History": {
        "Alarm IDs": [],
        "Ack Count": 0,
        "Ack Operator": "",
        "Purge Count": 0
    },
    "Switch": {
        "ANY": "ANY"
```

	} }] }

2.9.3 1.1.4 PON Controller NBI Retrieve ONUs Configuration

ONU Configuration	
HTTP Request	GET /onus/configs/
Description	Retrieve OLTs configuration
Request Body JSON	No body required

Response JSON	{
	"status": "success",
	"details": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	},
	"data": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	}
	}

Example Request	No body required

Examples Response	{
	"status": "success",
	"data": [
	{
	"_id": "AZRS00012230",
	"CNTL": {
	"CFG Version": "R3.1.0"
	},
	"OLT": {
	"MAC Address": [
	"ANY"
	]
	},
	"OLT-Service 0": {
	"Enable": true,
	"Service Reference": "",
	"Name": "",
	"NNI Network": [
	"s0.c0.c0"
	],
	"PON Network": [
	"s0.c0.c0"
	],
	"Internal Network": {},

```
      "US CoS Treatment": "copy",
      "US CoS Value": 0,
      "DHCP": {
        "Remote ID": "",
        "Circuit ID": "",
        "Sub Options": "",
        "Enterprise Number": 54469
      },
      "PPPoE": {
        "Remote ID": "",
        "Circuit ID": ""
      },
      "RADIUS": {
        "NAS Identifier": "",
        "NAS Port ID": "",
        "User Name Override": "Default"
      },
      "Filter": {
        "DHCPv4": "pass",
        "DHCPv6": "pass",
        "EAPOL": "pass",
        "PPPoE": "pass"
      },
      "SLA-CFG": "Max",
      "DS-MAP-CFG": "",
      "Learning Limit": 2046,
      "Drop Unknown Source MAC": false
    },
    "OLT-Service 1": {
      "Enable": false,
      "Service Reference": "",
      "Name": "",
      "NNI Network": [],
      "PON Network": [],
      "Internal Network": {},
      "US CoS Treatment": "copy",
      "US CoS Value": 0,
      "DHCP": {
        "Remote ID": "",
        "Circuit ID": "",
        "Sub Options": "",
        "Enterprise Number": 54469
      },
      "PPPoE": {
        "Remote ID": "",
        "Circuit ID": ""
      },
      "RADIUS": {
        "NAS Identifier": "",
        "NAS Port ID": "",
        "User Name Override": "Default"
      },
      "Filter": {
```

	"DHCPv4": "pass", "DHCPv6": "pass", "EAPOL": "pass", "PPPoE": "pass" }, "SLA-CFG": "Max", "DS-MAP-CFG": "", "Learning Limit": 2046, "Drop Unknown Source MAC": false } }] }

2.9.4 1.1.5 PON Controller NBI Retrieve SLAs Configuration

Retrieve SLAs / Bandwidth Profile Configuration	
HTTP Request	GET /slas/
Description	Retrieve all SLAs
Request Body JSON	No body required

Response JSON	{
	"status": "success",
	"details": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	},
	"data": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	}
	}

Example Request	No body required

Examples Response	{
	"status": "success",
	"data": [
	{
	"_id": "Max",
	"CNTL": {
	"CFG Version": "R3.1.0"
	},
	"Up Fixed Rate": 0,
	"Up Guaranteed Rate": 128,
	"Up Guaranteed Max Burst": 409600,
	"Up Best Effort Rate": 10000000,
	"Up Best Effort Max Burst": 409600,
	"Up Priority": 1,

	"Up Service Limit": 128,
	"Up Min Grant Period": 0,
	"Up Max Grant Period": 10,
	"Down Guaranteed Rate": 128,
	"Down Guaranteed Max Burst": 256000,
	"Down Best Effort Rate": 10000000,
	"Down Best Effort Max Burst": 256000
	},
	{
	"_id": "Min",
	"CNTL": {
	"CFG Version": "R3.1.0"
	},
	"Up Fixed Rate": 0,
	"Up Guaranteed Rate": 128,
	"Up Guaranteed Max Burst": 409600,
	"Up Best Effort Rate": 0,
	"Up Best Effort Max Burst": 409600,
	"Up Priority": 1,
	"Up Service Limit": 2,
	"Up Min Grant Period": 0,
	"Up Max Grant Period": 40,
	"Down Guaranteed Rate": 128,
	"Down Guaranteed Max Burst": 256000,
	"Down Best Effort Rate": 0,
	"Down Best Effort Max Burst": 256000
	}
	]
	}

2.9.5 1.1.6 PON Controller NBI SLA Creation

SLAs / Bandwidth Profile Creation	
HTTP Request	POST /slas/
Description	Retrieve SLAs

Request Body JSON	{
	"type": "object",
	"properties": {
	"data": {
	"type": "object",
	"additionalProperties": {},
	"description": "SLA-CFG"
	}
	},
	"required": [
	"data"
	]
	}

Response JSON	{
	"status": "success",
	"details": {

	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	},
	"data": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	}
	}

Example Request	{
	"data": {
	"_id" : "example_sla",
	"CNTL" : {
	"CFG Version" : "R3.0.0"
	},
	"Up Fixed Rate" : 0,
	"Up Guaranteed Rate" : 128,
	"Up Guaranteed Max Burst" : 409600,
	"Up Best Effort Rate" : 10000000,
	"Up Best Effort Max Burst" : 409600,
	"Up Priority" : 1,
	"Up Service Limit" : 128,
	"Up Min Grant Period" : 0,
	"Up Max Grant Period" : 10,
	"Down Guaranteed Rate" : 128,
	"Down Guaranteed Max Burst" : 256000,
	"Down Best Effort Rate" : 10000000,
	"Down Best Effort Max Burst" : 256000
	}
	}

Examples Response	{
	"status": "success",
	"data": [...]
	}

2.9.6 1.1.7 PON Controller NBI SLA Update

SLAs / Bandwidth Profile Update	
HTTP Request	PUT /slas/{sla_id}
Description	Update specific SLA

Request Body JSON	{
	"type": "object",
	"properties": {
	"data": {
	"type": "object",
	"additionalProperties": {},
	"description": "SLA-CFG"
	}
	},
	"required": [

	"data"] }

Response JSON	{
	"status": "success",
	"details": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	},
	"data": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	}
	}

Example Request	{
	"data": {
	"_id" : "example_sla ",
	"CNTL" : {
	"CFG Version" : "R3.0.0"
	},
	"Up Fixed Rate" : 0,
	"Up Guaranteed Rate" : 128,
	"Up Guaranteed Max Burst" : 409600,
	"Up Best Effort Rate" : 10000000,
	"Up Best Effort Max Burst" : 409600,
	"Up Priority" : 1,
	"Up Service Limit" : 128,
	"Up Min Grant Period" : 0,
	"Up Max Grant Period" : 10,
	"Down Guaranteed Rate" : 128,
	"Down Guaranteed Max Burst" : 256000,
	"Down Best Effort Rate" : 10000000,
	"Down Best Effort Max Burst" : 256000
	}
	}

Examples Response	{
	"status": "success",
	"data": […]
	}

2.9.7 1.1.8 PON Controller NBI SLA Delete

SLAs / Bandwidth Profile Update	
HTTP Request	DEL /slas/{sla_id}
Description	Delete specific SLA

Request Body JSON	{
	"type": "object",
	"properties": {

	"data": {
	"type": "object",
	"additionalProperties": {},
	"description": "SLA-CFG"
	}
	},
	"required": [
	"data"
	]
	}

Response JSON	{
	"status": "success",
	"details": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	},
	"data": {
	"additionalProp1": "string",
	"additionalProp2": "string",
	"additionalProp3": "string"
	}
	}

Example Request	{
	"data": {
	"_id" : "example_sla ",
	}
	}

Examples Response	{
	"status": "success",
	"data": […]
	}

2.10 LiFi Integration

2.10.1 NBI for the LiFi controller

The NBI for the LiFi Controller serves as a channel for communication with the broader B5G-OPEN infrastructure. Notably, this NBI is implemented based on REST API principles, ensuring an easily accessible and standardised interface. By leveraging these endpoints described below, users can proficiently manage the LiFi Access Points (APs) connected to the system.

1) **Device Management Endpoints**:

- Get device name:
  ‘GET /devices/{device-id}/lifi/interface/name’
  Retrieve the unique name designated to the specified LiFi device.

- Set device name:
  ‘PUT /devices/{device-id}/lifi/interface/name’
  Update or assign a distinct name to the specified LiFi device.

- Get device status:
  ‘GET /devices/{device-id}/lifi/interface/status’
  Determine the current operational status (either 'up' or 'down') of the specified LiFi device.

2) **IP Configuration Endpoints**:

- Get IP address:
  ‘GET /devices/{device-id}/lifi/interface/ip-addr’
  Fetch the current IP address configuration of the specified LiFi device.

- Set IP address:
  ‘PUT /devices/{device-id}/lifi/interface/ip-addr’
  Update or assign a new IP address for the specified LiFi device.

- Get netmask:
  ‘GET /devices/{device-id}/lifi/interface/netmask’
  Retrieve the subnet mask associated with the specified LiFi device.

- Set netmask:
  ‘PUT /devices/{device-id}/lifi/interface/netmask’
  Update or assign a new subnet mask for the specified LiFi device.

- Get gateway:
  ‘GET /devices/{device-id}/lifi/interface/gateway’
  Fetch the gateway address currently set for the specified LiFi device.

- Set gateway:
  ‘PUT /devices/{device-id}/lifi/interface/gateway’
  Update or designate a new gateway address for the specified LiFi device.

3) Wireless Configuration Endpoints:

- Get wireless status:
  ‘GET /devices/{device-id}/lifi/interface/access-point/enabled’
  Identify whether the wireless capability on the specified LiFi device is active (enabled) or not.

- Enable or disable wireless:
  ‘PUT /devices/{device-id}/lifi/interface/access-point/enabled’
  Toggle the wireless functionality of the specified LiFi device between enabled and disabled states.

- Get operating mode:
  ‘GET /devices/{device-id}/lifi/interface/access-point/mode’
  Retrieve the current operational mode (e.g., 'ap' for Access Point) set for the specified LiFi device.

- Set operating mode:
  ‘PUT /devices/{device-id}/lifi/interface/access-point/mode’

Define the desired operational mode for the specified LiFi device.

- Get SSID:
  ‘GET /devices/{device-id}/lifi/interface/access-point/ssid’
  Access the SSID being broadcast by the specified LiFi device.

- Set SSID:
  ‘PUT /devices/{device-id}/lifi/interface/access-point/ssid’
  Define or update the SSID for the specified LiFi device to broadcast.

- Set password:
  ‘PUT /devices/{device-id}/lifi/interface/access-point/security/password’
  Assign a new or update the existing password for the specified LiFi network.

- Set encryption type:
  ‘PUT /devices/{device-id}/lifi/interface/access-point/security/encryption’
  Designate the desired encryption method for the specified LiFi network's security.

4) **Network Configuration Endpoints**:

- Get VLAN ID:
  ‘GET /devices/{device-id}/lifi/interface/access-point/vlan-id’
  Fetch the VLAN ID associated with the specified LiFi network.

- Set VLAN ID:
  ‘PUT /devices/{device-id}/lifi/interface/access-point/vlan-id’
  Assign or update the VLAN ID for the specified LiFi network.

Below is a representation of the default setting of a LiFi device in JSON:

```
{
 "lifi": {
  "interface": {
   "name": "wlan0",
   "status": "up",
   "ip-addr": "192.168.1.100",
   "netmask": "255.255.255.0",
   "gateway": "192.168.1.1",
"access-point": {
    "enabled": true,
    "mode": "ap",
    "ssid": "LiFi",
    "security": {
     "password": "string",
     "encryption": "NONE"
    },
    "vlan-id": 1
   }
  }
 }
}
```

2.10.2 NBI for the LiFi AP

The primary protocol underling the LiFi AP NBI implementation for configuration pureposes is NETCONF. It is an XML-based protocol that provides mechanisms to install, manipulate, and

```
module: plf-lifi
 +--rw lifi
   +--rw interface
     +--rw name?          string
     +--rw status?        enumeration {down, up}
     +--rw ip-addr?       ip-address
     +--rw netmask?        ip-address
     +--rw gateway?        ip-address
     +--rw access-point
       +--rw enabled?     boolean
       +--rw mode?        enumeration {ap}
       +--rw ssid?        string
       +--rw security
       | +--rw password?  string
       | +--rw encryption? enumeration {WPA2, NONE}
       +--rw vlan-id?     uint32
```

delete the configuration of network devices. Below is the YANG model for the LiFi AP, indicating the primary configuration items:

As seen from the structure, the YANG model describes the primary components and configurations of the LiFi AP. This includes basic configurations such as the device's name, IP address, netmask, and gateway, along with wireless-specific settings under the access-point container, such as SSID, security configurations, and VLAN ID. For example, the retrieve the current IP configuration data for the LiFi AP:

It's noteworthy to mention that, for users aiming to integrate the LiFi AP with broader network systems, the combination of NETCONF and YANG provides a standardized, predictable interface. This facilitates seamless integrations, efficient management, and ensures compatibility with a wide range of network management tools.

While this NETCONF-based NBI is primarily used for configuration and management tasks, the LiFi AP also features another NBI for telemetry purposes. Instead of configuration, this interface focuses on the real-time export of telemetry data, providing performance metrics and operational insights from the LiFi AP. A detailed description on this telemetry interface will be presented in subsequent sections of this documentation.

2.11 B5G-OPEN North Bound Interface

The B5G-OPEN project will provide ad-hoc developed APIs using REST/APIs paradigm, following best practices (e.g., Open-API documentation), as the northernmost API of the ecosystem, this way being exposed by B5G-ONP component. The target user of this API in an industrial deployment would be e.g., the operators' OSS systems, or directly operator personnel in charge on provisioning of the different services, IT, IP, DSR or everything together encapsulated under the network slice principles.

```
<get>
    <filter type="subtree">
      <interface xmlns="http://.purelifi.com/yang">
        <ip-addr/>
        <netmask/>
        <gateway/>
      </interface>
    </filter>
 </get>
```

This API is designed to provide open and programmatic access to the different use cases to be demonstrated. The details of these APIs will be defined later in the project, appropriately reported.

General policies have been already defined for those APIs:

- REST-based API, exposed via open documentation frameworks, preferably OpenAPI.
    - Making use of the IETF YANG model for network slices [NSv16].
- Secured service provisioning operations enabled by user authentication.
- Adoption of the Optimization-as-a-Service (OaaS) paradigm [Gar19][Pav15]:

- This relies on the concept of *algorithm repository,* as a set of algorithms exposed, browsable in a catalogue, and runnable via the open APIs. A subset of the algorithms developed along B5G-OPEN will be integrated using this form.
- Utilization of container models for shipping algorithm implementations. This is an enabler for integrating algorithms developed in different languages and platforms (a practical aspect, that becomes a booster for the OaaS concept), and possibly by third parties, into the B5G-ONP system.

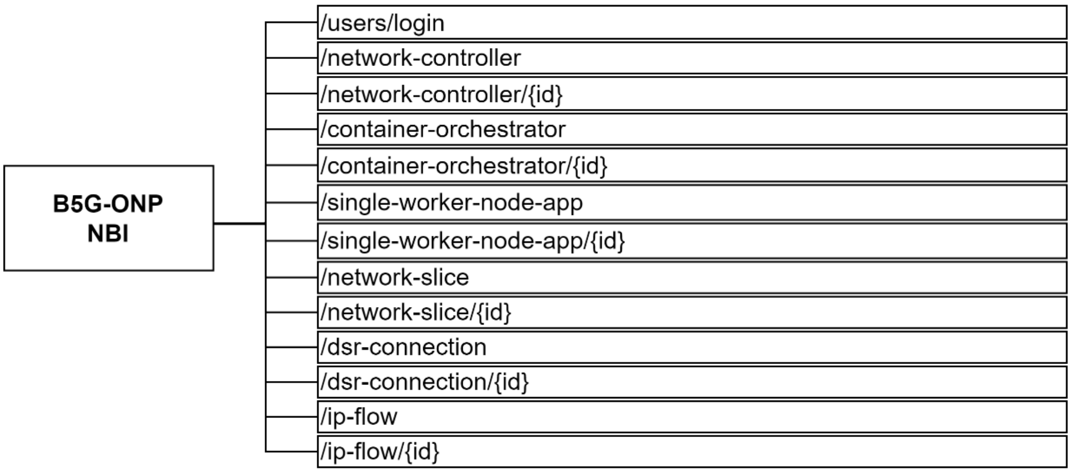


Figure 2-22: B5G-ONP NBI endpoint structure

2.11.1 B5G-ONP NBI User Authentication

User Authentication	
HTTP Request	POST /user/login
Description	This endpoint is used by upper layer user to access to the API. This is performed by sending user/password credentials and the B5G-ONP's AUTH sub-module oversees processing them and returning a concrete token for that user and session. The bearer token is necessary to remotely execute the rest of HTTP requests offered by this NBI API.
Headers	{"Content-Type": "application/json", "Accept": "application/json"}
Authentication	User/password

Body	`{`
	`  "grantType": "token",`
	`  "user": "string",`
	`  "password": "string",`
	`}`

Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	`{`
	`  "token": "string"`
	`}`

2.11.2 B5G-ONP NBI SDN Network Controller Management

SDN Network Controller Management

HTTP Request	POST /network-controller
Description	Registers a new network controller to be orchestrated by the B5G-ONP system and a unique *controller-id* is returned if proceed. The types of network controllers compatibles are the ones expected to interact with the B5G-ONP: IP, Optical, XR an PON.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token

Body	{
	  "uid": "string",
	  "type": "string",
	  "ip-address": "string",
	  "port": 0,
	  "user": "string",
	  "password": "string"
	}

Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	{
	  "id": "string"
	}

Endpoint	GET /network-controller
Description	Obtain the list of registered controllers in the B5G-ONP.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	[
	  {
	    "uid": "string",
	    "type": "type",
	    "ip-address": "string",
	    "port": 0,
	    "user": "string",
	    "password": "string"
	  }
	]

Endpoint	GET /network-controller/{id}
Description	Obtain the information of the SDN network controller associated to a concrete *id*.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	{
	  "uid": "string",
	  "type": "type",
	  "ip-address": "string",
	  "port": 0,
	  "user": "string",
	  "password": "string"
	}

Endpoint	DELETE /network-controller/{id}
Description	Unregister the SDN network controller associated to a concrete *id*.

Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request
Response JSON	

2.11.3 B5G-ONP NBI Container Orchestrator Management

Container Orchestrator Management	
HTTP Request	POST /container-orchestrator
Description	Registers a new container orchestrator. During the lifetime of the project is expected to handle just Kubernetes type. Additionally, it is possible to execute specific configuration profiles in the “kube-config” field in a similar way than in the Kubernetes control plane API. An *id* is assigned id registered.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token

Body	{
	  "uid": "string",
	  "type": "Kubernetes",
	  "ip-address": "string",
	  "kube-config": "string"
	}

Response code	200 OK / 404 Not Found / 422 Error

Response JSON	{
	  "id": "string"
	}

Endpoint	GET /container-orchestrator
Description	Obtain the list of registered container orchestrators in the B5G-ONP system.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	[
	  {
	    "uid": "string",
	    "type": "Kubernetes",
	    "ip-address": "string",
	    "kube-config": "string"
	  }
	]

Endpoint	GET /container-orchestrator/{id}
Description	Obtain the information of a given container orchestrator associated to a concrete *id*.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	{
	  "uid": "string",
	  "type": "Kubernetes",
	  "ip-address": "string",
	  "kube-config": "string"
	}

Endpoint	DELETE /container-orchestrator/{id}
Description	Unregister the container orchestrator associated to a concrete *id*.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request
Response JSON	

2.11.4 B5G-ONP NBI Network application provisioning

Network Application Provisioning	
HTTP Request	POST /single-worker-node-app
Description	Sends a POST request to instantiate a network service/application in a single worker node of a Kubernetes cluster previously registered in the B5G-OPEN ecosystem. To facilitate its usage, the data model of the body of the request is based on the K8s model and naming. For instance, it uses K8s concepts as pod, service or image. To clarify, this functionality is restricted to deploy only a network application in one single worker node, considering one or multiple pods related by the same service. Here, only IT resources are considered in the provisioning.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token

Body	{
	  "container-orchestrator-uid": "string",
	  "worker-node": "string",
	  "metadata": {
	    "name": "string",
	    "namespace": "string",
	    "labels": "string",
	    "annotations": "string"
	  },
	  "containers": [
	    {
	      "name": "string",
	      "image": "string",
	      "ports": "string",
	      "resources": {
	        "cpu-requests": 0,
	        "cpu-limits": 0,
	        "memory-requests": 0,
	        "memory-limits": 0
	      },
	      "env": "string",
	      "command": "string"
	    }
	  ],
	  "volumes": [

```
    "string"
  ],
  "restartPolicy": "string",
  "services": {
    "name": "string",
    "type": "LoadBalancer",
    "pod-label": "string",
    "ports": [
      {
        "name": "string",
        "internal-port": 0,
        "external-port": 0
      }
    ]
  }
}
```

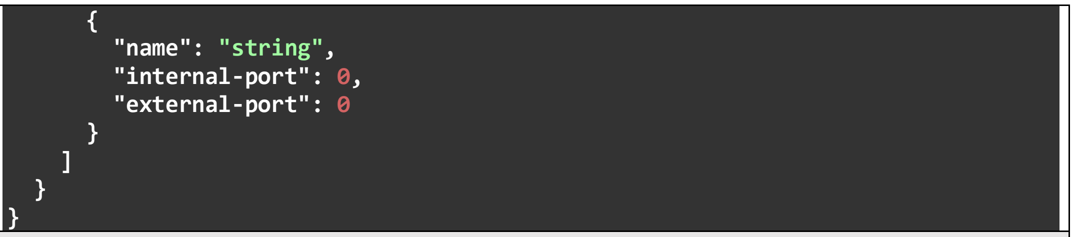
Response code	200 OK / 404 Not Found / 422 Error

Response JSON	{
	  "id": "string"
	}

Endpoint	GET /single-worker-node-app/{id}
Description	Obtain the information of the instantiation of a network application registered in the system with an *id*.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

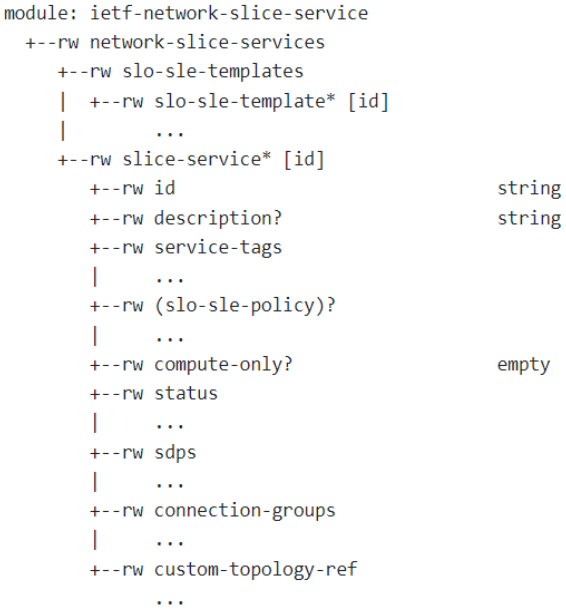
Response JSON

```
{
  "container-orchestrator-uid": "string",
  "worker-node": "string",
  "metadata": {
    "name": "string",
    "namespace": "string",
    "labels": "string",
    "annotations": "string"
  },
  "containers": [
    {
      "name": "string",
      "image": "string",
      "ports": "string",
      "resources": {
        "cpu-requests": 0,
        "cpu-limits": 0,
        "memory-requests": 0,
        "memory-limits": 0
      },
      "env": "string",
      "command": "string"
    }
  ],
  "volumes": [
    "string"
  ],
  "restartPolicy": "string",
  "services": {
    "name": "string",
    "type": "LoadBalancer",
    "pod-label": "string",
    "ports": [
```

	
Endpoint	DELETE /single-worker-node-app/{id}
Description	Send a request to remove the deployment of a single-worker-node kubernetes service assigned with a given *id*.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request
Response JSON	

```
          {
            "name": "string",
            "internal-port": 0,
            "external-port": 0
          }
        ]
      }
}
```

2.11.5 B5G-ONP NBI Network slice provisioning

Network Slice Provisioning	
HTTP Request	POST /network-slicing
Description	Sends a POST request to instantiate a network slice, defining a potentially multi-worker node network application reserving both IT and network resources among the VNF/CNFs of the network slice. The model of the request is assumed as a mix of two models, one for the IT management defined as the previously presented single-worker-node-app (note that can define more than one) and the YANG model of the IETF Network Slice Service. To clarify, the field “ietf-network-slice-definition:additionalProp1” is expected to receive a JSON that complies with the IETF NSS model, as depicted as follows: 
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}

```
module: ietf-network-slice-service
  +--rw network-slice-services
     +--rw slo-sle-templates
     |  +--rw slo-sle-template* [id]
     |        ...
     +--rw slice-service* [id]
        +--rw id                              string
        +--rw description?                    string
        +--rw service-tags
        |     ...
        +--rw (slo-sle-policy)?
        |     ...
        +--rw compute-only?                   empty
        +--rw status
        |     ...
        +--rw sdps
        |     ...
        +--rw connection-groups
        |     ...
        +--rw custom-topology-ref
              ...
```

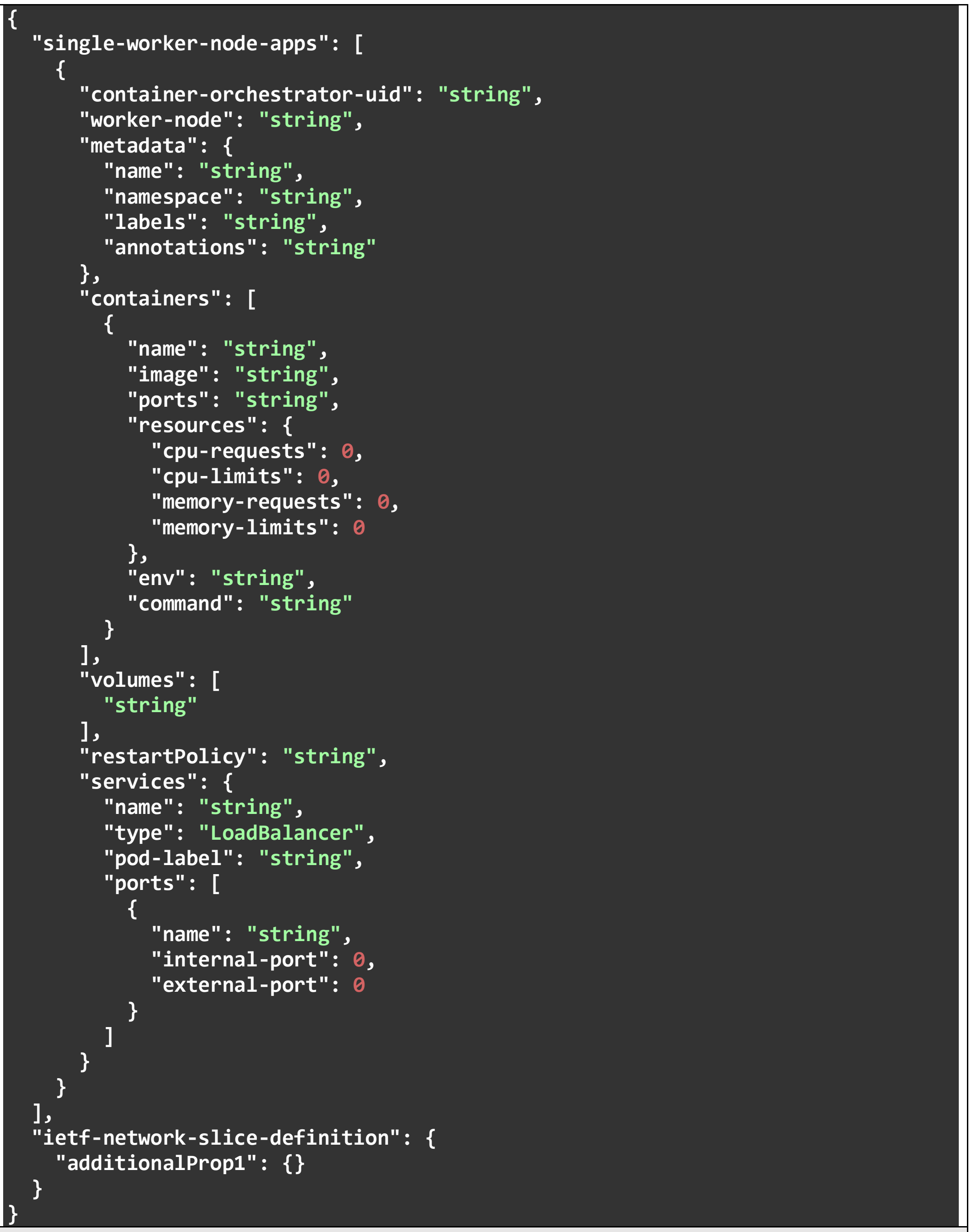
Authentication	Bearer Token
Body	

```
{
  "single-worker-node-apps": [
    {
      "container-orchestrator-uid": "string",
      "worker-node": "string",
      "metadata": {
        "name": "string",
        "namespace": "string",
        "labels": "string",
        "annotations": "string"
      },
      "containers": [
        {
          "name": "string",
          "image": "string",
          "ports": "string",
          "resources": {
            "cpu-requests": 0,
            "cpu-limits": 0,
            "memory-requests": 0,
            "memory-limits": 0
          },
          "env": "string",
          "command": "string"
        }
      ],
      "volumes": [
        "string"
      ],
      "restartPolicy": "string",
      "services": {
        "name": "string",
        "type": "LoadBalancer",
        "pod-label": "string",
        "ports": [
          {
            "name": "string",
            "internal-port": 0,
            "external-port": 0
          }
        ]
      }
    }
  ],
  "ietf-network-slice-definition": {
    "additionalProp1": {}
  }
}
```

Response code	200 OK / 404 Not Found / 422 Invalid Request
Response JSON	

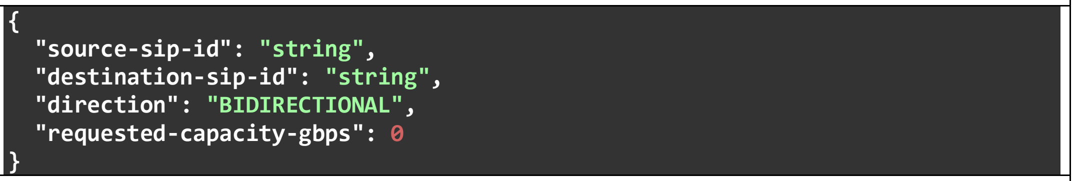
```
{
  "id": "string"
}
```

Endpoint	GET /network-slice/{id}
Description	Obtain the information related to the network-service identified with *id*
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

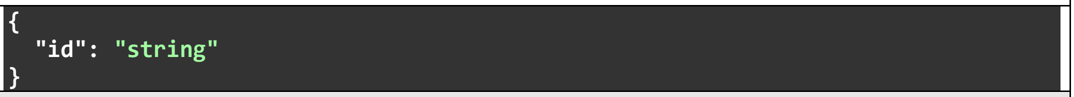
<table>
<tr><td>Response JSON</td><td>
<pre>
{
  "single-worker-node-apps": [
    {
      "container-orchestrator-uid": "string",
      "worker-node": "string",
      "metadata": {
        "name": "string",
        "namespace": "string",
        "labels": "string",
        "annotations": "string"
      },
      "containers": [
        {
          "name": "string",
          "image": "string",
          "ports": "string",
          "resources": {
            "cpu-requests": 0,
            "cpu-limits": 0,
            "memory-requests": 0,
            "memory-limits": 0
          },
          "env": "string",
          "command": "string"
        }
      ],
      "volumes": [
        "string"
      ],
      "restartPolicy": "string",
      "services": {
        "name": "string",
        "type": "LoadBalancer",
        "pod-label": "string",
        "ports": [
          {
            "name": "string",
            "internal-port": 0,
            "external-port": 0
          }
        ]
      }
    }
  ],
  "ietf-network-slice-definition": {
    "additionalProp1": {}
  }
}
</pre></td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Endpoint</td><td>DELETE /network-service/{id}</td></tr>
<tr><td>Description</td><td>Remove the connection and service deployment of a network-slice registered with <i>id</i></td></tr>
<tr><td>Headers</td><td>{“Content-Type”: “application/json”, “Accept”: “application/json”}</td></tr>
<tr><td>Authentication</td><td>Bearer Token</td></tr>
<tr><td>Body</td><td>-</td></tr>
<tr><td>Response code</td><td>200 OK / 404 Not Found / 422 Invalid Request</td></tr>
<tr><td>Response JSON</td><td></td></tr>
</table>

2.11.6 B5G-ONP NBI DSR connection provisioning

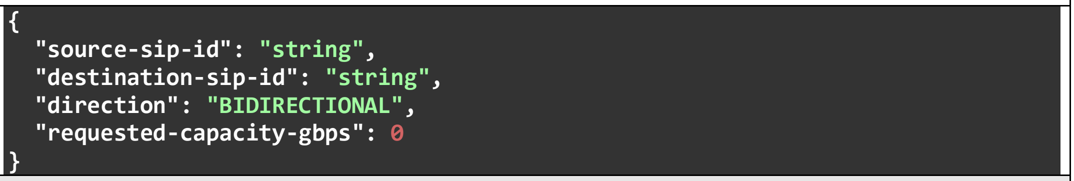
DSR Connection Provisioning	
HTTP Request	POST /dsr-connection
Description	Request to create a DSR connection in the underlying OTN network indicating, the source and destination nodes, direction and capacity to reserve in the network.
Headers	{"Content-Type": "application/json", "Accept": "application/json"}
Authentication	Bearer Token

Body	
	{
	  "source-sip-id": "string",
	  "destination-sip-id": "string",
	  "direction": "BIDIRECTIONAL",
	  "requested-capacity-gbps": 0
	}

Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	
	{
	  "id": "string"
	}

Endpoint	GET /dsr-connection/{id}
Description	Obtain the information of a DSR connection established in the B5G-ONP system with *id*.
Headers	{"Content-Type": "application/json", "Accept": "application/json"}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

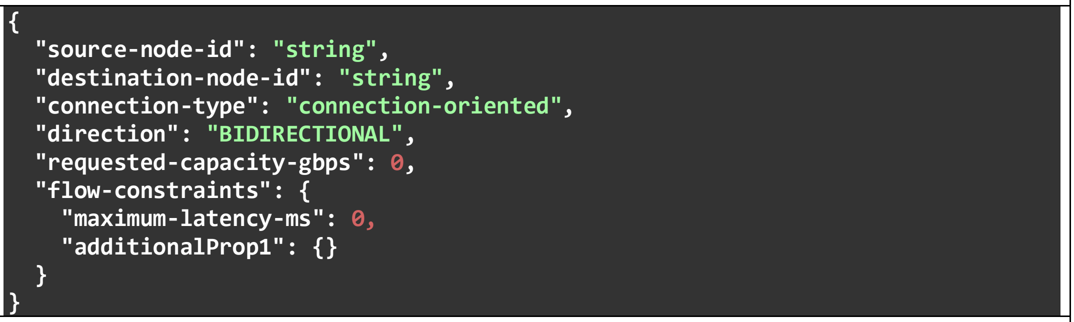
Response JSON	
	{
	  "source-sip-id": "string",
	  "destination-sip-id": "string",
	  "direction": "BIDIRECTIONAL",
	  "requested-capacity-gbps": 0
	}

Endpoint	DELETE /dsr-connection/{id}
Description	Delete the DSR connection established in the underlying OTN network associated to a concrete *id*.
Headers	{"Content-Type": "application/json", "Accept": "application/json"}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request
Response JSON	

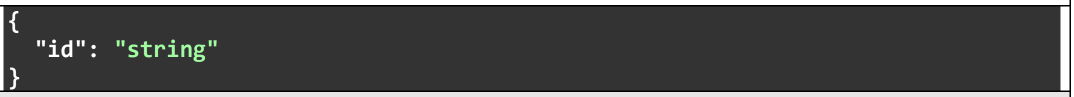
2.11.7 B5G-ONP NBI IP flow provisioning

IP Flow Provisioning	
HTTP Request	POST /ip-flow

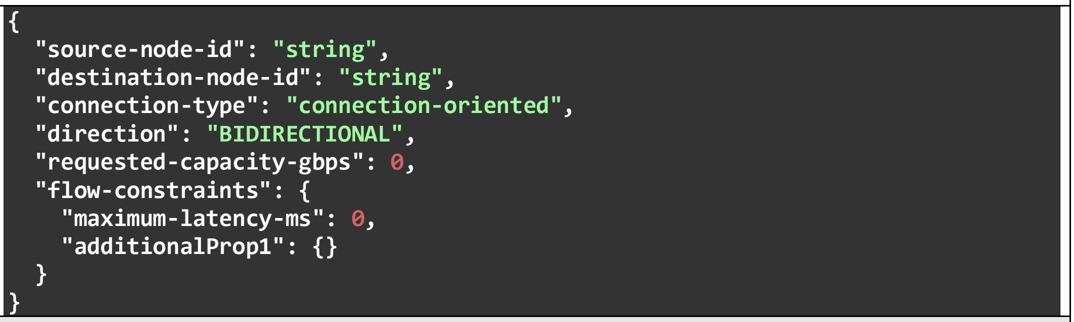
Description	Similar to DSR connections, this request creates an IP connection in the IP/MPLS domain. It is worth mentioning that flow constraints can be added to the the definition of an IP flow to assume, for instance, maximum latency or other IP-related metrics.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token

Body	
	{
	  "source-node-id": "string",
	  "destination-node-id": "string",
	  "connection-type": "connection-oriented",
	  "direction": "BIDIRECTIONAL",
	  "requested-capacity-gbps": 0,
	  "flow-constraints": {
	    "maximum-latency-ms": 0,
	    "additionalProp1": {}
	  }
	}

Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	
	{
	  "id": "string"
	}

Endpoint	GET /ip-flow/{id}
Description	Obtain the information an IP flow established with an *id*.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request

Response JSON	
	{
	  "source-node-id": "string",
	  "destination-node-id": "string",
	  "connection-type": "connection-oriented",
	  "direction": "BIDIRECTIONAL",
	  "requested-capacity-gbps": 0,
	  "flow-constraints": {
	    "maximum-latency-ms": 0,
	    "additionalProp1": {}
	  }
	}

Endpoint	DELETE /ip-flow/{id}
Description	Remove IP connection registered with a concrete *id*.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found / 422 Invalid Request
Response JSON	

Note that the definition of this NBI may undergo some variation, adding or removing functionalities or model changes as needs arise during the life of the project.

2.12 Kubernetes

The innovative solution proposed by B5G-OPEN represents a new method to manage resources and services in the Kubernetes environment, leveraging the power and flexibility of the Kubernetes API Server (kube-apiserver). This resource, located at the core of the Kubernetes cluster, is available in Kubernetes master and allows control and monitoring to efficiently manage resources under this environment.

The Kubernetes architecture adopted in this project is divided into two fundamental elements that work together to achieve effective orchestration and management as shown in Figure 2-23: Kubernetes cluster architecture (source: [K8s])

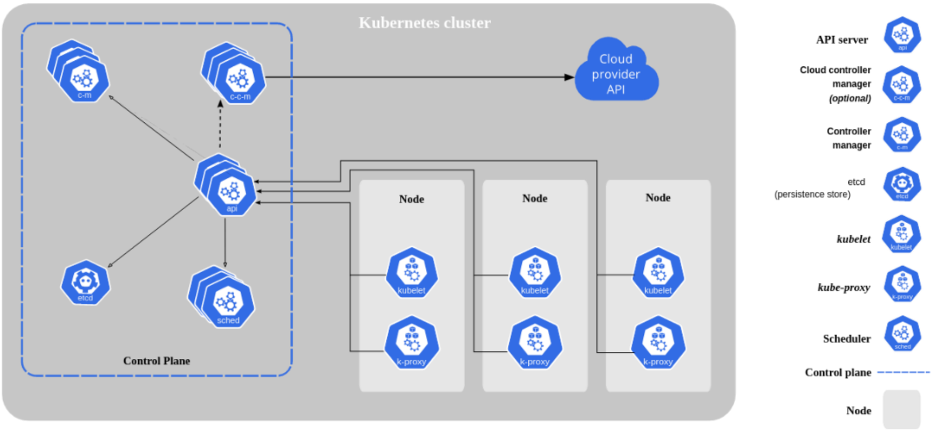


Figure 2-23: Kubernetes cluster architecture (source: [K8s])

- **Control Plane:** It represents the brain of Kubernetes, being the epicentre of all decisions such as scheduling pods, maintaining desired state, or adapting to changes in the environment. Among its main components are *kube-apiserver*, Controller Manager, Scheduler and etcd. Each of them has its role in the coherent and fluid operation of the cluster.
- **Worker Nodes:** These are machines within the cluster where containerized applications are executed. These nodes receive instructions and directives from the control plane, allowing the execution of tasks and reporting the status of the same.

By deepening the role of *kube-apiserver*, a series of endpoints emerge that become the primary reference points for the comprehensive management of resources within the previously outlined architecture. To interact with these endpoints, it is necessary to meet the authentication and authorization requirements established by Kubernetes. These security measures are designed to safeguard access to cluster resources through its API, thus ensuring the integrity and confidentiality of data and operations.

The integration of the B5G-OPEN Platform with Kubernetes and its focus on kube-apiserver underline the importance of a secure and efficient link between the platform and the underlying infrastructure. Next, we will explore in more depth the different endpoints and how this synergy with Kubernetes empowers advanced resource and service management in the context of the B5G-OPEN architecture. Figure 2-24 summarizes the main endpoints. To achieve this, the content follows the identical structure provided in section 8.4 of D1.2 within the B5G-OPEN project.

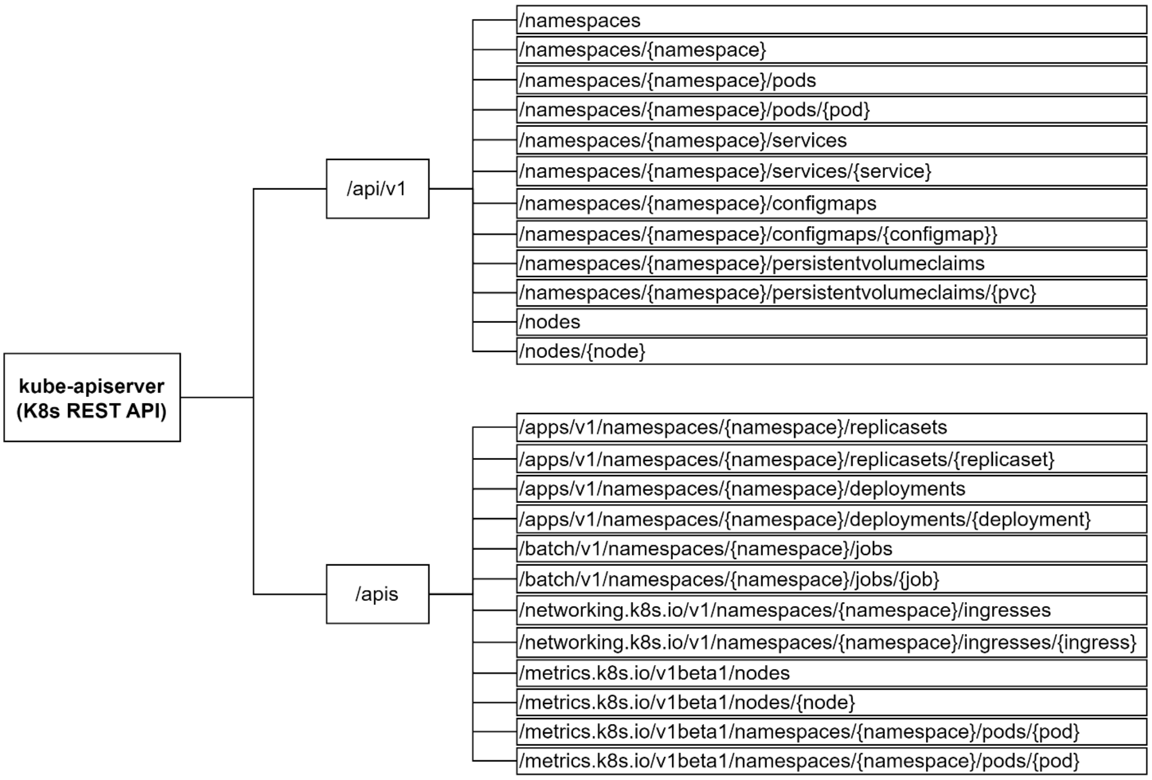

kube-apiserver (K8s REST API)
/api/v1
/namespaces
/namespaces/{namespace}
/namespaces/{namespace}/pods
/namespaces/{namespace}/pods/{pod}
/namespaces/{namespace}/services
/namespaces/{namespace}/services/{service}
/namespaces/{namespace}/configmaps
/namespaces/{namespace}/configmaps/{configmap}}
/namespaces/{namespace}/persistentvolumeclaims
/namespaces/{namespace}/persistentvolumeclaims/{pvc}
/nodes
/nodes/{node}
/apis
/apps/v1/namespaces/{namespace}/replicasets
/apps/v1/namespaces/{namespace}/replicasets/{replicaset}
/apps/v1/namespaces/{namespace}/deployments
/apps/v1/namespaces/{namespace}/deployments/{deployment}
/batch/v1/namespaces/{namespace}/jobs
/batch/v1/namespaces/{namespace}/jobs/{job}
/networking.k8s.io/v1/namespaces/{namespace}/ingresses
/networking.k8s.io/v1/namespaces/{namespace}/ingresses/{ingress}
/metrics.k8s.io/v1beta1/nodes
/metrics.k8s.io/v1beta1/nodes/{node}
/metrics.k8s.io/v1beta1/namespaces/{namespace}/pods/{pod}
/metrics.k8s.io/v1beta1/namespaces/{namespace}/pods/{pod}

Figure 2-24: Structure for the kube-apiserver endpoints

Given the inherent complexity of the Kubernetes model and the plethora of functional possibilities offered by its *kube-apiserver*, in this project we only focus on the HTTP requests related to the jobs to be performed by the B5G-ONP in IT orchestration and network app provisioning. Thus, the metrics to obtain and operations to command under the umbrella of the B5G-OPEN project are related to pods, services, namespace and deployments only. This set of requests and intelligence will be developed in as Kubernetes client in the B5G-ONP's SBI module.

To complement the understanding, in the following table two exemplary requests are shown to highlight the usage of the Kubernetes API, in this case, first, to retrieve overall information of the pods instantiated in a given Kubernetes namespace, and second, to obtain information about a particular pod instance individually:

Pods	
Endpoint	GET /api/v1/namespaces/*{namespace}*/pods
Description	The endpoint is used to retrieve information about pods in a specific namespace in a Kubernetes cluster. Pods are the basic unit of deployment in Kubernetes, and they can contain one or more containers. The default namespace is default. When you make a call to this endpoint, the response is a JSON object that contains details about the pods in the specified namespace. The response may include information such as the names of the pods, their states, the container images, the IP addresses, the associated labels, and other relevant attributes. It is important to note that if there are many pods in the namespace, the response may be paginated, which means that pagination links will be provided to access additional results if necessary.
Headers	{"Content-Type": "application/json", "Accept": "application/json"}
Authentication	Bearer Token
Body	-

Response code	200 OK

Endpoint	GET /api/v1/namespaces/*{namespace}*/pods/{pod}
Description	This API endpoint allows users to retrieve detailed information about a specific Pod within the designated namespace of a Kubernetes cluster. A Pod is the smallest deployable unit in Kubernetes, representing a single instance of a running process within the cluster. The response will be in JSON format, providing comprehensive details about the requested Pod. This information includes metadata, status, container details, labels, and other relevant attributes that describe the state and configuration of the Pod.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	-
Response code	200 OK / 404 Not Found

2.13 Telemetry Data Sources

2.13.1 TAPI Optical Network Orchestrator / SDN controller

In this case, the objective is to use streaming (mechanism that handles the providing of data from one system to another in some form of steady and continuous data flow) for the reporting (notification) of ongoing change of state of the controlled system from one Management-Control entity (TAPI Optical Network Orchestrator) to another (usually superior) management-control entity. Since a significant part of the information is derived from instrumentation the data flow is often called telemetry. A streaming approach is defined that focuses on conveying TAPI entities, i.e., yang sub-trees and allow a client to achieve and maintain eventual consistency with the state of the controlled system.

In this setting, an Event source/server streaming mechanism is made available as an alternative to traditional notifications. The streaming capability is distinct from TAPI Notification and is designed to better deal with scale and to provide an improved operational approach. In this context, any component of the SDN control plane may act as a source of streaming telemetry.

In particular, the TAPI Optical Network Orchestrator SDN controller will act as a data source. For this, the internal architecture of the software will be modified to report asynchronous events that happen in the network Macroscopically, the component will implement a REDIS client following the B5G-OPEN network streaming telemetry architecture and will generate asynchronous events related to topology and connection management. The events that will be notified cover network events, related to:

- Topology (new link, new node, updated node edge point…)
- Connectivity (new service, new connection)

The encoding of such events follows TAPI streaming and Telemetry yang model. For example, the next snippet shows a specific event:

```
{
   "metadata":{
      "measurement":"EventTelemetry",
      "index":"sdn_index"
   },
   "data":{
      "tapi-streaming:log-record":{
         "log-record-body":{
            "event-time-stamp":{
               "primary-time-stamp":"2022-11-02 11:05:25.080535465 UTC"
            },
            "link":{
               "cost-characteristic":[
                  {
                     "cost-name":"te-metric",
                     "cost-value":"1.000000"
                  }
               ],
               "direction":"UNIDIRECTIONAL",
               "layer-protocol-name":[
                  "PHOTONIC_MEDIA"
               ],
               "node-edge-point":[
                  {
                     "node-edge-point-uuid":"89937add-3380-58ab-94ff-9b2fb4efbeeb",
                     "node-uuid":"589df6c1-90e1-51f5-bda4-b4cd6b2d01e4",
                     "topology-uuid":"d8013ae5-12d1-54c0-b653-5d3b5080989f"
                  }
               ],
               "uuid":"71505848-d2b3-57dd-8069-295ce111ec61"
            },
            "record-content":"LINK"
         },
         "log-record-header":{
            "entity-key":"71505848-d2b3-57dd-8069-295ce111ec61",
            "log-append-time-stamp":"2022-11-02 11:05:25.080519123 UTC",
            "record-type":"RECORD_TYPE_CREATE_UPDATE",
            "tapi-context":"1d2ba340-41c3-53a9-a615-88380211e6fc",
            "token":"0"
         }
      }
   }
}
```

2.13.2 LiFi Access Points

The LiFi AP also serves as a significant source of telemetry data. This data plays a crucial role in monitoring, maintaining, and optimizing the performance of the LiFi network.

Prometheus, an industry-standard monitoring system, fetches metrics from its monitored entities by scraping metrics exposed on HTTP endpoints. While numerous systems adopt a push model, sending metrics directly to their monitoring platforms, Prometheus distinguishes itself by employing a pull or scrape approach.

The LiFi AP is harnessed as a Prometheus exporter utilizing the Lua scripting language, renowned for its lightweight footprint. This exporter undertakes the role of aggregating the LiFi telemetry data and formulating it into a structure palatable for Prometheus. After the data extraction by the Lua script, these metrics are displayed on the '/metrics' HTTP endpoint, setting the stage for Prometheus server's periodic scrapes.

Here is the telemetry workflow:

1) Data Collection at the AP:

Using utilities like ‘ubus’ and ‘iwinfo’, the LiFi AP extracts crucial telemetry data. Information like signal strength, link speed, inactivity durations, and data transmission rates are some of the metrics harvested.

- ‘ubus’ offers a comprehensive messaging interface, facilitating queries and interaction with various system components of the AP, including the wireless network status.
- ‘iwinfo’ acts as a bridge over diverse Linux commands, ensuring consistent information retrieval across different wireless device drivers.

2) Exporting Data to Prometheus Server:

As a Prometheus client, the LiFi AP collects its telemetry data, making it accessible for Prometheus to scrape.

3) Data Storage in JSON Format

The telemetry data sent by the AP is well structured and converted in the JSON format.

4) Telemetry Adaptor:

The ‘TelemetryAdaptor’ class receives data in JSON format and sends it into a Redis Stream inside the GODAI architecture.

By following these processes, the LiFi AP ensures that the B5G-OPEN infrastructure remains informed on the LiFi status and ready for further management.

The current telemetry data metrics include signal strength (‘station.signal’), inactivity time (‘station.inactive’), transmission rate (‘station.tx_rate’), reception rate (‘station.rx_rate’), number of transmitted packets (‘station.tx_packets’), and number of received packets (‘station.rx_packets’). Here’s an example of the telemetry data:

lifi_station_signal_dbm{mac="70:B3:D5:95:91:7A",interface="LiFi-Sec"} -43
lifi_station_transmit_kilobits_per_second{mac="70:B3:D5:95:91:7A",interface="LiFi-Sec"} 35000
lifi_station_receive_kilobits_per_second{mac="70:B3:D5:95:91:7A",interface="LiFi-Sec"} 10000
lifi_station_transmit_packets_total{mac="70:B3:D5:95:91:7A",interface="LiFi-Sec"} 297
lifi_station_receive_packets_total{mac="70:B3:D5:95:91:7A",interface="LiFi-Sec"} 30

The telemetry data being represented in a structed JSON format is as follows:

```
{
  "interface": "LiFi-Sec",
  "mac": "70:B3:D5:95:91:7A",
  "lifitelemetry": {
    "signal_dbm": -43,
    "transmit_speed_kbps": 35000,
    "receive_speed_kbps": 10000,
    "transmit_packets_total": 297,
    "receive_packets_total": 30
  }
}
```

2.13.3 Data Collection

“Flex-Telemetry” (see Figure 413) is a program that performs periodically requests to collect performance measurements from ADVA/ADTRAN devices, using NETCONF and a combination of open (OpenConfig) and proprietary data models. Meanwhile, a modular plugin system provides a NBI interface capable of providing a stable source of stream telemetry to different mediums, such as time-series and in-memory DBs, International Data Spaces (IDS)

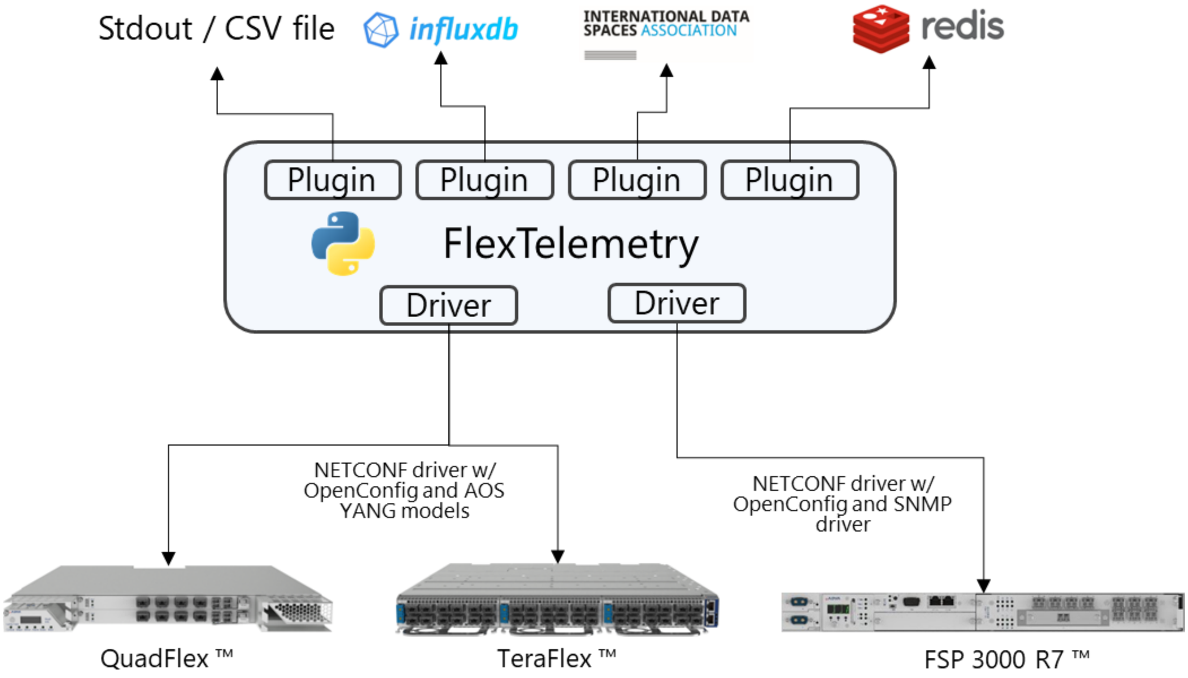

Stdout / CSV file
influxdb
INTERNATIONAL DATA SPACES ASSOCIATION
redis
Plugin
Plugin
Plugin
Plugin
FlexTelemetry
Driver
Driver
NETCONF driver w/ OpenConfig and AOS YANG models
NETCONF driver w/ OpenConfig and SNMP driver
QuadFlex ™
TeraFlex ™
FSP 3000 R7 ™

Figure 2-25: ADVA Flex-Telemetry agent.

The FlexTelemetry agent was extended with a northbound plugin to the Redis database and a southbound driver for monitoring of OLS elements such as amplifiers and ROADMs. The driver is based on Netconf using OpenConfig models and SNMP polling for parameters such as Laser bias current for amplifiers which do not support NETCONF).

The FlexTelemetry agent was successfully tested in the OFC 2023 demo [OFC2023]. The KPIs to measure the saleability and performance of the FlexTelemetry agent will be minimum time interval between reads and number of supported parameters/devices with different SBI drivers and NBI plugins.

2.13.4 Spectrum monitoring

To measure optical spectrum, we leverage the telemetry system developed by B5G-OPEN by implementing a telemetry adaptor. We can monitor the spectrum at the ingress and at the egress port of an optical node.

For monitoring purposes, we use the Nokia Optical Network testbed which combines more than 400 km optical fibre with 7 Nodes (Figure 2-26). The hardware elements come from 3 different vendors (Nokia, Lumentum, and Bell Labs prototype). The mesh network testbed also integrates offline lab measurement. We have fully characterized i) the fibres through fibre length, loss and chromatic dispersion and ii) optical nodes: ROADM losses. We can insert traffic thanks to real-time transponders (12 commercial elastic TRX Line) and offline transponder. Other channels based on ASE noise are available to load the testbed. The testbed is continuously monitored by our agent.

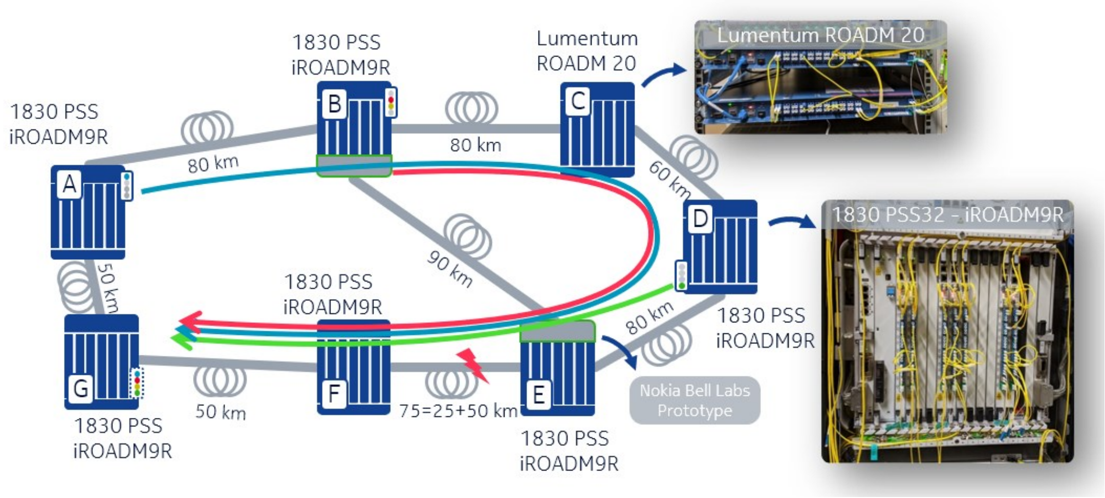


Figure 2-26 Nokia Optical Network testbed

The telemetry message for spectrum monitoring is sent to REDIS instance via JSON with the following structure:

```
{
    freq : [<array of frequency value in MHz>]
    power_x : [<array of power value in mB>],
    power_y : [<array of power value in mB>],
    channel: [<array of [<central channel frequency MHz>,< channel width in
MHz >]>]
    location:<string containing location identifier>
}
```

This spectrum monitoring has been used in a B5G-OPEN demonstration showcased in [Gon23].

3 B5G-OPEN Optical Network Planner (B5G-ONP)

The B5G-OPEN Optical Network Planner (B5G-ONP) component is an integral part of the control plane and is orchestrating both IT and network resources. Within the B5G-OPEN project, the B5G-ONP serves as the hub that provides design, optimization, and planning tools for deploying, managing and configuring services and resources, easing the integration with external components. It includes three main key modules:

- Provisioning and Discovery for resource allocation and identification, ensuring seamless integration.
- Dimensioning and analysis module making use of the integrated algorithms and resources.
- Optical Path Computation Element for enhancing the efficiency of the network.

The connectivity between B5G-ONP and the application/service layer (operators) is through the Northbound Interface (NBI) and the ONP interacts with the rest of the control plane elements (PON SDN Controller, IP SDN Controller, TAPI Optical Network Orchestrator, and Kubernetes) by using the Southbound Interface (SBI). Figure 3-1 shows the architecture and interconnections of the B5G-ONP component.

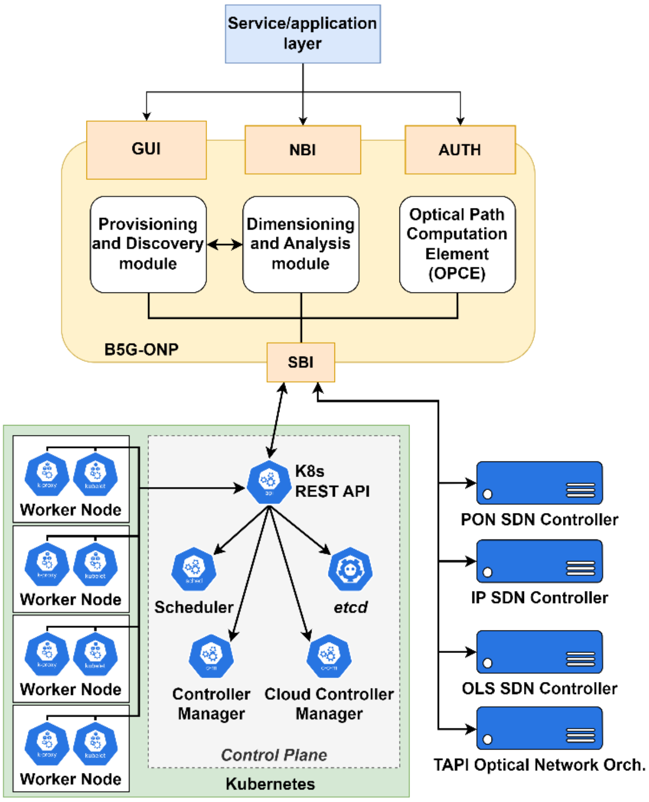


Figure 3-1 Architecture and interconnection of B5G-ONP component

3.1 Component architecture

The B5G-ONP component is a control plane module which allows the coordination and orchestration of IT and network resources. This module is endowed with an array of capabilities encompassing the provisioning of tools for designing, optimizing, and strategizing the deployment, administration, and configuration of services and resources within the network infrastructure.

In addition to these core functionalities, the B5G-ONP component will prototype a user-friendly Graphical User Interface (GUI) aiming to improve the Quality of user Experience (QoE). This GUI aims to streamline interactions with the underlying network components, ultimately contributing to an elevated level of usability and efficiency.

Regarding process automation and the concept of Zero-Touch management, the B5G-ONP component exposes a Northbound Interface (NBI) REST API. This interface designed for network operators as well as higher-level components or services. Conversely, the B5G-ONP component makes use of the Southbound Interface (SBI) to establish seamless integration with external components such as PON SDN Controller, IP SDN Controller, TAPI Optical Network Orchestrator, and Kubernetes.

The B5G-ONP component functions as a self-contained entity, that integrates diverse modules to introduce necessary functionalities and to enhance network performance by a clear understanding of the global requirements. These modules are:

- *The Provisioning and Discovery module* bears the responsibility of orchestrating the allocation of network resources, both physical and virtual, by automating deployment and configuration processes in accordance with user requests, policies, and predefined templates. This unit conducts thorough scans and imports of network segment topologies. Subsequently, it identifies the services and dependencies, consolidating this information into a centralized repository. In effect, this module emerges as a comprehensive platform, effectively managing the network environment on behalf of network administrators.
- *The Dimensioning and Analysis module* assumes the critical role of forecasting and optimising the performance of the network infrastructure. This entails making informed choices concerning capacity planning and network dimensioning. Employing a fusion of data analytics and machine learning methodologies, the B5G-ONP examines network traffic patterns and resource utilization. The goal is to prognosticate the performance requisites of the network – encompassing elements such as capacity, latency, jitter, and more. These predictions are meticulously aligned with the network's topology and application requisites.
- *The Optical Path Computation Element* evaluates technical variables to estimate the optimal route for optical connections, leveraging available resources within the B5G-ONP framework to mitigate congestion and enhance overall efficiency. The module builds upon preceding discovery efforts, estimating the performance of various admissible paths. Subsequently, the B5G-ONP undertakes an assessment of the performance of these newly identified route options, configuring the underlying resources (e.g., Optical SDN Controller) to attain the optimal setup that guarantees Quality of Transmission (QoT).

The previous modules are incorporated within the B5G-ONP ecosystem as part of the network orchestration, making easier the integration and coordination of B5G-OPEN modules and component by a unified unit. The subsequent section details the integrated interfaces within the B5G-ONP component for the exchange of the information with external modules, collaboratively developed by project partners. The B5G-ONP is implemented as part of the e-Lighthouse Network Planner [ENP].

3.2 Interfaces

As described above and shown in Section 2.11, the NBI is the northernmost interface between the B5G-ONP and the applications and services that leverage network resources through a REST API framework. The Graphical User Interface (GUI) employs the information surfaced by this interface to define an aesthetically organized representation of the network's structure. Network operators harness this interface to estimate the network requirements, involving real-time tasks such as bandwidth allocation for specific applications and traffic priority. Subsequently, the underlying controllers take charge of configuring network resources and implementing policies on network devices.

On the other hand, the SBI of the B5G-ONP serves as the juncture through which external components managed by B5G-ONP are interconnected. This interface effectively translates high-level network requests into actions and configurations within underlying layers according to directives provided through the NBI by the network operators. This interface may include the RESTCONF protocol and different standards to enable communication with other modules such as Transport API (TAPI) [TAPI 2.1.3] or IETF RFC8345 [RFC8345]. B5G-ONP's SBI empowers network automation tasks, such as real-time provisioning of new network equipment. These features significantly enhance the efficiency, availability, and dependability of management operations.

The Optical Path Computation Element module exposes an autonomous endpoints accessed by the TAPI Optical Network Orchestrator. This interconnection allows strategic design and deployment of B5G-OPEN networks through meticulous planning and provisioning.

In summary, the set of interfaces that the B5G-ONP is involved can be grouped in three main interface types, that are listed as follows:

- *B5G-ONP – Service/application layer Interface*: the B5G-OPEN offers verticals and service/application layer consumers both a graphical interface and an Open NBI API to permit the orchestration in the provisioning of network services/applications, network slices, or DSR and IP flow. Details in Section 2.11.
- *B5G-ONP – Kubernetes Interface*: the B5G-ONP will consume, from a client-side perspective, the by-default Kubernetes API to, first, retrieve the necessary IT and deployment metrics to obtain the given cluster visibility to perform deployment recommendations, and second, to command K8s in the provision of network applications consuming the computational resources available in the Kubernetes cluster. More information is available in section 2.12.
- *B5G-ONP – SDN Controllers Interface*: This interface is a groping of interfaces devoted to the communication of the B5G-ONP and all the SDN controllers in charge of the different network domains to consider in the B5G-OPEN project, like IP, PON or Optical domains. This set of interfaces is focused on retrieving network topology-related metrics and to command SDN controllers in network slice and connection provisioning. For the communication with each controller type a different protocols and APIs are envisaged, as exposed previously.

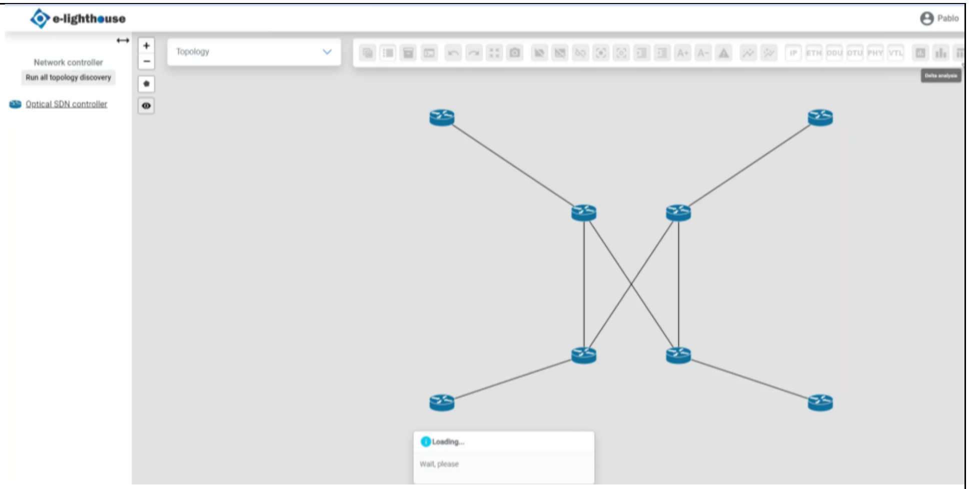
As an exemplary workflow to show how the interfaces are used, for instance, when the network operator specifies the network requirements, the B5G-ONP initiates the process of discovering network topologies across different segments. This initial exploration is followed by a thorough analysis, considering both the stipulated requirements and the available resources. planning and dimensioning tasks, informed by the results of the analysis, incorporate necessary network adjustments aimed at enhancing performance and meeting expected KPIs. The B5G-ONP then translates the selected solution into specific low-level commands or calls, effectively translating them into directives for the corresponding entities via the SBI.

Once executed these actions, the orchestrator comprehensively assesses network performance under diverse conditions to validate and ensure optimal outcomes. When external components report metrics back to the B5G-ONP, it must adeptly interpret and represent these metrics to network administrators, fostering a coherent understanding of network performance.

3.3 Functional Validations

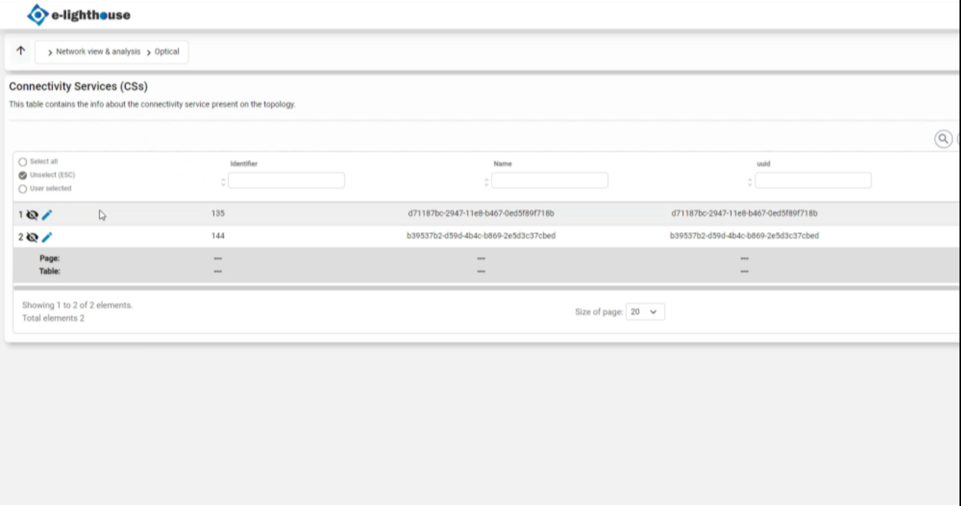
To ensure the reliability and effectiveness of the B5G-ONP component and the previously described interfaces, this section includes rigorous validation tests. The objective of these tests encompasses diverse aspects, ranging from load handling and scalability to the accuracy of discovery processes, Kubernetes cluster deployment, and analysis module proposals, as well as validating the functionalities that involve other external entities, e.g., verticals or SDN controllers. By subjecting the component to these meticulous assessments, the objective is to ensure its seamless operation under varying conditions and functionalities.

Test	**Description**
Load and Scalability Test	Verify the capabilities of the B5G-ONP component to efficiently handle anticipated user loads with minimal response times. Evaluate its ability to dynamically scale up or down in response to demand, perform concurrent operations, and maintain high efficiency even during stress scenarios.
Topology Discovery Validation	Validate the accuracy and coherence of discovery processes within both IP and optical segments. Ensure that responses provide precise information. Assess the GUI's ability to successfully import and visually represent the complete network topology. This test has been validated in EuCNC demo [EuCNC] where the scenario presented (see figure below). Once the response from TAPI Network Orchestrator is received, these data are represented graphically in the GUI’s Layout Window and sorted in tables within the GUI’s Control Window.

<table>
<tr><td></td><td>

e-lighthouse
Pablo
Network controller
Run all topology discovery
Optical SDN controller
Topology
Delta analysis

Loading...
Wait, please</td></tr>
<tr><td>Connectivity Service Provision Validation</td><td>After importing the topology into ENP tool, this allows connectivity service provisioning to be carried out on the topology. The request is sent to the underlaid module that exposes a NBI and after processing the request, it responds with a status code of the operation. This process requires send the request and check that the operation has been completed correctly.

Similar to Topology Discovery Validation, this test has also been validated in the EuCNC demo [EuCNC], where a topology has been provisioned with two Connectivity-Services (CSs) for sending traffic between two pair of nodes independently.

Connectivity Services (CSs)
This table contains the info about the connectivity service present on the topology.
<table>
<tr><th>○ Select all
● Unselect (ESC)
○ User selected</th><th>Identifier</th><th>Name</th><th>uuid</th></tr>
<tr><td>1</td><td>135</td><td>d71187bc-2947-11e8-b467-0ed5f89f718b</td><td>d71187bc-2947-11e8-b467-0ed5f89f718b</td></tr>
<tr><td>2</td><td>144</td><td>b39537b2-d59d-4b4c-b869-2e5d3c37cbed</td><td>b39537b2-d59d-4b4c-b869-2e5d3c37cbed</td></tr>
<tr><td>Page:
Table:</td><td>---
---</td><td>---
---</td><td>---
---</td></tr>
</table>
Showing 1 to 2 of 2 elements.
Total elements 2
Size of page: 20</td></tr>
<tr><td>Kubernetes Cluster Deployment Test</td><td>Deploy containers and complete applications within the Kubernetes cluster and assess their proper implementation and status using <i>kube-apiserver</i>. Evaluate the security, scalability, and resilience of the Kubernetes cluster by progressively increasing the workload.</td></tr>
<tr><td>Analysis Module Proposal Validation</td><td>Validate the proposed analysis module by comparing its outcomes against the current network conditions. Ensure that the proposed solution aligns with optimal network performance.</td></tr>
</table>

Collectively, these validation tests serve as a comprehensive assessment of the B5G-ONP component. By subjecting it to meticulous exam by diverse scenarios and functionalities, we ensure its robustness, accuracy, and efficiency. The transition to the next phase of component integrations, as well as the insights gained from these tests will lay a strong foundation for the convergence of the B5G-ONP within the project framework.

3.4 COMPONENT INTEGRATION

As the integration phase advances toward the integration phase within the B5G-OPEN architecture, the functional validations performed on the B5G-ONP component have paved the way for seamless synergies with other B5G-OPEN entities. These integration efforts not only highlight the flexibility of the component but also promise to augment the overall effectiveness of the architecture. The architecture defined includes different components that must be integrated with B5G-ONP to carry out their goals:

3.4.1 Kubernetes

Kubernetes is accessible, and B5G-ONP utilizes the exposed REST API of the *kube-apiserver* for automating the deployment, scaling, and management of applications. This approach aims to construct more efficient and scalable IT infrastructures. The endpoints provided by Kubernetes are outlined in section 4.12. These endpoints serve purposes such as resource discovery, resource provisioning, and retrieval of performance metrics within the Kubernetes cluster. Internal works to support Kubernetes within the E-Lighthouse Network Planner (as part of the B5G-ONP) controller suite is already started, and preliminary results are obtained.

3.4.2 PON SDN Controller

The PON SDN Controller enables network management and automation within this domain, utilizing the accessible tools and platforms through the NBI of the PON SDN Controller exposed by a NETCONF/REST server. This interface allows to B5G-ONP component to provision and configure PONs. It is pending to start this integration, expected in next stages of the project.

3.4.3 IP SDN Controller

The IP SDN Controller will provide an exposed REST API NBI, which the B5G-ONP will utilize to send requests to essential endpoints. These requests are made to acquire topological information about the underlying IP network, and to implement necessary configurations following a process of analysis and resource optimization within the B5G-ONP. The IP topology, imported in the B5G-ONP, is defined using RFC 8345 [RFC8345] and B5G-ONP includes the methods to interpretate it. The E-lighthouse Network Planner already supports a version of the IETF network model, capability obtained from other commercial and research projects. It is expected to perform some adjustments to adapt it to the particularities of the B5G-OPEN project.

3.4.4 TAPI Orchestrator

The TAPI Optical Network Orchestrator provides a consistent, open, and standardized method for gathering information from subsystems and sub-controllers to B5G-ONP. This controller uses TAPI 2.1.3 [TAPI 2.1.3] to export the optical topology. This integration has been proven and validated in [EuCNC] Demonstration of SDN control of disaggregated multi-band networks with externalized path computation.

3.4.5 OLS Controller

The OLS controller is based on the Ensemble Network Controller software solution and is offering a northbound ONF Transport-API (TAPI) towards the Optical Controller. The topology of the OLS Controller is based on the ONF TAPI 2.1.3 [TAPI 2.1.3] model where a topology represents all network layers such as OCH and Photonic Media. Consequently, B5G-ONP allows the import of the OLS topology from the TAPI 2.1 model. The integration works presented in the previous bullets can been seen as a starting point to integrate OLS controller in B5G-ONP.

3.5 ROADMAP

To ensure the realization of a seamlessly integrated architecture within the stipulated project timeline, the subsequent action items have been defined:

- Q4/2023: Integration of the B5G-ONP component with the Kubernetes REST API. This strategic integration aims to empower the architecture with the dynamic capabilities of Kubernetes, bolstering resource management and deployment efficiency.
- Q4/2023: Refinement of the control plane specification for inclusion in M4.4 is currently underway.
- Q1/2024: Integration and Topology Import with OLS SDN Controller. By materializing this integration, we aim to enhance our grasp over optical resources and fortify network orchestration capabilities.
- Q1/2024: Initial Integration and Testing with PON SDN Controller. These steps are poised to pave the way for a harmonious interplay between these controllers and the B5G-ONP component. This harmonization is critical in realizing an architecture that thrives on interconnectedness and adeptly adapts to dynamic networking exigencies.
- Q2/2024: Elaborate on the specifics of the control plane architecture in D4.3 to outline the network capabilities for zero-touch and autonomous management.

The proposed roadmap emphasizes strategic integration phases to achieve the incorporation of the B5G-ONP component with other components into the overarching architecture. The preceding roadmap includes diverse integration and validation tasks adapted towards improved resource management and alignment with other controllers, putting the effort into achieving an interconnected and adaptable architecture. By adhering to these strategic phases and evaluating performance through defined KPIs, the B5G-ONP component is poised to play a substantial role in propelling next-generation network paradigms within the context of the B5G-OPEN project.

3.6 COMPONENT KPIS

As the B5G-OPEN project is geared towards enhancing the efficiency and capabilities of the orchestrated network environment, it is necessary a quantitative assessment of performance and efficiency. The metrics under consideration assess the time periods and latencies associated with the discovery, analysis, sizing, and provisioning tasks executed by the B5G-ONP component. This approach enables the computation of the efficiency and responsiveness of the component, while also actively contributing to the attainment of the objectives set forth by the B5G-OPEN project. The main KPIs are detailed in Table 1:

Table 1: KPIs for B5G-ONP component

KPI	Definition	Methodology
Topology Discovery Time	The time elapsed from the moment a network operator sends the finding command until all layer of the topology is imported into B5G-ONP.	Measured from the SBI of the B5G-ONP. Target (< 5 seconds up to 30 seconds).

Analysis / dimensioning delay	The time required to run the algorithm/s covering dimensioning and/or analysis functions.	Measured from the B5G-ONP console/logs. Target (from < 1 second up to 1 minute).
Connectivity service provisioning latency	Time to provision connectivity service starting when B5G-ONP receives a request for it and ending when the service is properly established.	Measured from the B5G-ONP console/logs. Target (< 10 min with hardware and < 1 minute with emulated hardware).
Optical Path computation element latency	Refers to the time it takes to perform a path and is directly proportional to the number of network elements involved in the path computation and the traffic load on the network.	Measured in the message exchange between the two involved entities, the PCE and TAPI Orchestrator.
Scalability In terms of Number of Elements	Number of elements that can be controlled by a single instance	Setup scenarios (with emulated hardware)

3.6.1 Topology Discovery Time

The topology discovery of network resources increases the flexibility and efficiency within acceptable delay. It makes the request to the defined component (mainly SDN Controller and TAPI Network Orchestrator) to import the topology details.

The proposed solution for B5G-ONP seeks to minimize complexity and optimize efficiency by interacting between the components to operate as efficiently as possible. The topology discovery ends when the imported topology is represented within the Layout and Control Windows as mentioned previously.

Regarding the network for EuCNC demo composed of 8 nodes and 16 links (small size), the topology data were taken from T-API Network Orchestrator with a delay lower than 2 seconds. The maximum duration of this phase is expected to be 30 seconds.

Negative Factors:

- Network size
- Software complexity

3.6.2 Analysis / dimensioning delay

Although this phase may have algorithms preloaded for execution, it requires the Topology Discovery Time (see 3.6.1) to be able to apply these algorithms to the network infrastructure.

The execution of these analysis and dimensioning algorithms depends on the network and algorithm complexity on which they are applied, and it is convenient to analyse them case by case. This KPI has a significant impact on the operational efficiency of the B5G-ONP module and aims to increase the efficiency of the networks by making decisions based on accurate and up-to-date data. The optimization of these processes, based on traffic fluctuations or events, allows agile decisions that will keep the resilience of the networks and will be reflected in a significant improvement of the user experience.

3.6.3 Connectivity service provisioning latency

The provisioning requires a previous synchronization with the existing network to abstract the different nodes and Service Interface Points (SIPs) on which to deploy a Connectivity Service (CS). The synchronization process runs in the Topology Discovery Time (see 3.6.1).

This interval considers the time from the provisioning request is made until the requested CS is operative. These tasks may be carried out by a network operator or by the network orchestrator module after makes an analysis or resource optimization.

If simultaneous provisioning requests are made, the B5G-ONP module is able to manage them in parallel to minimize the required time for these tasks.

Regarding the network for EuCNC demo composed of 8 nodes and 16 links (small size), the demo continued provisioning 2 CSs. The delay experienced at this stage, from the request until receive a favourable response, was less than 2 seconds. Although the topology over which the CSs were provisioned is small, it is not expected to observe a delay greater than 10 minutes for more complex topologies.

Negative factors:

- Topology complexity
- Hardware configuration latency

3.6.4 Optical Path computation element latency

This KPI evaluates an efficient control of a high volume of devices between which it is intended to establish a path for communications. This phase requires the Topology Discovery Time (see 3.6.1) to import the information related to the network and estimates the communication latency based on certain parameters and network conditions.

The results section presents the findings based on the defined KPIs for OPCE latency such as relationship between latency and network elements, impact of traffic load and latency variability.

3.6.5 Scalability In terms of Number of Elements

This KPI measures the ability to optimize the network resources according to the network elements present in the topology. For this purpose, the aforementioned planning algorithms and AI/ML techniques are used to provide guidelines on the SDN control plane in a scenario with a large volume of devices.

The evaluation of the KPIs allows for the assessment of the performance of the B5G-ONP component. The acquired information evaluates aspects such as network topological complexity, algorithmic efficiency, and the quality of operational orchestration. The values obtained from these KPIs will prove instrumental in shaping strategic decisions within the project. Subsequently, a roadmap for the forthcoming months of the project is outlined, delineating the intended path to be followed.

4 TAPI-ENABLED OPTICAL NETWORK ORCHESTRATOR WITH EXTERNALIZED PATH COMPUTATION

4.1 INTRODUCTION

The TAPI-enabled Optical Network Orchestrator is a functional element of the architecture that is responsible for the following functions: i) providing a uniform, open and standard view and interface to the higher levels and components of the B5G-OPEN control, orchestration, and telemetry system; ii) Composing a complete context to be consumed by B5G-OPEN network planner and additional consumers combining information retrieved from subsystems and sub-controllers (Optical Controller, external databases, monitoring systems, etc), iii) Enabling single entry point for provisioning DSR and Photonic Media services, including externalized path computation and iv) providing an event telemetry data source that reports events that happen asynchronously in the network.

As shown in Figure 2-1, the TAPI-enabled Optical Network Orchestrator coordinates the provisioning of services delegating the path computation and the actual device configuration to specialized controllers.

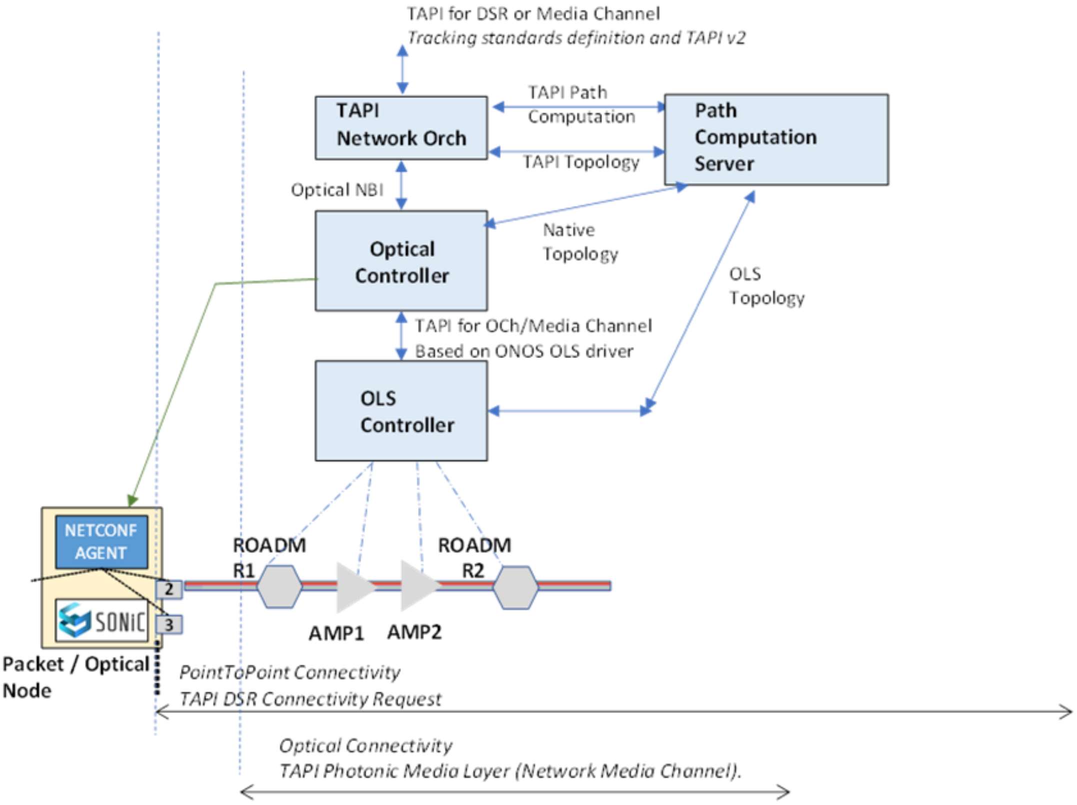


Figure 4-1 B5G-OPEN Control Plane architecture for the Optical Network Control, showing the role of the TAPI enabled Optical Network Orchestrator with Externalized Path Computation.

4.2 INTERNAL ARCHITECTURE

The core of the TAPI Optical Network Orchestrator controller is an asynchronous event loop. On the one hand, it exports multiple services via its multiple North Bound Interfaces (NBI) to users or clients, using RESTCONF/YANG. The most relevant services are Topology Management, Connectivity Service Management and Path Computation.

The RESTCONF server is responsible for processing requests using the RESTCONF protocol. The Yang models are a subset of the ONF TAPI v2.1. Service requests are mapped to internal structures and processed by functions in the event manager. The TAPI-enabled Optical Network Orchestrator is a multi-threaded application, written in C++. It targets GNU/Linux systems (e.g., Ubuntu 20.04 and later) and can be executed as docker containers. The design is highly modular, so additional functionality can be implemented as shared link libraries that can be configured and loaded on demand. The block design of the controller can be seen in the Figure below (Figure 4-2), showing its key North Bound Interfaces (TAPI Connectivity, TAPI Topology and TAPI Path Computation) and telemetry data source using Redis (see Telemetry Section) as well as its East Bound Interface towards the Path Computation Function

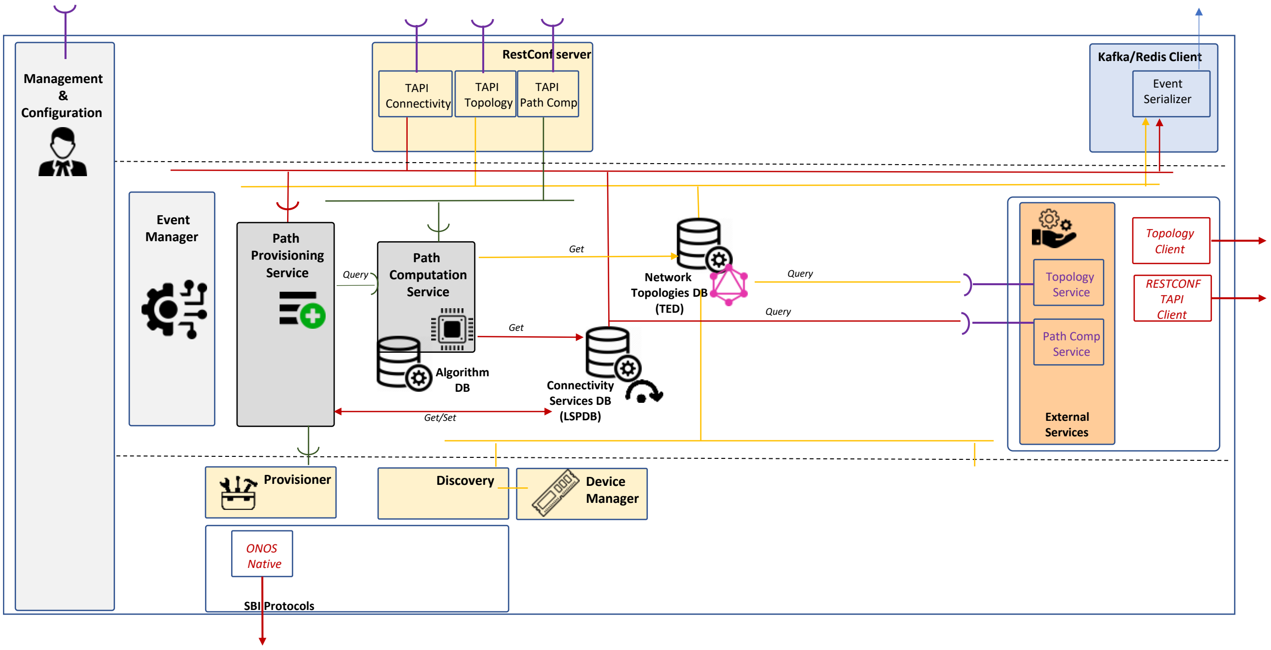
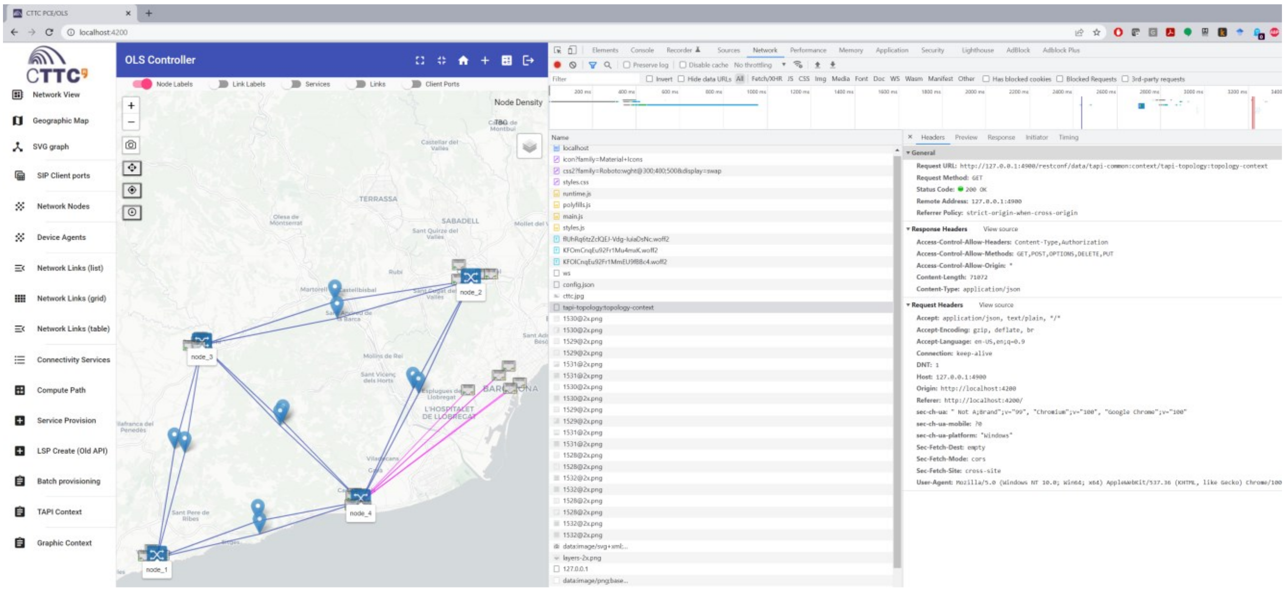


Figure 4-2 B5G Internal module architecture of the B5G-OPEN TAPI enabled Optical Network Orchestrator with Externalized Path Computation

Along with the core TAPI orchestrator, a Graphical User Interface (GUI) is provided to operate the underlying network. In this sense, the TAPI orchestrator can be used by a human operator, in addition to other entities consuming the NBI.

The GUI (see Figure 4-3) is implemented as a Web Application, using the Angular framework. It can be used to visualize the network topology and associated resources, the status of the links and to perform operations such as path computation or service provisioning.



General
Request URL: http://127.0.0.1:4900/restconf/data/tapi-common:context/tapi-topology:topology-context
Request Method: GET
Status Code: 200 OK
Remote Address: 127.0.0.1:4900
Referrer Policy: strict-origin-when-cross-origin

Response Headers View source
Access-Control-Allow-Headers: Content-Type,Authorization
Access-Control-Allow-Methods: GET,POST,OPTIONS,DELETE,PUT
Access-Control-Allow-Origin: *
Content-Length: 71072
Content-Type: application/json

Request Headers View source
Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Connection: keep-alive
DNT: 1
Host: 127.0.0.1:4900
Origin: http://localhost:4200
Referer: http://localhost:4200/
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: cross-site
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100

Figure 4-3 the Graphical User Interface of the TAPI enabled Optical Network Orchestrator with Externalized Path Computation

4.3 Interface Specification

This section lists the Interfaces that have been implemented in the TAPI Network Orchestrator, as described in Section 2.

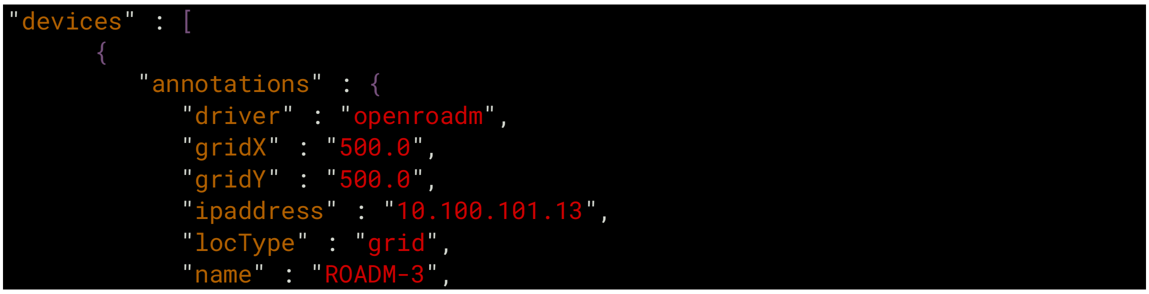
4.3.1 North Bound Interface (NBI) towards the B5G-ONP

The TAPI Optical Network Orchestrator exports multiple services via its multiple North Bound Interfaces (NBI) to users or clients, using RESTCONF/YANG. The most relevant services are Topology Management, Connectivity Service Management and Path Computation. This interface is based on TAPI (see Section 2.3).

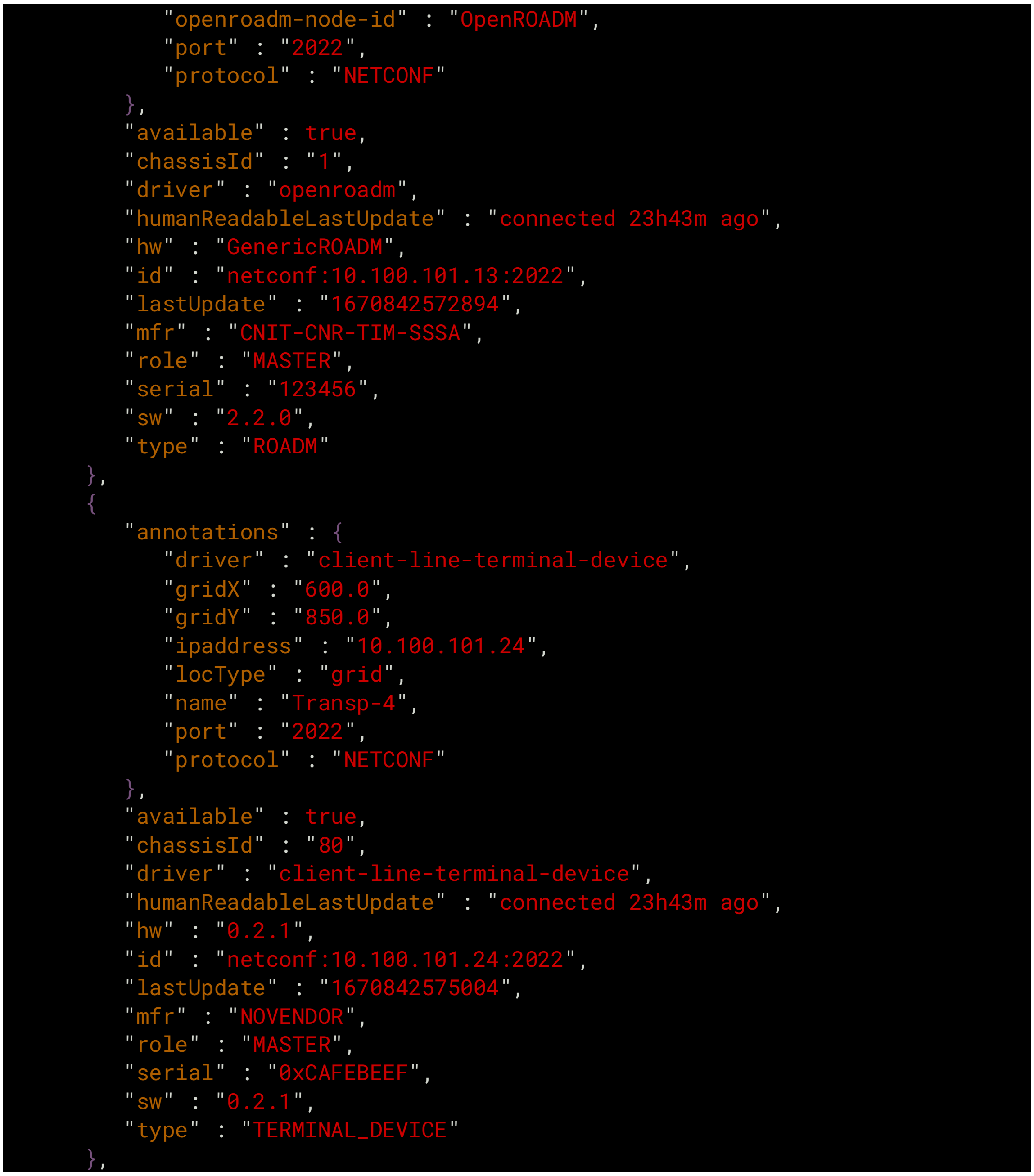
In particular, the TAPI Optical Network Orchestrator receives a new connection request with some requirements (source and destination, bandwidth provision, latency constraints, QoT conditions, etc.).

4.3.2 South Bound Interface (SBI) towards ONOS Native Interface

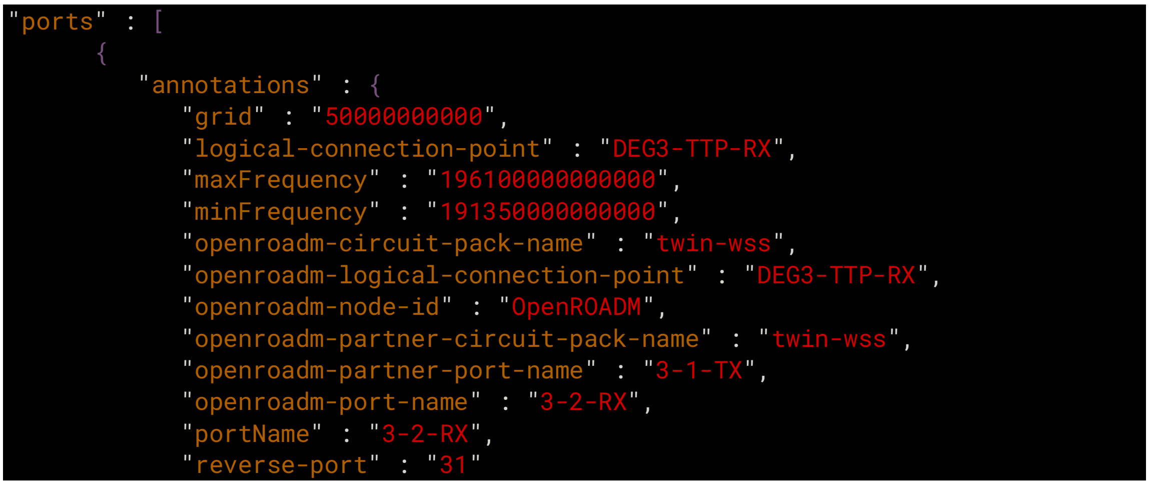
The interface from the TAPI Optical Network Orchestrator to the optical controller has been implemented based on the ONOS native interface, extending the existing implementation to support additional requirements and use cases. The figures below show examples of the topology retrieved by the TAPI orchestrator from the underlying ONOS in demonstrations carried out in EuCNC’23 and ECOC’23.



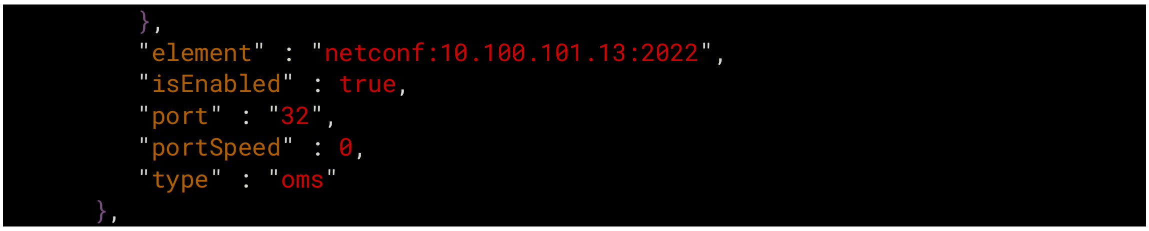
```
"devices" : [
      {
         "annotations" : {
            "driver" : "openroadm",
            "gridX" : "500.0",
            "gridY" : "500.0",
            "ipaddress" : "10.100.101.13",
            "locType" : "grid",
            "name" : "ROADM-3",
```



```
            "openroadm-node-id" : "OpenROADM",
            "port" : "2022",
            "protocol" : "NETCONF"
         },
         "available" : true,
         "chassisId" : "1",
         "driver" : "openroadm",
         "humanReadableLastUpdate" : "connected 23h43m ago",
         "hw" : "GenericROADM",
         "id" : "netconf:10.100.101.13:2022",
         "lastUpdate" : "1670842572894",
         "mfr" : "CNIT-CNR-TIM-SSSA",
         "role" : "MASTER",
         "serial" : "123456",
         "sw" : "2.2.0",
         "type" : "ROADM"
      },
      {
         "annotations" : {
            "driver" : "client-line-terminal-device",
            "gridX" : "600.0",
            "gridY" : "850.0",
            "ipaddress" : "10.100.101.24",
            "locType" : "grid",
            "name" : "Transp-4",
            "port" : "2022",
            "protocol" : "NETCONF"
         },
         "available" : true,
         "chassisId" : "80",
         "driver" : "client-line-terminal-device",
         "humanReadableLastUpdate" : "connected 23h43m ago",
         "hw" : "0.2.1",
         "id" : "netconf:10.100.101.24:2022",
         "lastUpdate" : "1670842575004",
         "mfr" : "NOVENDOR",
         "role" : "MASTER",
         "serial" : "0xCAFEBEEF",
         "sw" : "0.2.1",
         "type" : "TERMINAL_DEVICE"
      },
```



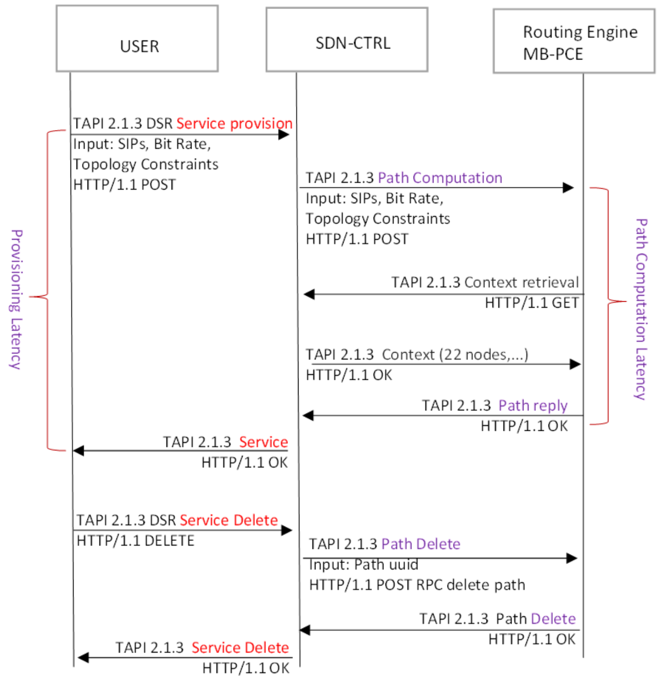
```
"ports" : [
      {
         "annotations" : {
            "grid" : "50000000000",
            "logical-connection-point" : "DEG3-TTP-RX",
            "maxFrequency" : "196100000000000",
            "minFrequency" : "191350000000000",
            "openroadm-circuit-pack-name" : "twin-wss",
            "openroadm-logical-connection-point" : "DEG3-TTP-RX",
            "openroadm-node-id" : "OpenROADM",
            "openroadm-partner-circuit-pack-name" : "twin-wss",
            "openroadm-partner-port-name" : "3-1-TX",
            "openroadm-port-name" : "3-2-RX",
            "portName" : "3-2-RX",
            "reverse-port" : "31"
```

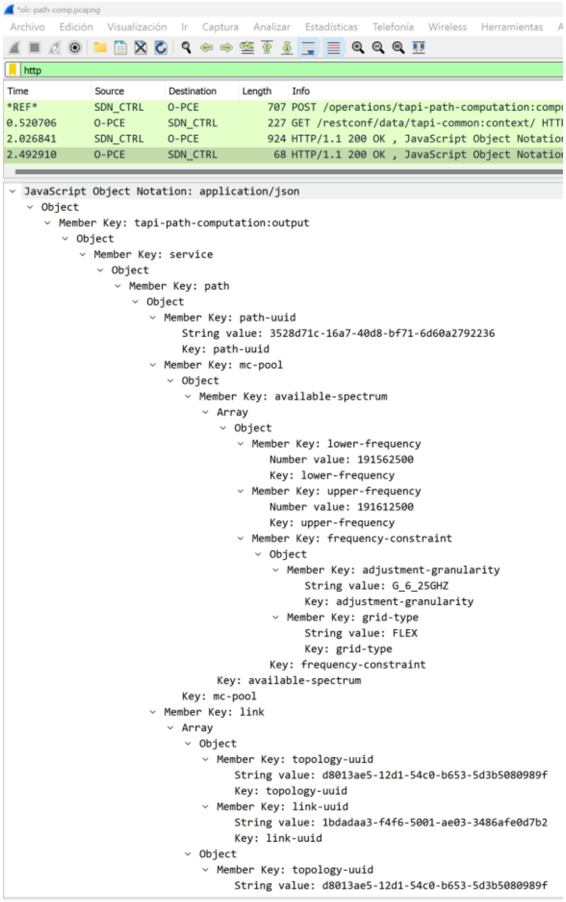


```
          },
          "element" : "netconf:10.100.101.13:2022",
          "isEnabled" : true,
          "port" : "32",
          "portSpeed" : 0,
          "type" : "oms"
      },
```

4.3.3 Interface towards the Path Computation Element (PCE)

The interface from the TAPI Optical Network Orchestrator to the path Computation Engine is based on a specific instance of path computation interface defined in TAPI (as detailed in Section 2.3 for Path Computation). Figure 4-4 shows both the workflow and Wireshark capture of the exchanges between the TAPI Orchestrator and the O-PCE. Regarding the workflow, the operator requests a Digital Signal Rate (DSR) service between transceiver’s client ports. The request contains the identifiers of the Service Interface Points (SIPs, transceiver client ports); the requested bit rate (e.g., 100G, 200G, 400G) and any applicable routing and topological constraints. The SDN controller delegates the computation to the externalized routing engine of the PCE that explores the PLI-aware RSMA platform. Upon successful completion, it provides the number of required OTSi and, for each OTSi, the path in terms of links, the number of frequency slots and the selected transmission system parameters (note that in this work we have considered single OTSi). The SDN controller instantiates the DSR, OTSi and MC layers connection and connection end point (CEPs) objects, it configures the transceivers’ operational modes and the media channels in the ROADM devices via the SBI interface.




*olc-path-comp.pcapng
Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas

```
http

Time      Source    Destination  Length Info
*REF*     SDN_CTRL  O-PCE           707 POST /operations/tapi-path-computation:compu
0.520706  O-PCE     SDN_CTRL        227 GET /restconf/data/tapi-common:context/ HTT
2.026841  SDN_CTRL  O-PCE           924 HTTP/1.1 200 OK , JavaScript Object Notatio
2.492910  O-PCE     SDN_CTRL         68 HTTP/1.1 200 OK , JavaScript Object Notatio

⌄ JavaScript Object Notation: application/json
  ⌄ Object
    ⌄ Member Key: tapi-path-computation:output
      ⌄ Object
        ⌄ Member Key: service
          ⌄ Object
            ⌄ Member Key: path
              ⌄ Object
                ⌄ Member Key: path-uuid
                    String value: 3528d71c-16a7-40d8-bf71-6d60a2792236
                    Key: path-uuid
                ⌄ Member Key: mc-pool
                  ⌄ Object
                    ⌄ Member Key: available-spectrum
                      ⌄ Array
                        ⌄ Object
                          ⌄ Member Key: lower-frequency
                              Number value: 191562500
                              Key: lower-frequency
                          ⌄ Member Key: upper-frequency
                              Number value: 191612500
                              Key: upper-frequency
                          ⌄ Member Key: frequency-constraint
                            ⌄ Object
                              ⌄ Member Key: adjustment-granularity
                                  String value: G_6_25GHZ
                                  Key: adjustment-granularity
                              ⌄ Member Key: grid-type
                                  String value: FLEX
                                  Key: grid-type
                              Key: frequency-constraint
                        Key: available-spectrum
                    Key: mc-pool
                ⌄ Member Key: link
                  ⌄ Array
                    ⌄ Object
                      ⌄ Member Key: topology-uuid
                          String value: d8013ae5-12d1-54c0-b653-5d3b5080989f
                          Key: topology-uuid
                      ⌄ Member Key: link-uuid
                          String value: 1bdadaa3-f4f6-5001-ae03-3486afe0d7b2
                          Key: link-uuid
                    ⌄ Object
                      ⌄ Member Key: topology-uuid
                          String value: d8013ae5-12d1-54c0-b653-5d3b5080989f
```

Figure 4-4 the Graphical User Interface of the TAPI enabled Optical Network Orchestrator with Externalized Path Computation

4.3.4 Interface towards the Telemetry System

The interface towards the Telemetry System relies on acting as a REDIS client sending telemetry information following the TAPI Reference Implementation Agreement (RIA) for streaming TR-548. This interface has been listed in Section 2.7

4.4 Component Integration

The TAPI-enabled optical network orchestrator integrates with the following elements:

4.4.1 B5G-ONP

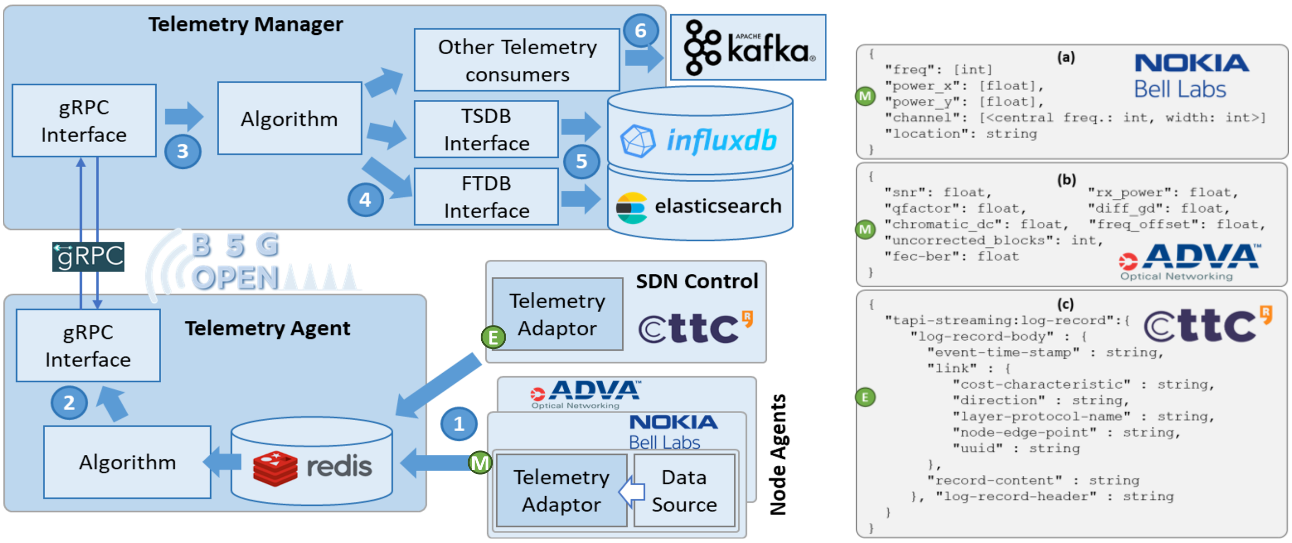
The B5G-ONP: This functional entity is the main client of the controller. The B5G-ONP performs requests related to service provisioning in the optical network, using TAPI and requesting DSR connectivity services. This has been shown in [EuCNC] Demonstration of SDN control of disaggregated multi-band networks with externalized path computation.

4.4.2 PCE

The Optical Path Computation Element (PCE): The network orchestrator relies on a dedicated system for externalized path computation. For this, it uses extended TAPI interfaces for the purposes of topology discovery and path computation functions. This is shown in [EuCNC] Demonstration of SDN control of disaggregated multi-band networks with externalized path computation and [RCasOFC22]

4.4.3 Telemetry System

The Telemetry system: Acts as a data source for the reporting of events. This means that the TAPI orchestrator sends JSON encoded telemetry data to clients, such as OSS or data visualizers [OFCDemo]



```
{
  "freq": [int]
  "power_x": [float],
  "power_y": [float],
  "channel": [<central freq.: int, width: int>]
  "location": string
}
```

```
{
  "snr": float,                "rx_power": float,
  "qfactor": float,            "diff_gd": float,
  "chromatic_dc": float,       "freq_offset": float,
  "uncorrected_blocks": int,
  "fec-ber": float
}
```

```
{
  "tapi-streaming:log-record":{
     "log-record-body" : {
       "event-time-stamp" : string,
       "link" : {
          "cost-characteristic" : string,
          "direction" : string,
          "layer-protocol-name" : string,
          "node-edge-point" : string,
          "uuid" : string
       },
       "record-content" : string
     }, "log-record-header" : string
  }
}
```

Figure 4-5 Distributed architecture supporting intelligent Optical measurement Aggregation and Streaming Event Telemetry [Vel23]

This is shown in Distributed architecture supporting intelligent Optical measurement Aggregation and Streaming Event Telemetry (Figure 4-5).

4.4.4 ONOS SDN controller

The ONOS SDN controller: Takes care of provisioning connectivity by means of optical connectivity intents, using a dedicated interface. The interface shall be augmented to support the specification of a computed path (in terms of links as well as frequency ranges for the optical media channel to be used).

4.5 Functional Validations

The following tests is a non-exhaustive list of tests that have been done to validate the component:

4.5.1 TAPI Orchestrator / SDN Controller Integration, Discovery

- **Description:** Launch the TAPI orchestrator and retrieve the topology in terms of nodes and links and display this information. This test shall be carried out by i) loading the information from a set of JSON files that have previously been retrieved and ii) performing dynamic loading of links and related data from the ONOS instance.

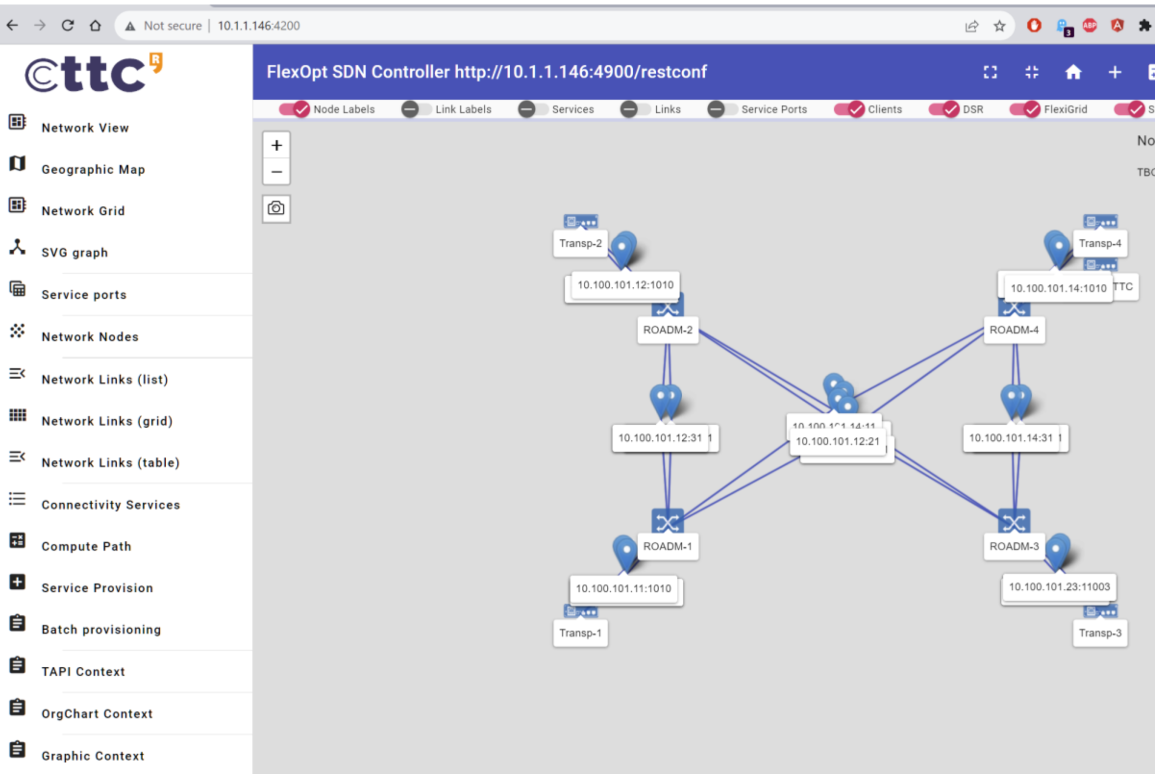


Figure 4-6 TAPI Network Orchestrator showing the topology retrieved from ONOS

The TAPI Orchestrator has been integrated with the ONOS controller and we have retrieved the ONOS topology, via a VPN tunnel connecting CTTC and CNIT premises. The latency in this case is largely affected by the propagation delay between the hosts and related overhead. Based on the logs, shown next, we can estimate the initial latency and synchronization:

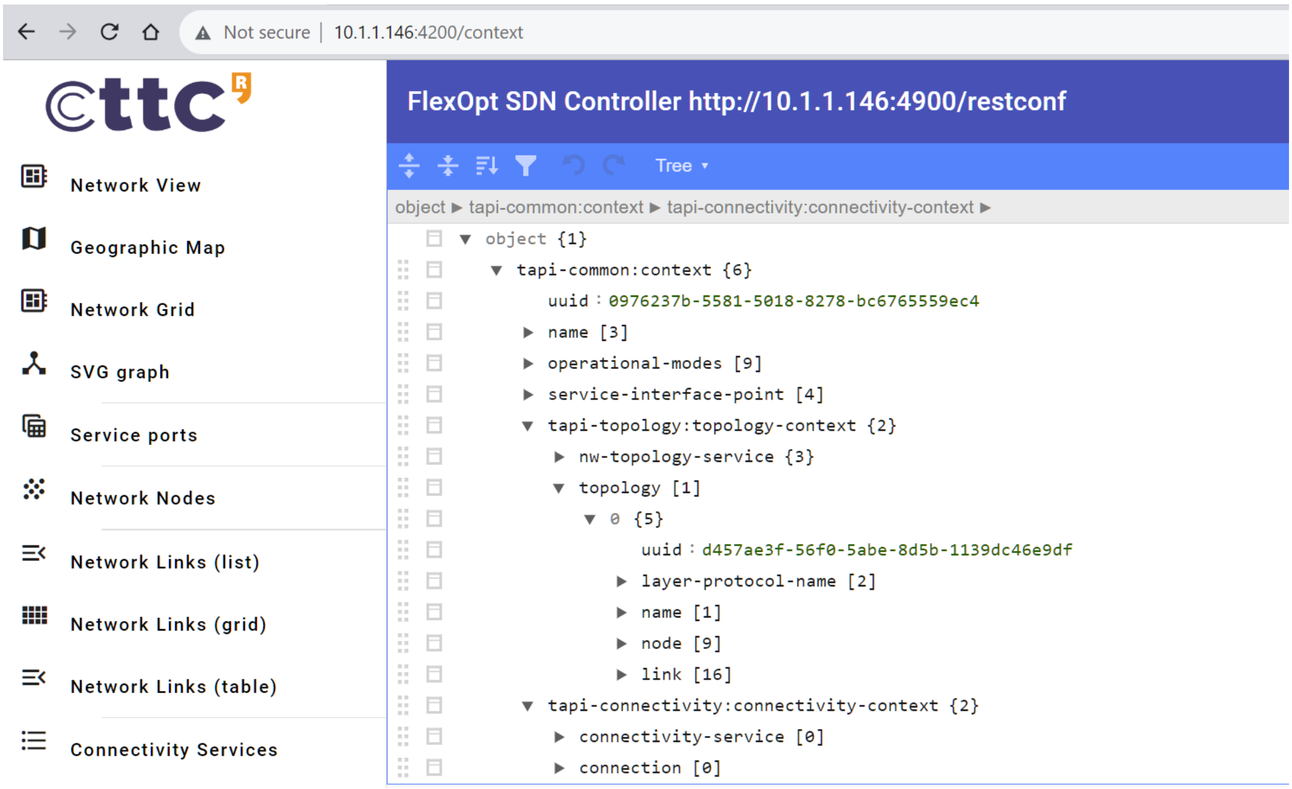
```
[INF] T   0 [2023-09-14 09:35:09.892694994]  CTTC TAPI Network Orchestrator v20.6.4.0 ---
[INF] T   0 [2023-09-14 09:35:09.892932766]  Exec: /adnet/cttc-pce-20.6.4/bin/pced
[INF] T   0 [2023-09-14 09:35:09.892994755]  Conf: pced.conf
(...)
[DBG] T 126 [2023-09-14 09:35:09.950986591]  [ONOS_WRAPPER]: retrieve
http://172.19.18.89:8181/onos/v1/links
(...)
[DBG] T 126 [2023-09-14 09:35:10.931894182]  [ONOS_WRAPPER] retrieved 9 operational modes
(...)
[DBG] T 126 [2023-09-14 09:35:10.935659537]  [ONOS_WRAPPER]: GET
http://172.19.18.89:8181/onos/v1/devices/netconf%3A10.100.101.13%3A2022/ports
[DBG] T 126 [2023-09-14 09:35:11.917683018]  [ONOS_WRAPPER]: checking channels...
[DBG] T 126 [2023-09-14 09:35:11.917753660]  [ONOS_WRAPPER]: GET
http://172.19.18.89:8181/onos/optical/links/perBandChannels
[VRB] T 126 [2023-09-14 09:35:12.565778947]  [ONOS_WRAPPER]: total number of registered_channels 132
[-1371..467] [184531250000000 196018750000000
~
```

Initial TAPI Orch / ONOS Controller synchronization: O(3s) [via VPN/Internet, debug mode and logging]

4.5.2 TAPI Orchestrator Context Retrieval and Service Provisioning

- **Description:** Retrieve the TAPI context from the TAPI orchestrator and validate the topology in terms of nodes and links. Validate that the TAPI context is correct and consistent and can be consumed by: i) B5G-ONP clients as well as ii) Path Computation Elements

The GUI Can be used to verify that the Context can be retrieved and to request service provisioning as detailed earlier in the document.



```
▼ object {1}
  ▼ tapi-common:context {6}
      uuid : 0976237b-5581-5018-8278-bc6765559ec4
    ▶ name [3]
    ▶ operational-modes [9]
    ▶ service-interface-point [4]
    ▼ tapi-topology:topology-context {2}
      ▶ nw-topology-service {3}
      ▼ topology [1]
        ▼ 0 {5}
            uuid : d457ae3f-56f0-5abe-8d5b-1139dc46e9df
          ▶ layer-protocol-name [2]
          ▶ name [1]
          ▶ node [9]
          ▶ link [16]
    ▼ tapi-connectivity:connectivity-context {2}
      ▶ connectivity-service [0]
      ▶ connection [0]
```

Figure 4-7 TAPI Network Orchestrator showing the TAPI context

4.5.3 REDIS Integration with B5G-OPEN Telemetry System

- **Description:** Load a topology and validate that the TAPI orchestrator is able to report Telemetry data to the REDIS database that is part of the Telemetry System.

  This has been carried out in the context of a demonstration in OFC.

4.5.4 Externalized Path Computation

- **Description:** Perform an externalized path computation and validate the function using an external PCE with TAPI enabled interface.

This section details the externalized path computation, supported over a distributed control plane testbed that consist of the CTTC TAPI Orchestrator and OLC-E's MB-PCE (with IP, e.g., 10.8.0.6) which are connected via tunnels over the public Internet. The emulated network is BT's optical mesh that consists of 22 ROADM nodes, 56 amplifiers, 28 terminal devices (106 network elements in total) and 238 unidirectional links. For the scope of this experiment, it is assumed that each link may support E, S, C and L bands. We have developed hypothetical scenarios where multiple connection requests need to be provisioned between the node_ROADM1 and the node_ROADM2 where computational task to the MB-PCE which, upon request, retrieves (if needed) the network topology and the list of active services/connections. Subsequently, the MB-PCE executes the PLI-aware RMSA algorithm and for each of these requests it returns to the optical SDN controller the selected path as well as the configuration details and their operational parameters for each network element along this link.

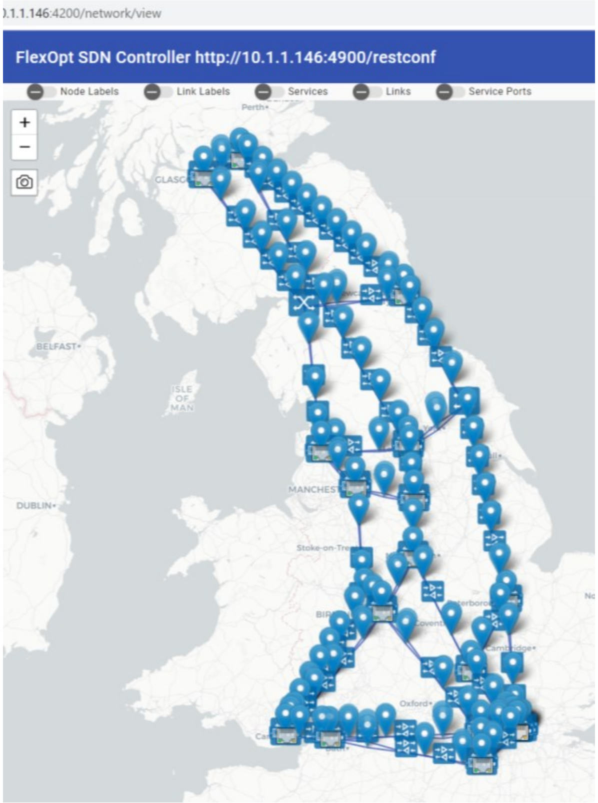


Figure 4-8 BT 22-ROADM node used in Path computation KPI validation.

For such scenarios, the MB-PCE is instructed to iterate available bands with the same order of preference as {C, E, L, S1, S2} so if the C-band is not available, the routing engine proceeds to the second best which is the E-band and so on. We wish to investigate the effectiveness of the MB-PCE and its behaviour/response against legitimate and illegitimate instructions from the OMB SDN controller. The latter could be misconfigurations or even deliberate/malicious acts.

Scenario A: the 400G DSR service between two transceivers at the source-destination nodes, node_101 and node_102 is indicated by means of the corresponding Service Interface Points (SIPs) - the 400G line-rate is based on a 16QAM, 48 Gbaud symbol-rate transceivers. The selected powers render the C-band unusable as the RMSA algorithm returns no route for the connection requests. As a result, the algorithm proceeds to assess the potential of the E-band to support the connection service (second band in the list) and it, indeed, grants these requests to the E-band. It is critically important for network operators that no misconfigured paths for the connection requests are selected.

Scenario B: The procedure of Scenario A is repeated and the optical SDN controller sends sequentially 3 service requests but now, the MB-PCE is instructed to launch the power of the C-band channels at +0.6 dBm per channel (the E-band uses the optimal launch power). The first two requests are granted in the C-band albeit the PLI-aware RMSA algorithm selects a larger guard-band between the channels serving these two successive requests. In particular, the two requests are allocated to channels spectrally space by 162.5 GHz (or 13 FSUs) which corresponds to over three times the channel's optical bandwidth. At such wide guard-band, the effect of XCI is minimized.

Scenario C: the same processes are repeated but this time the optical SDN controllers sends to the MB-PCE, five service requests sequentially designating a -3.7 dBm launch power for all of them which is with acceptable range of powers (albeit non-optimal). Given the light traffic load, the PLI-aware RMSA does not observe any OSNIR degradation, so all service requests are granted spectrally successive channels in the C-band. Also, of interest is to measure the response time (latency) needed to complete the provisioning processes in the three scenarios. These messages include the messages sent by the optical SDN controller to the MB-PCE related to the TAPI operations to synchronize and update the MB-PCE on the current network status. It is pointed out that as part of further optimization of the control-plane operations, only the initial synchronization is mandatory; beyond that, only the synchronization for the status of the active services is necessary.

4.6 Roadmap

At the time of writing, the key features for the TAPI Network Orchestrator have been introduced and validated with bilateral integrations. In the remaining of WP4, the ongoing activities are:

- New extensions as needed in view of new studies to be carried out.
- Extensions for the multi-domain in a transparent setting.
- Integration with data plane in the scope of WP5

4.7 Component KPIs

This section details the KPIs that have been defined for the component.

KPI	Definition	Methodology
Service Provisioning Latency	This is the time it takes to provision an optical service. WP4 scope to be detailed with emulated hardware.	Measured from the NBI. Target (< 10 min with hardware and < 30 seconds with emulated hardware).
Service Provisioning Overhead	In terms of messages, message size, encoding, etc. This includes a characterization of the protocol overhead (e.g., HTTP, RESTCONF, etc).	Measured from the NBI. Target (< 10 min with hardware and < 30 seconds with emulated hw).
Path Computation Latency	Measured as the time it takes to perform a path computation with a dedicated PCE	Measured in the message exchange between the two involved entities.
Scalability In terms of	Number of elements that can be controlled by a single instance	Setup scenarios (with emulated hardware)

Number of Elements		
Provisioning Rate	Number of connections that can be served.	Setup scenarios (with emulated hardware)

4.7.1 Service Provisioning Latency

As baseline, we consider [Metro-Haul] serialized provisioning targeting, as KPI the order of minutes. We can achieve Setup Time Reduction on the average setup time of connectivity service by 30% compared to serialized provisioning, exploiting approaches relying on parallelism and concurrency at the expenses of more complex workflows. This can be achieved depending on whether hardware configuration is slow.

This period varies from seconds to several minutes according to several factors as the network size, complexity, and efficiency of the orchestrator. With existing prototypes, service provisioning without data plane integrating key control plane elements, service provisioning can *achieve sub-second provisioning*. With no hardware latency and with simplified path computation algorithms (e.g., a k-shortest path with spectrum continuity) O(nanoseconds / milliseconds)

Order of Seconds / Minutes when considering PLI PCE (EuCNC demo / JOCN paper)

Negative Factors

- Hardware configuration latency
- Algorithm complexity
- Control Network dimensioning (latency)

4.7.2 Service Provisioning Overhead

The overhead is contemplated in terms of RESTCONF messages for a service provisioning (see Section 4). For a service provisioning, example:

POST http://127.0.0.1:4900/restconf/data/tapi-common:context/tapi-connectivity:connectivity-context

```
{
   "tapi-connectivity:connectivity-service" : [
      {
         "connectivity-constraint" : {
            "connectivity-direction" : "BIDIRECTIONAL",
            "requested-capacity" : {
               "total-size" : {
                  "unit" : "GBPS",
                  "value" : 400
               }
            }
         },
         "direction" : "BIDIRECTIONAL",
         "end-point" : [
            {
               "direction:" : "BIDIRECTIONAL",
               "layer-protocol-name" : "DSR",
               "layer-protocol-qualifier" : "LAYER_PROTOCOL_QUALIFIER_UNSPECIFIED",
               "local-id" : "8eb843ff-ea74-5b5d-a8c0-661673b6c823",
               "service-interface-point" : {
                  "service-interface-point-uuid" : "8eb843ff-ea74-5b5d-a8c0-661673b6c823"
               }
            },
            {
               "direction:" : "BIDIRECTIONAL",
               "layer-protocol-name" : "DSR",
               "layer-protocol-qualifier" : "LAYER_PROTOCOL_QUALIFIER_UNSPECIFIED",
               "local-id" : "7c09f268-6b42-5c30-b02a-b8b54a0ff7c5",
               "service-interface-point" : {
                  "service-interface-point-uuid" : "7c09f268-6b42-5c30-b02a-b8b54a0ff7c5"
               }
            }
         ],
         "layer-protocol-name" : "DSR",
         "layer-protocol-qualifier" : "LAYER_PROTOCOL_QUALIFIER_UNSPECIFIED",
         "requested-capacity" : {
            "total-size" : {
               "unit" : "GBPS",
               "value" : 400
            }
         },
         "route-objective-function" : "26000",
         "uuid" : "d71187bc-2947-11e8-b467-0ed5f89f718b"
      }
   ]
}
```

```
Response: 201 Created
Location
```

The overhead is mostly of the order of a few Kbytes. In case the client needs to retrieve the context, the size may increase to a few hundreds of Kbytes or Megabytes (depending on network size). This exchange takes on the order of a few milliseconds and the overall latency is mostly determined by the lower layers and the data plane.

4.7.3 Path Computation Latency

The PCE latency for scenario C was measured to be in the range between 1.8 – 2.2 seconds. This value depends on whether the PCE is also tasked to retrieve the network topology as explained above (first request in the series). On the contrary, this latency is two orders of magnitude lower, ranging between 17ms and 36 ms, when the network status was already UpToDate up to date (other requests assuming state synchronization). In contrast, for scenarios A and B, the deliberate degradation of the OSNIR rendered a large number of the listed frequency slots as 'unavailable/void', so the RMSA algorithm had to execute the PHY layer validation process thousands of times resulting to considerable delays for the MB-PCE to return results. For these two scenarios A and B, the latency is measured to be between 2.5 and 3.2 seconds.

To meet such a KPI, let us note the importance to incorporate to the routing engine a PLI-aware RMSA based on closed-form expression as this is the only way to attain service provisioning within a time frame compatible to the dynamicity of the events. This is particularly true as the projected capacity for the F6G transportation network would entail several hundred connection requests. Moreover, to ensure the scalability, it is crucial to use rapidly converging algorithms to determine the optimization launch power per band or even per channel.

Path Computation Latency for a 22-node network O(10s)

4.7.4 Scalability in Number of Elements

This KPI is related to K7.3 10x number of controlled devices, based on advanced SDN deployments with microservice-based lightweight virtualization and hierarchical arrangements and device / node abstraction. Metro-Haul targeted SDN Control of Scenarios of the order O(10) elements.

B5G-OPEN TAPI Network orchestrator is able to deal with a large number of nodes and links. For example, in the topology below represents Telefonica photonic topology. It consists of 108 nodes. 376 bundles (with 1, 2, 4 or 8 links per bundle) and over 100k client ports. In this case, the scalability issue is defined by the number of active Netconf connections to the devices.

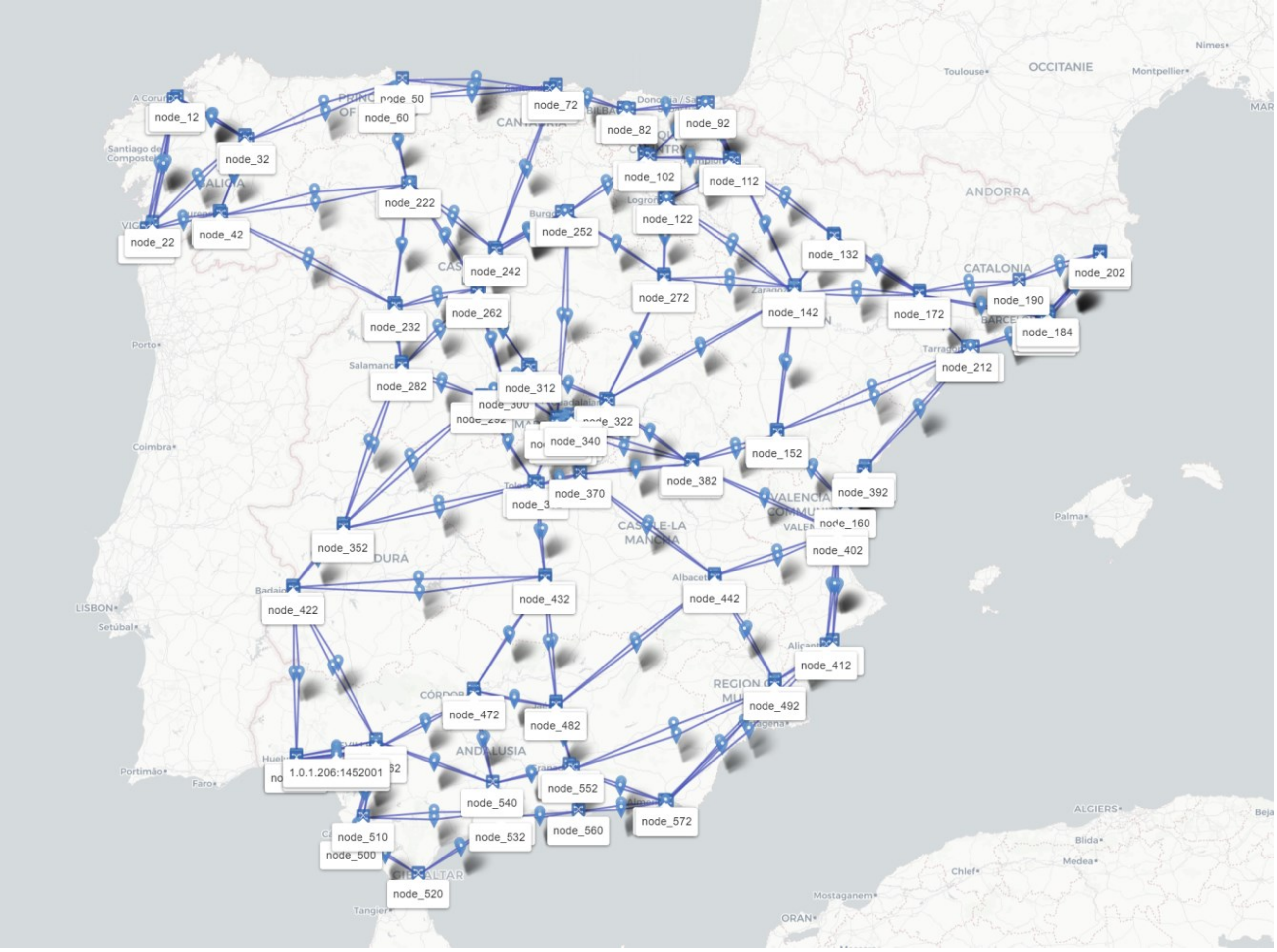


Figure 4-9 Telefonica topology to validate scalability.

4.7.5 Provisioning Rate

This KPI is related to Service Provisioning (with same theoretical assumptions). The control plane can theoretically support hundreds of connection requests per second for small (<10-100) network sizes. Metro-Haul targeted SDN did not cover high-rate provisioning, and requests were manually triggered.

Negative Factors

- Hardware configuration latency
- Algorithm complexity
- Control Network dimensioning (latency)

In general, we can only guarantee sub-second provisioning in very specific scenarios. Updated Target in the order of seconds/minutes

5 Path Computation Elements – PCE

5.1 Introduction

The Multi-Band Path Computation Engine (MB-PCE) (cf. Figure 5-1)is based on a multi-band routing engine which ensures that: i) routing is implemented by means of an efficient spectrum and modulation-format assignment; and ii) the impact of physical layer effects over the selected optical paths is estimated and the results are benchmarked against QoT target values (BER, OSNIR, OSNR, etc). In this way, the planning tool ascertains the conditions that maximize the total capacity of the network while it minimizes the global blocking probability and prevents network misconfiguration.

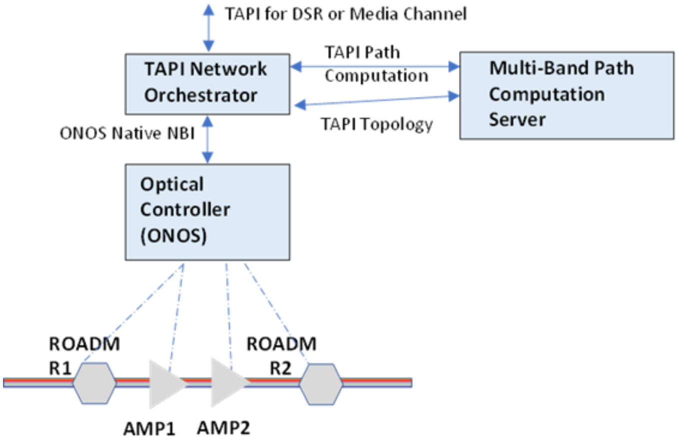

TAPI for DSR or Media Channel
TAPI Network Orchestrator
TAPI Path Computation
TAPI Topology
Multi-Band Path Computation Server
ONOS Native NBI
Optical Controller (ONOS)
ROADM R1
ROADM R2
AMP1
AMP2

Figure 5-1 B5G-OPEN Control Plane architecture for the Optical Network Control, showing the role of the Multi-Band Path Computation Engine (MB-PCE).

5.2 Internal Architecture

The MB-PCE functionality is realised in three stages as follows, while is illustrated in Figure 5-2.

STAGE-I: **Network Topology Implementation:** the network topology is defined by setting the connectivity pattern between the nodes and the traffic matrix. Next, the k-shortest paths for all network node pairs are derived. More specifically, in this step, the following quantities are defined: the network topology including nodes, edges and amplifiers, the available optical bands, the capacity per band, the traffic matrix, the average time duration of the demands and the average inter-arrival time between two consecutive demands, as well as the available line-rates and their distribution on the demands.

STAGE–II: **Spectral and Modulation Assignment (SMA) and PL entanglement:** the operation is completed in two steps: In the first step, i) a preliminary spectrum and modulation format assignment (SMA) is made for a number of the k-shortest paths, and ii) the Optical Signal to Noise plus Interference Ratio (OSNIR) for these shorter paths is estimated taking into account the impact of the physical layer effects by means of closed-form expressions.

In the second step, the Optical Multi-band Physical Layer Aware Routing Modulation and Spectral Assignment (OMB-PLA-RMSA) algorithm either selects or rejects a lightpath. A path is rejected if a) no contiguous spectral slots are available in any optical band to support the end-to-end connection, b) either the OSNIR of the candidate lightpath falls short of the QoT estimator threshold or the OSNIR of at least one of the already established lightpaths would perform below the QoT threshold due to the presence of this candidate lightpath. In either (a), (b) cases, the rejected lightpath is assigned the next available path from the sorted list of k-shortest paths and it is then re-iterated. If these paths are all rejected, the first step is repeated using a lower cardinality SMA values. If no path is retained, the engine registers a blocking condition.

STAGE–III: **Path Allocation:** This is the stage where the lightpaths are established in the network. The final assessment on network's throughput is completed and a lightpath is successfully set if contiguous spectral slots are available over the end-to-end transparent path with acceptable physical layer performance (above the QoT estimator threshold). The successful establishment of a lightpath triggers the update of the corresponding arrays for each link of the path, e. g., arrays of power, modulation format, consumed frequency slots.

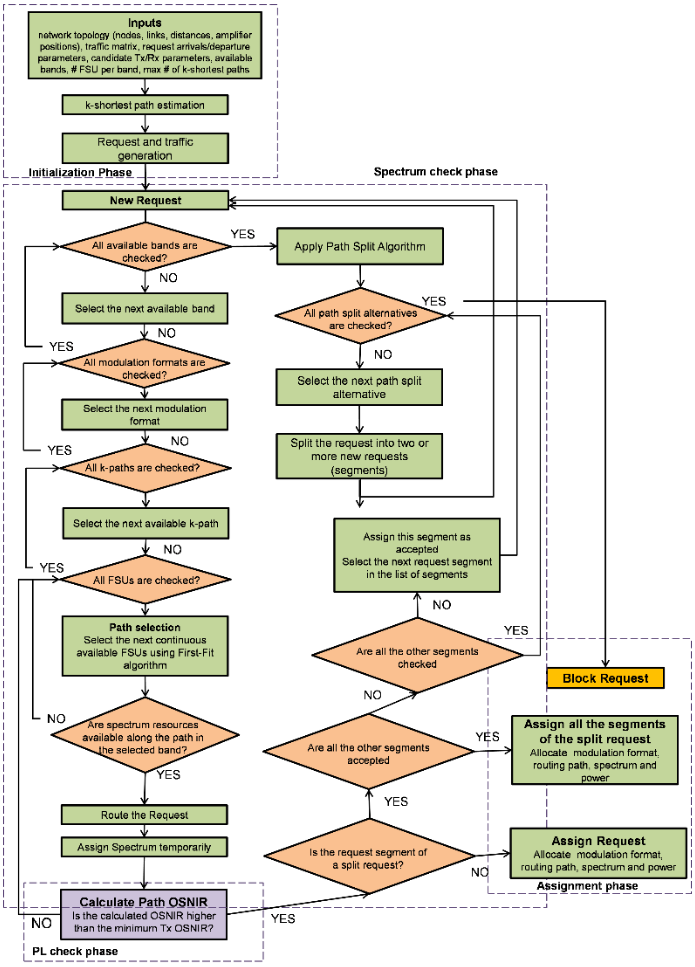

Inputs
network topology (nodes, links, distances, amplifier positions), traffic matrix, request arrivals/departure parameters, candidate Tx/Rx parameters, available bands, # FSU per band, max # of k-shortest paths
k-shortest path estimation
Request and traffic generation
Initialization Phase
Spectrum check phase
New Request
All available bands are checked?
YES
Apply Path Split Algorithm
NO
Select the next available band
NO
YES
All modulation formats are checked?
Select the next modulation format
NO
YES
All k-paths are checked?
Select the next available k-path
NO
YES
All FSUs are checked?
Path selection
Select the next continuous available FSUs using First-Fit algorithm
NO
Are spectrum resources available along the path in the selected band?
YES
Route the Request
Assign Spectrum temporarily
Calculate Path OSNIR
Is the calculated OSNIR higher than the minimum Tx OSNIR?
NO
YES
PL check phase
All path split alternatives are checked?
YES
NO
Select the next path split alternative
Split the request into two or more new requests (segments)
Assign this segment as accepted
Select the next request segment in the list of segments
NO
YES
Are all the other segments checked
NO
Block Request
Are all the other segments accepted
YES
Assign all the segments of the split request
Allocate modulation format, routing path, spectrum and power
YES
Is the request segment of a split request?
NO
Assign Request
Allocate modulation format, routing path, spectrum and power
Assignment phase

Figure 5-2 The flow chart of a multi-band PCE operation.

5.3 INTERFACE SPECIFICATION

This section lists the Interfaces that have been implemented in the Multi-Band Path Computation Engine. The MB-PCE uses two interfaces to communicate with the TAPI Optical Network Orchestrator

5.3.1 South Bound Interface (SBI) towards the TAPI Optical Network Orchestrator

The first interface is based on TAPI v2.1 and it is exposed by the TAPI Optical Network Orchestrator. MB-PCE uses this interface in order to retrieve the current optical network topology and status.

The operation of the specific interface was demonstrated in EuCNC’23 and ECOC’23.

5.3.2 North Bound Interface (NBI) towards the TAPI Optical Network Orchestrator

The second interface is exposed by the MB-PCE. It is again based on TAPI v2.1 and it is used by the TAPI Orchestrator. The TAPI Orchestrator request a path computation from the MB-PCE. The MB-PCE analyses the request and computes the optimum path and send this information back to the TAPI Orchestrator.

The operation of the specific interface was demonstrated in EuCNC’23 and ECOC’23.

5.4 Component Integration

The Multi-Band Path Computation Engine integrates with the following element:

5.4.1 TAPI Optical Network Orchestrator

The MB-PCE communicates with the TAPI Optical Network Orchestrator to realise two functionalities: a) Retrieve optical network topology; b) Path computation for a new service request.

Initially, MB-PCE communicates with the TAPI Optical Network Orchestrator using the TAPI v2.1 interface to retrieve the current optical network topology context and status.

Then, MB-PCE receives a new service request from the TAPI Optical Network Orchestrator using the TAPI v2.1 interface. The MB-PCE compute the optimum path and send this information back to the TAPI Optical Network Orchestrator.

5.5 Functional Validations

The following tests is a non-exhaustive list of tests that have been done to validate the component:

5.5.1 Network Discovery

Description: We launched the MB-PCE and retrieved the topology from TAPI Optical Network Orchestrator in terms of nodes and links and we displayed this information. This test was carried out: i) loading the information from a set of JSON files that have previously been retrieved and ii) performing dynamic loading of links and related data from the TAPI Optical Network Orchestrator.

This has been carried out in the context of EuCNC’23 and ECOC’23 demonstrations.

5.5.2 Path Computation

Description: The MB-PCE received a new service request from TAPI Optical Network Orchestrator and validated that: a) the path is properly estimated; b) the correct band is selected; c) the number of frequency allocations units are correctly assigned; d) the correct frequencies are assigned to the service; e) the correct message is generated and send back to TAPI Optical Network Orchestrator.

This has been carried out in the context of EuCNC'23 and ECOC'23 demonstrations.

5.6 Roadmap

At the time of writing, the key features for the Multi-Band Path Computation Engine have been introduced and validated with bilateral integrations. In the remaining of WP4, the ongoing activities are:

- New extensions as needed in view of new studies to be carried out.
- Integration activities under the scope of WP5

5.7 Component KPIs

This section details the KPIs that have been defined for the component and the results.

KPI	Definition	Methodology	Results
Path Computation Latency	This is the time it takes to MB-PCE to compute the path of a new service request.	Measured from the timestamps between request and response. Target (< 40s).	Several scenarios were executed. The MB-PCE Path Computation Latency was measured to be in the range between 1.8 – 3.2 seconds
TAPI Topology retrieval and parsing Latency	This is the time it takes to MB-PCE to retrieve and parse the optical network topology context (described using the TAPI format).	Measured from the timestamps between request and response. Target (< 20s).	Several scenarios were executed. The MB-PCE Topology retrieval and parsing Latency was measured to be in the range between 0.9 – 2.5 seconds

6 OPTICAL CONTROLLER

6.1 COMPONENT ARCHITECTURE AND PROGRESS DURING B5G-OPEN

The optical controller is based on results of the ONOS open-source project [ONOS] that, besides the control of optical devices, also provides a suitable environment for the control of packet devices (e.g., based on OpenFlow or P4Runtime protocols).

The main roles of the optical controller in the B5G-OPEN project are: (i) retrieve device descriptions from data plane and abstract them toward the upper control layers; (ii) receive the service configuration requests by the upper control layers and translate such requests in a set of configuration messages to be forwarded to each involved device.

The ONOS version available at the B5G-OPEN starting time was already providing a rich NBI based on REST APIs and, on its SBI it was already able to connect to a variety of packet and optical devices (e.g., exploiting NETCONF protocol). Also, the ONOS core already implements the basic connectivity services using the concept of intent that simplify and automate the service management (e.g., in case of network failure).

However, considerable development within the ONOS controller has been performed during B5G-OPEN at several levels: in the NBI, in the SBI and in the Core for introducing the following features:

1. Enable integration with T-API orchestrator (NBI)
2. Develop drivers toward new devices and update existing drivers against the most recent versions of standard models (SBI)
3. Introduce the support of flexible grid (NBI, core, SBI)
4. Introduce the support of multi-band (NBI, core, SBI)
5. Import/Export description of operational modes supported by transceivers/pluggables (SBI, NBI);
6. Activate intents using as end-points the ROADM's ports (Core).

In the figure below the aforementioned development tasks (from 1 to 6) are mapped within the ONOS architecture, where each number is reported in the affected blocks. Blocks reported in white are not modified during the project, block reported in green are upgraded during the project, while blocks reported in orange are specifically created during the project.

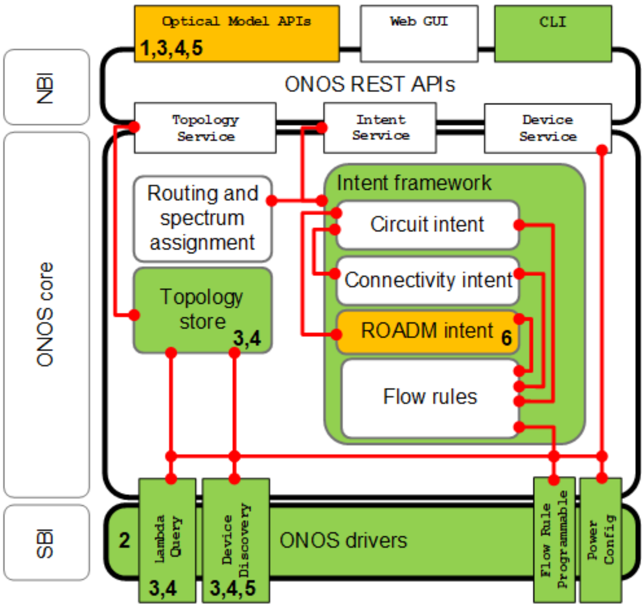


Figure 6-1 internal architecture of the optical controller, based on ONOS open-source software.

6.2 Integration with other components and Interfaces specification

The ONOS optical controller integrates with the following elements:

- The TAPI orchestrator. This entity uses the REST APIs developed on top of the ONOS controller. 1) It uses POST (and DELETE) calls to perform requests related to service provisioning (and deletion) in the optical network. 2) It uses GET calls to retrieve information regarding network topology and details regarding links and devices (e.g., description of operational modes supported by transceivers). The implemented REST interfaces (i.e., the "optical model APIs" as named in Fig. 45) is accurately described in Sec. 4.5.

- The data plane devices (i.e., packet-optical nodes, transponders, ROADM and OLS). Such devices expose to the SDN optical controller a YANG model based on OpenConfig or OpenROADM running a NETCONF server, as respectively described in Sec 4.1, and 4.6. The ONOS controller uses a NETCONF client to retrieve information and configure such devices.

6.3 Roadmap

At the time of writing, most of the described features for the SDN optical controller node have been introduced and validated with bilateral integrations. Such features have been demonstrated in an emulated environment during the review meeting and in the ECOC 2023 conference [GioDemo23]. In the remaining of WP4, the ongoing activities are:

- Enhancement of the ONOS drivers to support ROADM devices based on OpenROADM version 12.1 as described in Sec. 14.
- Deployment of an emulated multi-domain environment exploiting multiple optical controllers.

- Enhancement of the ONOS intent service to support intents requests in multi-domain scenarios, e.g., requests starting in a transceiver interface and terminating in a ROADM interface.

6.3.1 Open-source contributions

The following extensions to the ONOS software have been submitted to the open-source community:

- Driver for Cassini transceiver running proprietary NOS (merged, December 2022)
- https://gerrit.onosproject.org/c/onos/+/25168

- Extension for the support of flexi-grid (merged, February 2023)
- https://gerrit.onosproject.org/c/onos/+/25594

- Extension for the support of multi-band (merged, October 2023)
- https://gerrit.onosproject.org/c/onos/+/25596

- Extension for the support of intents between OCH ports of ROADMs (under revision)
- https://gerrit.onosproject.org/c/onos/+/24715

Other contributions are expected in the next months:

- Extended NBI providing “Optical model REST APIs” (under testing)

- Upgrade of OpenROADM driver to model version 12.1 (to be developed)

- Further extension of intent service (to be developed)

6.4 FUNCTIONAL VALIDATION

Tests have been done to validate the optical SDN controller in emulated environment.

- **Network discovery in emulated environment:** Launch the ONOS controller, including required applications and drivers. Post an emulated network topology including devices and links. Verify that all devices are correctly discovered, including interfaces and augmented details (e.g., related to physical impairments).

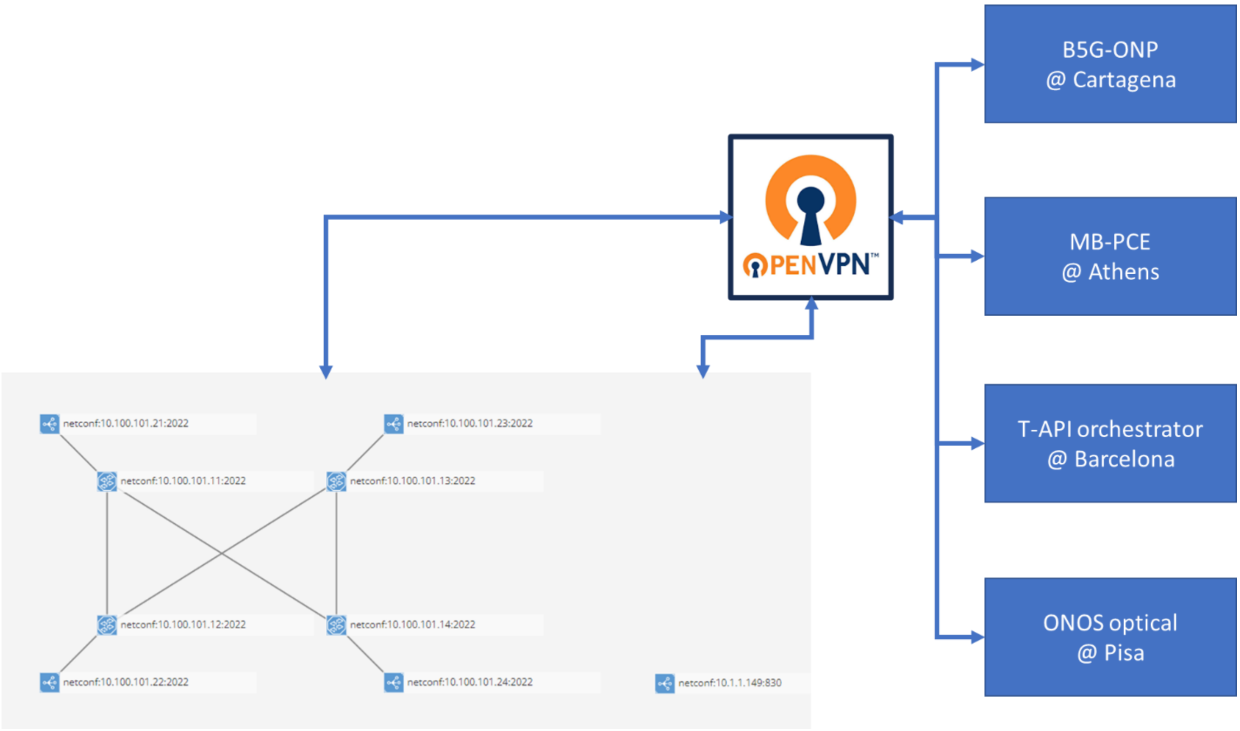

B5G-ONP
@ Cartagena
MB-PCE
@ Athens
T-API orchestrator
@ Barcelona
ONOS optical
@ Pisa
OPENVPN™
netconf:10.100.101.21:2022
netconf:10.100.101.23:2022
netconf:10.100.101.11:2022
netconf:10.100.101.13:2022
netconf:10.100.101.12:2022
netconf:10.100.101.14:2022
netconf:10.100.101.22:2022
netconf:10.100.101.24:2022
netconf:10.1.1.149:830

Figure 6-2 Network discovery validation. The ONOS web-gui shows that emulated devices are correctly discovered both in the CNIT deployment and in the CTTC deployment. Also, this ONOS controller is connected through an OpenVPN to the T-API orchestrator that is located at CTTC.

- **Network abstraction in emulated environment:** After network initialization, verify that all the acquired information regarding devices is correctly exported in the REST APIs toward upper layers.

- **Service provisioning (involve NBI, core and SBI), emulated environment:** Receive an intent request from the TAPI orchestrator (for several types of intent). Verify that the intent is correctly installed and that all involved devices are correctly configured.

- **Device tests (mainly involve SBI), real devices:** Push a specific device, test connectivity, device discovery and the ability to properly discover all the device details.

6.5 Scalability assessment and KPIs

This section describes the assessment of the open source ONOS controller on realistic network scenarios like the ones TIM is demanded to operate on field. The assessment has been performed on emulated networks whose topology and requirements, in terms of services to be carried, are taken from operating TIM metro regional DWDM infrastructure. For the purpose of these tests, three TIM metro regional networks with different features have been identified and some automation tools have been developed.

6.5.1 Network emulator

The emulator of the disaggregated optical network is based on two main components:

- The ONOS SDN controller.

- A flexible NETCONF agent based on OpenROADM models, implemented as a docker container emulating a single device (ROADM or transponder).

For running the emulator two virtual machines (VM) are deployed on the Proxmox cluster available in the TIM laboratories.

The first VM, dedicated to the ONOS controller, has the following features (in line with ONOS requirements):

- 2 processors.
- 16 GB RAM.
- 100 GB HDD.

ONOS was executed directly on the VM operating system (Ubuntu 18.04.2 LTS) and was configured to run the *gui*, *openroadm*, *optical-rest* and *drivers.odtn-driver* applications. The *ODTN-driver* implements the extensions required to control OpenROADM based ROADMs. Some small enhancements and bug fixes have been applied to it for these tests. The source code including these enhancements has been uploaded to the ONOS GitHub repository. The most significant enhancement to the *ODTN-driver* is the support of OpenROADM based transponders. Unlike the previously described contribution, the code for OpenROADM based transponders is not yet shared with the ONOS developers' community. Other small addons, specifically related to these tests, are the inclusion in the ONOS GUI of the background maps of the involved Italian regions.

The second VM runs all the containers, one for each device, needed to emulate the network. Its features are the following:

- 6 processors.
- 32 GB RAM.
- 72 GB HDD.

Docker version 18.09.2 (API version 1.39) is installed on the VM based on Ubuntu 22.04.3 LTS. However, the emulator has no dependencies on specific docker version. Two images are loaded on the local docker image repository, one for the ROADM emulator and one for the transponder emulator. Both images are based on the NETCONF server developed by TIM.

The two VMs are hosted on two separate servers interconnected by a 1 GB LAN. Given the possibly high number of devices building the emulated networks, a tool has been developed to automate the containers' start-up and the ONOS configuration. The tool takes as input a configuration file describing the emulation environment and two files describing the network (devices and links) to perform the following steps:

- Starting from templates, it creates the datastore files for each device.
- It starts all the docker containers emulating single devices (a container instance for every node).
- It creates ONOS device configuration (JSON file as required by ONOS native API).
- It sends device configuration to ONOS, waiting for all the devices to become ready.
- It creates ONOS link configuration (JSON file as required by ONOS native API).
- It sends link configuration to ONOS.

The same tool can be used to stop the emulation environment: it stops and removes all the containers and clears all the configuration from ONOS.

Another tool has been developed to automate setup of the circuits between client transponders' ports. The tool parses an input file describing the connection to be setup and asks ONOS, via REST API, to create the intent between the client ports of a source and a destination transponder. Then it waits for the status of the intent to be either 'INSTALLED' or 'FAILED'.

6.5.2 Reference networks

To assess the scalability of the ONOS controller, three DWDM transport networks of different sizes were identified. The networks have been chosen among the 14 macro regional TIM networks covering the whole country. They are structured in two tiers: a core mesh connecting the Metro core nodes, and extensions connecting aggregation nodes with the locations of the core mesh. The latter normally have a horseshoe topology but chains or weakly meshed topologies are also possible. The topology structure with its terminology is exemplified in the Figure below.

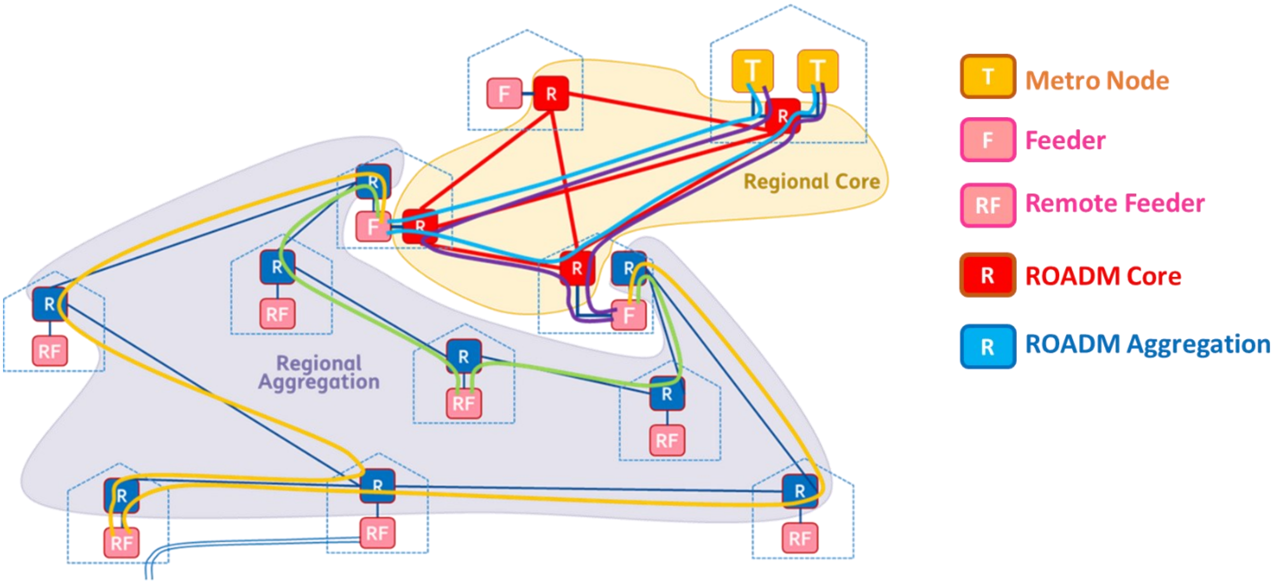


Figure 6-3: Structure of the reference Metro Regional Network

The boxes identified with "RF" (Remote Feeder), "F" (Feeder) and "T" (Metro Node) are packet switched nodes, while the boxes identified with "R" are optical nodes (in red boxes the nodes belonging to core part, in blue boxes the nodes belonging to the aggregation part, including the head ends co-located with the core nodes).

The Metro core nodes hosting two "T" boxes (one or few in each macro region) play the role of hub for packet switched traffic and are co-located with national backbone POP for long distance traffic exchange.

Each horseshoe collects a subset of aggregation nodes and connects them to a couple of core nodes which play the role of head-ends of the horseshoe. Aggregation nodes are organized in small arcs of at most 8 nodes each including two metro core nodes. Two different types of optical nodes are used for the metro core part and for the aggregation arcs: colorless directionless (CD) ROADMs with 1:9 WSS and 80 channels (currently used at 100G with coherent transmission on uncompensated systems) are used for the metro core (red boxes in Figure) while low cost colorless ROADMs with 1:9, 1:4 or 1:2 WSS (depending on degrees required on the specific nodes) and 40 channels (currently used at 10G or 100G with direct detection or coherent transmission on chromatic dispersion compensated systems) are used for the aggregation (blue boxes).

Taking this network scenario into account, three simplified networks, representing real TIM DWDM network of different sizes, have been adopted for the purpose of testing ONOS scalability.

The first is a small size poorly meshed core network of only 16 nodes of which 15 are hubbed to a single Metro hub node. This network allows preliminary tests on a network of small size before testing the controller on larger network structures.

The second network includes both the core, made of only two hub nodes, and the aggregation in a single medium size meshed network (48 nodes, of which 2 are Core nodes and 46 are Aggregation nodes). This network allows to make a test of a medium size poorly meshed network with paths composed of many hops and with a potential wavelength congestion between and near the two hubs.

The third network is the biggest core network among the TIM metro regional DWDM infrastructures with 107 nodes and 5 hub nodes and it constitutes the most challenging scenario among the Italian core networks for a network SDN controller. For example, here the ROADM with the higher (9) nodal degree employed in Italy can be found.

Concerning the list of connections, a basic set of connections to be loaded as a background carried traffic is created for the three networks. The basic set include a connection (a 100G circuit/lightpath) between each Metro Core node (not Hub) and its reference Metro Core Hub Node. This gives origin to 15 circuits for the first network (15 Metro Core nodes and 1 Metro Core Hub node), 46 circuits for second network (46 Metro Core nodes and 2 Metro Core Hub nodes) and 102 circuits for third network (102 Metro Core nodes and 5 Metro Core Hub nodes).

For the second network, two more connections are added between the two hubs, making 48 the total number of connections, while for the third network a connection between each pair of Metro Core Hubs is added (5 hubs implies 5*4/2=10 pair relationships) making 112 the total number of connections.

Even if this is not the actual traffic carried by the regional DWDM networks, the set of the defined connections reproduces a realistic pattern of the demand, which is made mainly by connections between packet equipment of Optical Packet Metro (OPM) network, and in particular 100G connections between Feeder nodes and Metro nodes (F and T nodes, respectively, in Figure).

Table 2 summarizes the main parameters of the selected networks.

Table 2: Main parameters of the selected networks

Network	Total Nodes	Metro Core Hub nodes	Aggregation nodes	Total Links	Core links	Extension links	ONOS devices	ONOS links	ONOS intents
Small	16	1	15	19	19	0	46	49	30
Medium	48	2	46	63	2	61	144	159	96
Large	107	5	102	163	163	0	331	383	224

Each network and its starting configuration of connections are described by three files:

1. a file with the list of nodes including both ROADMs and Transponders. Due to an ONOS limitation that will be described in the next subsection, one transponder node must be

created for each circuit termination: therefore, two transponders, one on each termination node (ROADM) of the circuit, must be created. Transponder nodes are directly linked to the co-located ROADM.

2. a file including the list of links between nodes describing the network topology. It includes the geographical links interconnecting ROADM in different locations (i.e. Central Offices) and the intra-CO link connecting the transponders with the co-located ROADM.
3. a file including the list of connections to be activated in the network. It includes the source and destination nodes (which must be transponder nodes).

The three files must be coherent with each other in terms of network elements (e.g. a link must be terminated on two existing and already loaded nodes) and have to be loaded in the indicated sequence from 1 to 3 (links require nodes to be terminated, and flows require nodes as terminations as well, and also links for their routing and allocation). As already described, a couple of tools have been developed to automate network emulation taking these files as input.

An extract form those files is the following:

Example of Node file

ID	Node	Latitude	Longitude	Node-type
………… file begins with the list of ROADM with their geographic coordinates …………..				
1	ABBIATEGRASSO	45.397721	8.916455	Metro-Core
2	VIGEVANO-TICINO	45.318899	8.885664	Metro-Core
3	BERGAMO	45.69819	9.674881	Metro-Core
……… etc . then the list of transponder nodes with their coordinates follows … .				
108	TS001-ABBIATEGRASSO	45.407721	8.926455	Transponder
109	TS002-VIGEVANO-TICIN	45.328899	8.895664	Transponder
110	TS003-BERGAMO	45.70819	9.684881	Transponder
……… etc up to the end of nodes ………….				

Example of Link file

ID Node A	ID Node Z	Type
………… file begins with the list of links between ROADMs (for nodes ID codes are used) …………..		
64	93	Core-link
90	93	Core-link
4	90	Core-link
90	33	Core-link
62	68	Core-link
……… etc … then a list of transponder links connecting ROADM with Transponders follows ………….		
08	1	Transponder-link
109	2	Transponder-link
110	3	Transponder-link
………… etc. up to the end …………….		

Example of connection file

Circuit-ID	TP-name-S	TP-name-D	TP-ID-S	TP-ID-D	Circuit-Type
1	TS001-ABBIATEGRASSO	TD001-MILANO-BERSAGL	108	220	100G
2	TS002-VIGEVANO-TICIN	TD002-MILANO-BERSAGL	109	221	100G
3	TS003-BERGAMO	TD003-BERGAMO-CAMPAG	110	222	100G
………… etc up to the end …………….					

6.5.3 ONOS enhancements

The ODTN driver, that provides OpenROADM support, was enhanced to find a solution in line with the requirements contained in the OpenROADM device whitepaper to the problem of assigning integer identifiers to device's ports, as required by ONOS, while the OpenROADM model assigns string identifier to them. By this proposal, integer port identifiers are taken from the *<degree>* and from the *<shared-risk-group>* branches of the model, respectively for line and add/drop ports, according to OpenROADM terminology.

Another enhancement to the ODTN driver implemented for these tests is OpenROADM transponders support. For the time being, the support is limited to 'pure' transponders and not to muxponders or switchponders, even if the OpenROADM model comprises them. In ONOS, the main issue with such kind of devices is the fact that it's not possible to express relationship between device's client and line ports. A transponder client port is constrained to a precise line port; similar condition applies to muxponders and switchponders where a group of client ports is constrained to a specific line port. It's currently not possible to instruct ONOS about such constraints and, for this reason, ONOS may incur in errors during path calculation. To overcome this limitation, the networks described in this document are modelled as being composed of terminal devices composed of a single transponder, increasing the number of emulated devices and loading ONOS device subsystem even more.

6.5.4 Test and measured KPIs

The main purpose of the tests was to assess ONOS ability of controlling a network composed of a large number of devices and links, approximating, as close as possible, the conditions that an SDN controller must afford in a real network. As it's not possible to have too many devices in a laboratory environment, the only possible solution was to use emulators. It must be said that the use of emulators doesn't produce the same results that are obtained using real devices, mainly due to the absence of the physical layer. For instance, in a real network connection, setup times are strongly influenced by physical layer parameters that need to be considered in order not to impact on already existing connections. However, even using emulators is possible to verify ONOS capability in terms of scalability of control plane features like number of simultaneous NETCONF sessions (this is because it doesn't teardown NETCONF session after device's configuration but keeps it running all time long), storing and management of topological elements (devices, links), path calculation on large graph and so on. Moreover, a real network environment would be affected also by the performances of the management network, surely less performant than a laboratory network. On the other hand, it must be highlighted that, due to the ONOS limitation on transponders' client and line ports constraints previously described that has been bypassed modelling terminal devices as being composed of a single transponder, the number of devices managed by the controller is considerably higher than what would have been without this limitation, imposing extra burden on the controller and moving the focus on its performances even further.

The first analysed aspect was the loading of the networks (devices and links) into the controller. The automation tool, after having started the docker containers emulating the network nodes, creates two JSON files, describing respectively all the devices and all the links according to ONOS native API, and POSTs them to the controller. When ONOS receives the devices configuration file, it concurrently setups the NETCONF sessions with all of them and collects basic information like number and type of ports. The automation tool waits for ONOS to advertise as many active devices as are included in the JSON file and measures the time passed since the POST operation. Table 3 shows the results obtained on the three networks: times reported are an average over 3 separate executions.

Table 3: ONOS network setup times

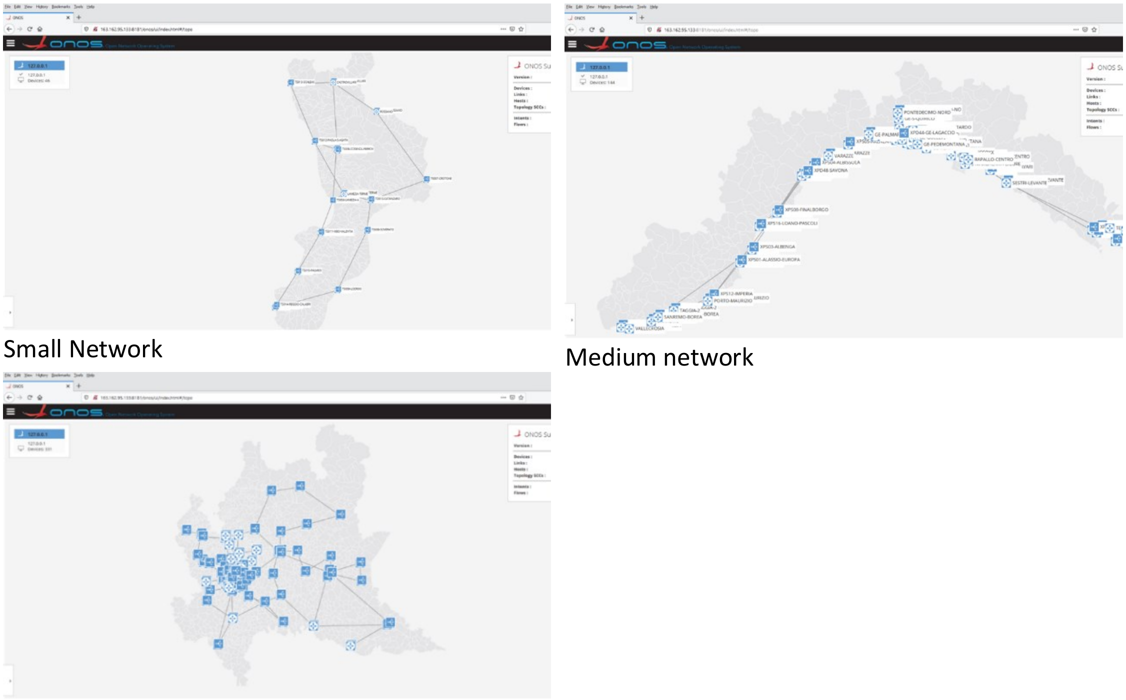
Network	ONOS devices	Network setup time	Setup time per device
Small	46	16 s	0.35 s

Medium	144	44 s	0.31 s
Large	331	120 s	0.36 s

It can be seen that ONOS seems to scale quite well since the average time to load a single device (about 350 ms) remains quite constant when the number of devices increases.

POSTing link configuration doesn't have macroscopic effects and it's not possible to observe different times as the network complexity increases, since ONOS uses the information contained in the JSON file to update its internal databases only, with no interaction with external elements.

Figure shows the ONOS GUI after the complete configuration of the three networks has been loaded. Many devices are overlapping because they have the same (or very near) coordinates, especially the devices in the hub nodes.



Small Network

Medium network

Large network

Figure 6-4: ONOS GUI representation of the three networks

The second aspect that has been analysed is the ability to set up a set of connections that represent a realistic pattern of the TIM network traffic demand. As previously described, connections are created between each leaf and the hub. Therefore, as many transponder devices as there are leaf nodes are instantiated in the hub node. A few more transponders are instantiated for inter-hub connections. In this case, the automation tool creates a JSON file for each connection, POSTs the connection request to ONOS, waits for the intent to became either "INSTALLED" or "FAILED" and continues with the next connections until the end. Table 4 shows the results obtained on the three networks: times reported are an average over 3 separate executions. It reports the average and the maximum intent "length" (in terms of number of traversed devices). The "shortest" intent is not shown, being always 4 (source and destination transponders and the two ROADMs at the source and destination locations). The number of links is reported as a reference of networks dimensions.

Table 4: ONOS connections setup times

Network	ONOS devices	ONOS intents	ONOS links	Intents setup time	Average #nodes per intent	Maximum #nodes per intent	Total time / # of intents
Small	46	30	49	45 s	5.53	7	1.5 s
Medium	144	96	159	180 s	6.12	11	1.875 s
Large	331	224	383	407 s	5.84	10	1.825 s

It should be highlighted that, for each connection request, ONOS creates two intents: the first, called *OpticalConnectivityIntent*, is created at the optical layer between the transponders' line ports: this intent configures the network media channel on the ROADMs along the path and the optical channel on the transponders and requires the identification of an available transmission wavelength. The second intent, called *OpticalCircuitIntent*, is the end-to-end circuit at the client (packet) layer and configures only the two transponders. For this reason, the number of intents in Table 4 is the double of the number of connection requests previously reported and intent setup times include the time needed by both intent types.

Again, the results show that ONOS has a good scalability also on this aspect. Since the length of the intents (in terms of number of devices) is, on average, quite the same for the three networks, one might expect a difference in setup times between the networks to be due to network dimensions having an impact on path calculation times. However, this is not the case because the average intent setup time (last column of Table 4) shows a small difference only for the first network, while between the second and the third, although the third is more than double the second (both in terms of devices and links), the difference is negligible.

An additional test has been manually performed on all the networks to check if ONOS is able to find an alternative path when the selected wavelength is not available on the shortest path. For this purpose, two pairs of additional termination transponder nodes are created with their links to ROADMs in a couple of leaf nodes. On the first pair of transponders, a connection between the client ports requiring a specific wavelength (choosing a wavelength available on all the network links) was created. On the second pair of transponders, a second connection between the nodes was created requiring the same wavelength. The expected behaviour was that ONOS had to be able to route the two connections over two different network paths, under the assumption the ROADMs to which the two pairs of transponders are attached have Colorless, Directionless and Contentionless (CDC) capabilities.

Table 5: Status of two disjoint connections (intents) instatiated over the large network.

```
{
  "intent id": "0x473",
  "app id": "org.onosproject.optical-
rest",
  "state": "INSTALLED",
  "src":
"netconf:163.162.95.81:31002/10",
  "dst":
"netconf:163.162.95.81:31003/10",
  "srcName": "XPSP2-BERGAMO-CAMPAG",
  "dstName": "XPDP2-BRESCIA-KENNED",
  "ochSignal": "+16x50.00GHz +/-
25.00GHz",
  "centralFreq": "193.9 THz",
  "pathName": "XPSP2-BERGAMO-CAMPAG ->
BERGAMO-CAMPAGNOLA -> BERGAMO ->
TRESCORE-B -> CHIARIX -> SAREZZO ->
BRESCIA -> BRESCIA-KENNEDY -> XPDP2-
BRESCIA-KENNED"
}
```

```
{
  "intent id": "0x46e",
  "app id": "org.onosproject.optical-
rest",
  "state": "INSTALLED",
  "src":
"netconf:163.162.95.81:31000/10",
  "dst":
"netconf:163.162.95.81:31001/10",
  "srcName": "XPSP3-BERGAMO-CAMPAG",
  "dstName": "XPDP3-BRESCIA-KENNED",
  "ochSignal": +16x50.00GHz +/-25.00GHz",
  "centralFreq": "193.9 THz",
  "pathName": "XPSP3-BERGAMO-CAMPAG ->
BERGAMO-CAMPAGNOLA -> BRESCIA-KENNEDY ->
XPDP3-BRESCIA-KENNED"
}
```

Table 5 shows the two intents created by ONOS over the large network (but similar results were obtained also on the other two): it’s possible to see that, as required, the same wavelength (193.9 THz) was used for both but, apart from the source and destination ROADMs (the second and the penultimate list items), two disjoint paths were employed.

7 OLS CONTROLLER

The OLS controller used in B5G-OPEN for an Adtran OLS is based on the Adtran Ensemble Network Controller software solution and is offering a northbound ONF Transport-API (TAPI) towards the Optical Controller.

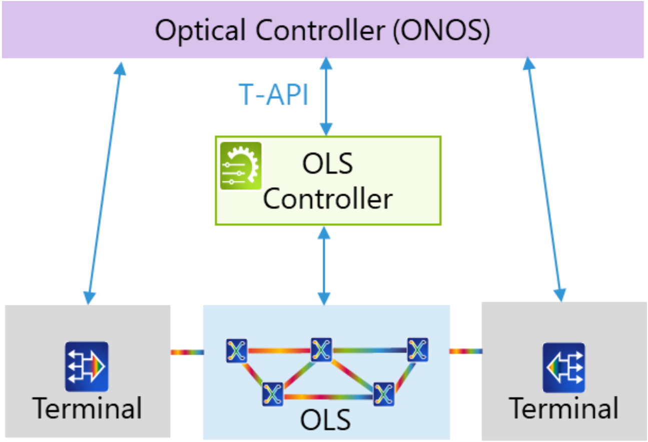


Figure 7-1 Adtran OLS Controller Northbound Interfaces

This section describes TAPI modeling of the reference topology abstraction model. The topology model provides the explicit multilayer topology that the Layer 2 to Layer 0 represents. This topology includes the OTS, OMS, OCH, ODU, and DSR layers.

Based on ONF TAPI 2.1 models, the Adtran TAPI Implementation supports a TAPI topology flat abstraction model that collapses all layers into a single multilayer topology. A single topology represents all network layers such as DSR, ODU, OCH, and Photonic Media, which include media channels, OMS, OTS and so on. This topology is modeled as a tapi- topology:topology object within the tapi-topology:topology-context/topology list. This release supports only a single topology, therefore the tapi-topology:topology-context/tapi-topology:nw- topologyservice object is not currently implemented.

SIPs represent the available service entry points. SIPs associate to all DSR, ODU and PHOTONIC_MEDIA NEPs in the network support service configuration. A SIP logically maps to one topology NEP through the tapi-topology:owned-node-edge-point/mapped-serviceinterface-point attribute. The TAPI topology data model consists of nodes and links. A node is a logical grouping of ports that provide a flexible view definition. For example, one view might represent the topology one-to-one, whereas another view can represent an entire network as a single logical node.

The current implementation delivers a single default context, with a single topology composed of:

- tapi-topology:node
- tapi-topology:link

The interface represents each physical node as a multilayer tapi-topology:node object, which creates a 1:1 logical-physical topology. The forwarding domain is the domain associated with the entire physical network element. The TAPI interface does not report any information about the internal structure of the network element. Each node displays: tapi-topology:node-edge-point. Each NEP represents an externally visible port that belongs to the node. The TAPI interface does not report any information about the internal structure of the network element. Each NEP represents:

- A client port
- An OTS port
- An OMS port
- An OCH trail termination point
- An ODUk trail termination point

The Adtran ENC TAPI implementation supports both fully aggregated topological scenario, where Adtran provides both optical line systems (OSLs) and optical transponders, as well as a partially disaggregated scenario, where Adtran provides only the OLS. In B5G-OPEN, only the partially disaggregated scenario will be used and integrated in the overall control architecture.

A full documentation of the Adran ENC TAPI is available in the ENC TAPI integration manual. In this deliverable, we focus on features used and extended in B5G-OPEN and integrated in the overall control architecture.

The ENC TAPI works over the REST framework offering the CRUD operations to perform over the network. This helps to create, read, update and delete services on the network. Further details on the operations are seen below.

Service Creation Request	
HTTP Request	POST
URL	`https://<IP>:<PORT>/restconf/data/tapi-common:context/tapi-connectivity:connectivity-context`
Description	Service Creation request between client ports of transponders A and B. Since the service is created between client ports of the transponders, service type "DSR" is used in the request and respective higher layer optical channels are created by the OLS domain controller during service provisioning. With source and destination SIPs, the TFS checks for links existing between them. If yes, the endpoint details are sent to ENC domain controller, which completes the path computation between the two client ports, and provisions the mentioned service.
Headers	`{"Content-Type": "application/yang-data+json", "Accept": "application/json"}`
Authentication	Bearer Token

Body	`{`
	`  "tapi-connectivity:connectivity-service": [`
	`    {`
	`      "end-point": [`
	`        {`
	`          "layer-protocol-name": "DSR",`

```
            "layer-protocol-qualifier": "tapi-dsr:DIGITAL_SIGNAL_TYPE_100_GigE",
            "service-interface-point": {
              "service-interface-point-uuid": "<TRANSPONDER_A_CLIENT_PORT_UUID>"
            },
            "tapi-adva:adva-connectivity-service-end-point-spec": {
              "adva-network-port-parameters": {
                "channel": {
                  "central-frequency": <CHANNEL_FREQ>
                },
                "termination-level": "OPU"
              }
            }
          },
          {
            "layer-protocol-name": "DSR",
            "layer-protocol-qualifier": "tapi-dsr:DIGITAL_SIGNAL_TYPE_100_GigE",
            "service-interface-point": {
              "service-interface-point-uuid": "<TRANSPONDER_B_CLIENT_PORT_UUID>"
            },
            "tapi-adva:adva-connectivity-service-end-point-spec": {
              "adva-network-port-parameters": {
                "channel": {
                  "central-frequency": <CHANNEL_FREQ>
                },
                "termination-level": "OPU"
              }
            }
          }
        ],
        "service-layer": "DSR",
        "service-type": "POINT_TO_POINT_CONNECTIVITY",
        "name": [
          {
            "value": <SERVICE_NAME>,
            "value-name": "USER"
          }
        ],
```

	`"include-link": ["<LINK_UUID_BTW_TRANSPONDERS>"]`
	`}`
	`]`
	`}`

Response codes	201 Created / 404 Not Found / 422 Invalid Request

Service Creation Request	
HTTP Request	GET
URL	`https://<IP>:<PORT>/restconf/data/tapi-common:context/tapi-connectivity:connectivity-context/connectivity-service`
Description	Read Service details in the network.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	**{}**
Response codes	200 OK / 404 Not Found / 422 Invalid Request

Service Deletion Request	
HTTP Request	DELETE
URL	`https://<IP>:<PORT>/restconf/data/tapi-common:context/tapi-connectivity:connectivity-context/tapi-connectivity:connectivity-service=<SERVICE_UUID>/`
Description	Delete Service in the network. This tears down all the optical layers of channels created between the source and destination.
Headers	{“Content-Type”: “application/json”, “Accept”: “application/json”}
Authentication	Bearer Token
Body	**{}**
Response codes	204 No Content / 404 Not Found / 422 Invalid Request

The deployment has the space for evaluation of KPI parameters including,

KPI	Definition	Methodology
Service Creation Time	Time consumed to create an end to end optical service between client ports of two optical terminals	Timestamp between creation request initiation and success response.
Control traffic size	Used to characterize the control traffic flowing in the network. This gives the size of requests for service creation/read/update and deletion	Capturing the packets used to take the necessary action between (mgnt. And control plane) and between (control and data plane).

Provisioning service across the domains:

In optical transport network, two types of services are created.

Client Facing Connectivity (CFC) - Optical service created between two client ports. For example, optical service with service layer "DSR" created between client ports of transponders provisioning an end to end service.

Network Facing Connectivity (NFC) - Optical service created between two network ports. For example, optical service with service layer "PHOTONIC MEDIA" created between network ports of ADD DROP multiplexers or between two OLS entities.

In a partially disaggregated network, the optical terminal like transponder and OLS domains are deployed from different vendors. Creating service across the domain is contentious as the domain controller offers only-CFC or only-NFC service creation. This means that service is created either between two network ports or two client ports, and it is not feasible to create a service between a client port of one device to network port of another device. This is still debatable as it has advantage of opening up the network but has huge complexity in combining standards and protocols of different optical layers.

A first validation of the OLS control architecture has been performed with a digital twin of a simple optical network using TeraFlowSDN as parent SDN controller. The next steps of software integration is to integrate the Adtran OLS domain controller with the Optical Network controller.

8 PON CONTROLLER

8.1 INTRODUCTION

The B5G-OPEN TDM-PON infrastructure is realised using an XGS-PON OLT pluggable transceiver (TiBit pluggable) and two ONUs. The OLT is interfaced directly to a whitebox switch while the OLT is interconnected to the ONUs by means of a splitter, forming up an ODN branch. The PON vendor (Tibit) is providing the pluggable software and the PON controller software. The integration of Tibit PON Controller with the B5G OPEN platform is realised with the development of an Access Controller as illustrated in the below figure. The Access Controller is responsible for: a) monitor the PON network and receive any requests for PON reconfiguration; b) translate these requests into high level traffic requests that will be reported to the B5G-ONP App; c) execute the appropriate actions in the PON Controller in order to support the new requests.

In addition, the Access Controller will communicate with the LiFi Controller for retrieving any connection/traffic requests.

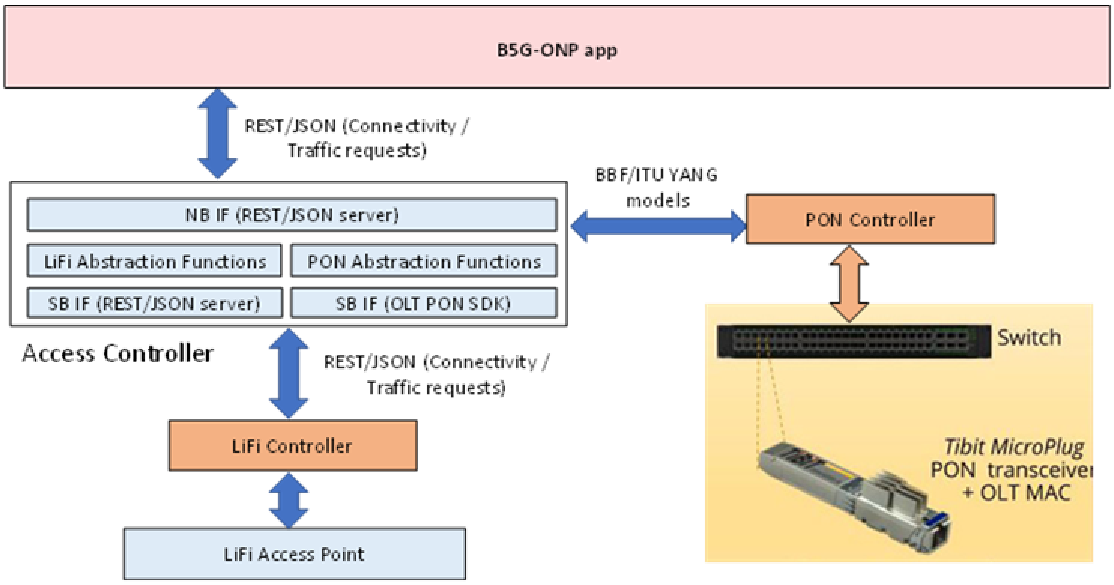

B5G-ONP app
REST/JSON (Connectivity / Traffic requests)
BBF/ITU YANG models
NB IF (REST/JSON server)
LiFi Abstraction Functions
PON Abstraction Functions
SB IF (REST/JSON server)
SB IF (OLT PON SDK)
Access Controller
PON Controller
Switch
REST/JSON (Connectivity / Traffic requests)
LiFi Controller
Tibit MicroPlug PON transceiver + OLT MAC
LiFi Access Point

Figure 8-1 B5G-OPEN PON Controller and Access Controller

8.2 INTERNAL ARCHITECTURE

The Access Controller is developed as part of the B5G-OPEN software platform, and provides the below functionalities:

- On the South Bound Interface (SBI): the Access Controller communicates with the PON Controller using REST/JSON. The SBI that communicates with the PON Controller is a software client that is developed based on provided vendor specific data modelling (YAML file).
- On the South Bound Interface (SBI): the Access Controller communicates with the LiFi Controller using REST/JSON for receiving any connectivity/traffic requests generated in the LiFi network.
- The Access Controller implements a set of: a) PON abstraction functions which are responsible to extract the PON parameters and their values; b) LiFi abstraction functions for extracting LiFi traffic parameters. In addition, the PON and LiFi abstraction functions

will expose to the higher layers only the valuable for the B5G-OPEN software platform set of parameters.

- On the Northbound Interface (NBI), the Access Controller communicates with the B5G-ONP app. The NBI implements a REST/JSON server which will support the exchange of traffic related information adopting a data structure defined in B5G-OPEN project.

8.3 Interface Specification

This section lists the Interfaces that have been implemented (as part of WP4 activities) or their implementation are ongoing as part of WP5 activities.

8.3.1 South Bound Interface (SBI) towards the PON Controller (Tibit)

The Access Controller communicates with the PON Controller (Tibit) using REST/JSON based on the vendor specific data modelling (YAML file). The integration is completed.

8.3.2 North Bound Interface (NBI) towards the LiFi Controller

The Access Controller communicates with the LiFi Controller using REST/JSON. The actual integration will be completed and reported in WP5.

8.3.3 South Bound Interface (SBI) towards the B5G-ONP app

The Access Controller communicates with the B5G-ONP app component. The actual integration will be completed and reported in WP5.

8.4 Component Integration

The Access Controller integrates with the following elements:

- The vendor specific (Tibit) PON Controller using the SBI. Communication is realized using REST/JSON.
- The LiFi Controller using the SBI. Communication will be realized using REST/JSON.
- The B5G-ONP app using the NBI. Communication will be realized using REST/JSON.

8.5 Functional Validations

The following tests is a non-exhaustive list of tests that are be used to validate the component:

8.5.1 Communication with PON Controller and Retrieving Network information

Description: The steps that we followed are listed below:

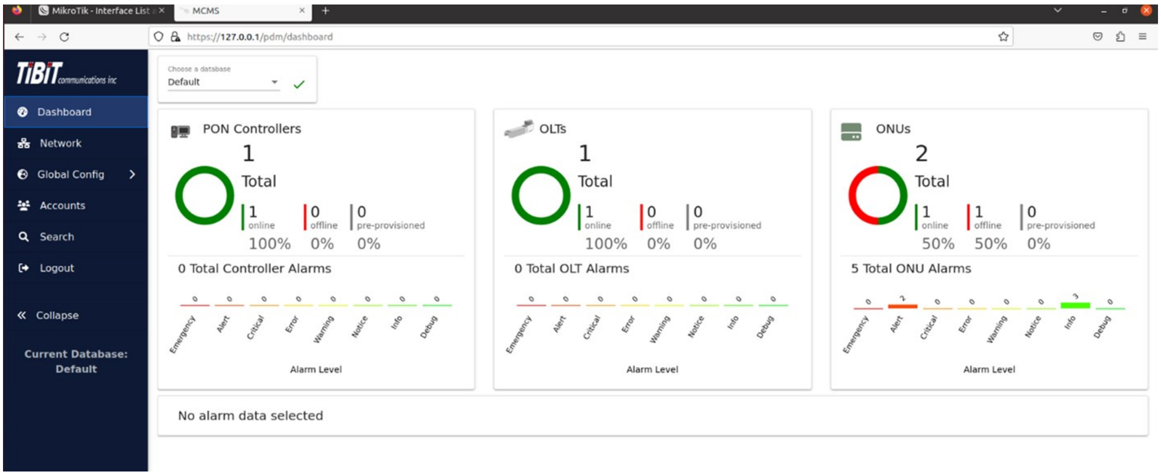
1. The PON network is deployed including the XGS-PON OLT pluggable transceiver and a couple of ONUs.
2. The PON Controller is started.
3. The Access Controller is launched, and the following items are tested:

a. The Access Controller communicates successfully with the PON Controller

b. The Access Controller retrieves the PON configuration information. In detail:
   - Controller configuration
   - OLT Configuration
   - ONUs Configuration

- SLAs Configuration
- Bandwidth Profile Configurations

Result: The test was successful. The results regarding the PON Controller are illustrated in the next figures, while the results regarding the Access Controller are reported in detail in section 2.9 (as REST result examples).



Current Database: Default

PON Controllers: 1 Total; 1 online 100%; 0 offline 0%; 0 pre-provisioned 0%

0 Total Controller Alarms

OLTs: 1 Total; 1 online 100%; 0 offline 0%; 0 pre-provisioned 0%

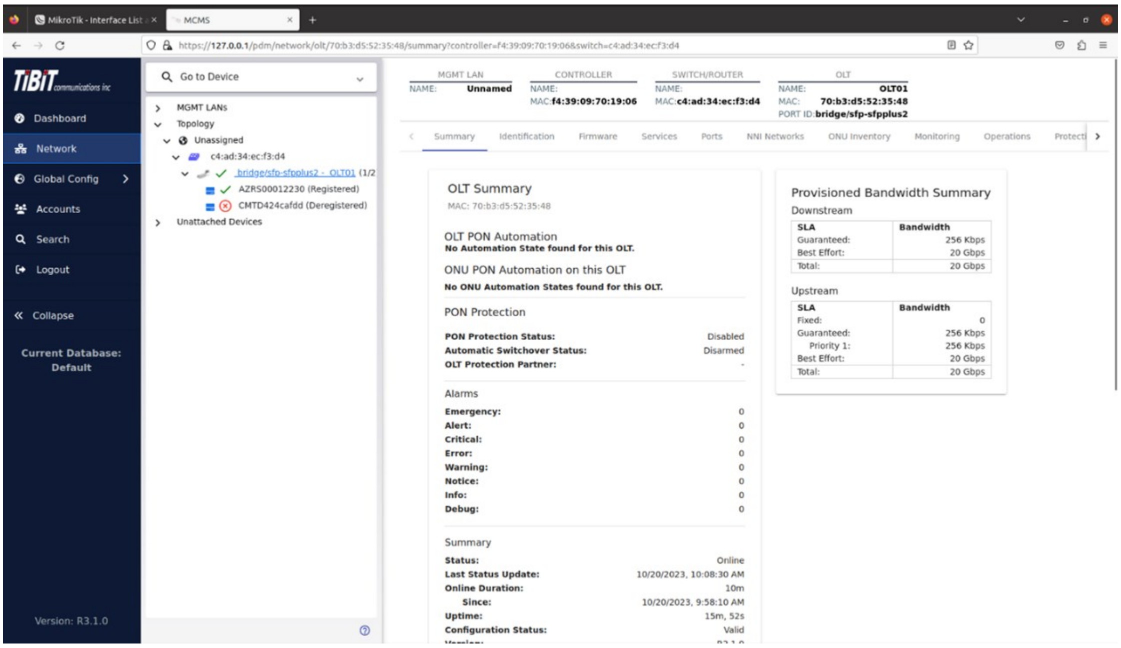
0 Total OLT Alarms

ONUs: 2 Total; 1 online 50%; 1 offline 50%; 0 pre-provisioned 0%

5 Total ONU Alarms

No alarm data selected

Figure 8-2 PON Controller Dashboard – General dashboard



Current Database: Default

Version: R3.1.0

MGMT LAN — NAME: Unnamed

CONTROLLER — NAME: MAC: f4:39:09:70:19:06

SWITCH/ROUTER — NAME: MAC: c4:ad:34:ec:f3:d4

OLT — NAME: OLT01 MAC: 70:b3:d5:52:35:48 PORT ID: bridge/sfp-sfpplus2

OLT Summary

MAC: 70:b3:d5:52:35:48

OLT PON Automation

No Automation State found for this OLT.

ONU PON Automation on this OLT

No ONU Automation States found for this OLT.

PON Protection

PON Protection Status: Disabled
Automatic Switchover Status: Disarmed
OLT Protection Partner: -

Alarms

Emergency: 0
Alert: 0
Critical: 0
Error: 0
Warning: 0
Notice: 0
Info: 0
Debug: 0

Summary

Status: Online
Last Status Update: 10/20/2023, 10:08:30 AM
Online Duration: 10m
Since: 10/20/2023, 9:58:10 AM
Uptime: 15m, 52s
Configuration Status: Valid

Provisioned Bandwidth Summary

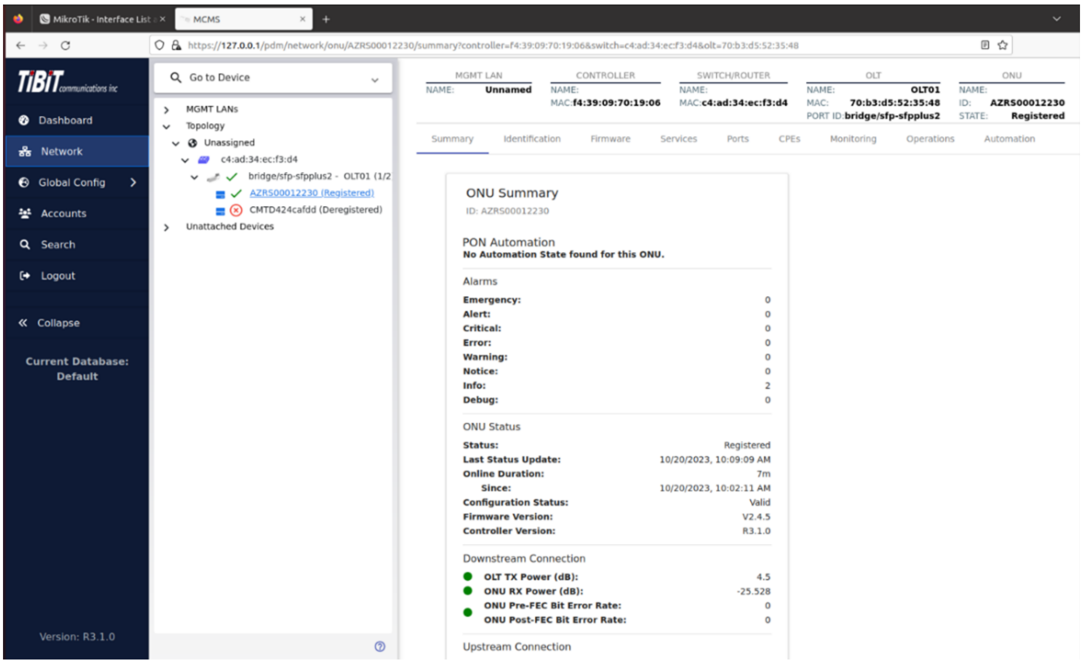
Downstream

SLA	Bandwidth
Guaranteed:	256 Kbps
Best Effort:	20 Gbps
Total:	20 Gbps

Upstream

SLA	Bandwidth
Fixed:	0
Guaranteed:	256 Kbps
Priority 1:	256 Kbps
Best Effort:	20 Gbps
Total:	20 Gbps

Figure 8-3 PON Controller Dashboard – OLT Summary


MikroTik - Interface List
MCMS
https://127.0.0.1/pdm/network/onu/AZRS00012230/summary?controller=f4:39:09:70:19:06&switch=c4:ad:34:ec:f3:d4&olt=70:b3:d5:52:35:48
TiBiT communications inc
Dashboard
Network
Global Config
Accounts
Search
Logout
Collapse
Current Database: Default
Version: R3.1.0
Go to Device
MGMT LANs
Topology
Unassigned
c4:ad:34:ec:f3:d4
bridge/sfp-sfpplus2 - OLT01 (1/2
AZRS00012230 (Registered)
CMTD424cafdd (Deregistered)
Unattached Devices
Summary
Identification
Firmware
Services
Ports
CPEs
Monitoring
Operations
Automation

MGMT LAN	CONTROLLER	SWITCH/ROUTER	OLT	ONU
NAME: **Unnamed**	NAME:	NAME:	NAME: **OLT01**	NAME:
	MAC: **f4:39:09:70:19:06**	MAC: **c4:ad:34:ec:f3:d4**	MAC: **70:b3:d5:52:35:48**	ID: **AZRS00012230**
			PORT ID: **bridge/sfp-sfpplus2**	STATE: **Registered**

ONU Summary
ID: AZRS00012230

PON Automation
No Automation State found for this ONU.

Alarms

Emergency:	0
Alert:	0
Critical:	0
Error:	0
Warning:	0
Notice:	0
Info:	2
Debug:	0

ONU Status

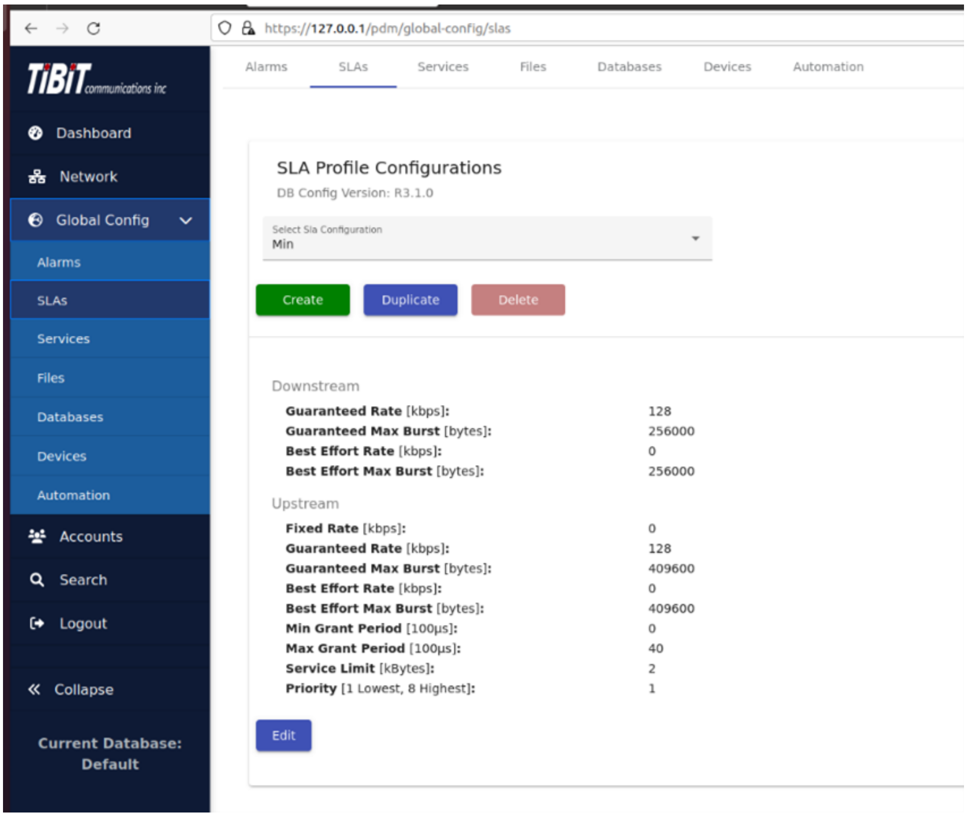
Status:	Registered
Last Status Update:	10/20/2023, 10:09:09 AM
Online Duration:	7m
Since:	10/20/2023, 10:02:11 AM
Configuration Status:	Valid
Firmware Version:	V2.4.5
Controller Version:	R3.1.0

Downstream Connection

● **OLT TX Power (dB):**	4.5
● **ONU RX Power (dB):**	-25.528
● **ONU Pre-FEC Bit Error Rate:**	0
ONU Post-FEC Bit Error Rate:	0

Upstream Connection

Figure 8-4 PON Controller Dashboard – ONU Summary



SLA Profile Configurations
DB Config Version: R3.1.0

Select Sla Configuration: Min

Downstream

Guaranteed Rate [kbps]:	128
Guaranteed Max Burst [bytes]:	256000
Best Effort Rate [kbps]:	0
Best Effort Max Burst [bytes]:	256000

Upstream

Fixed Rate [kbps]:	0
Guaranteed Rate [kbps]:	128
Guaranteed Max Burst [bytes]:	409600
Best Effort Rate [kbps]:	0
Best Effort Max Burst [bytes]:	409600
Min Grant Period [100µs]:	0
Max Grant Period [100µs]:	40
Service Limit [kBytes]:	2
Priority [1 Lowest, 8 Highest]:	1

Figure 8-5 PON Controller Dashboard – SLA/Bandwidth profile configuration

8.5.2 Communication with LiFi Controller and B5G-ONP APP

Description: The steps that we will follow are the ones below:

1. Deploy the PON network and launch both the PON Controller and LiFi Controller.
2. Launch the Access Controller.
3. LiFi Controller will generate a new traffic request and send to Access Controller.
4. Test that:
    a. the Access Controller receives successfully the request and parse all its data;
    b. translate the new request into a high-level traffic request;
    c. deliver the high-level traffic request to the B5G-ONP app using the NBI;
    d. execute the appropriate reconfiguration actions in the PON Controller;
    e. observe that the new PON configuration is realised in the testbed.

Result: Test is ongoing since the integration between Access Controller and LiFi Controller, Access Controller and B5G-ONP app are ongoing. These integration tests will be completed in WP5 and reported in WP5 deliverables.

8.6 ROADMAP

At the time of writing, the basic functionality of PON Controller and Access Controller are completed. In the remaining of WP4 and WP5, the ongoing activities are:

- Complete the integration between PON Controller and Access Controller.
- Complete the integration between LiFi Controller and Access Controller
- Complete the integration between Access Controller and B5G-ONP app

8.7 COMPONENT KPIS

This section details the KPIs that have been defined for the component and the results.

KPI	**Definition**	**Methodology**	**Results**
PON Reconfiguration Latency	This is the time it takes for the actual reconfiguration of the PON network.	Measured as time difference between the request timestamp and response timestamp. Target (< 20s).	The results are reported in the below subsection.
Access Controller Latencies	This is the time it takes to Access Controller to execute different functionalities including: a) to receive a new request and parse all its data; b) to translate the new request into a high level traffic request; c) to deliver the high level traffic request to the	Measured from the timestamps between request and response or timestamps between starting and completing a specific task.	Evaluation ongoing. Results to be reported when the integration between components will be completed (in WP5).

	B5G-ONP app; d) to execute the appropriate actions in the PON Controller.		

9 LIFI CONTROLLER

9.1 SUMMARY

The LiFi controller serves as the central component responsible for managing LiFi Aps in the network. It is strategically positioned between the PON controller and the LiFi AP. This specific positioning ensures seamless communication and enhanced coordination between the optical network layer and the wireless LiFi communication layer. Leveraging ONOS, the controller ensures a seamless integration with existing network systems, offering a scalable and modular design. Through various REST API endpoints, the LiFi controller interfaces with client applications, allowing them to configure and monitor the state of LiFi devices in the network.

9.2 DESCRIPTION AND INTERNAL ARCHITECTURE

ONOS provides a robust platform that supports basic functionalities required for a LiFi controller. Key features that make ONOS a suitable candidate include:

- Extensive Northbound and Southbound Interface Support: ONOS has in-built capabilities to communicate with a wide range of networking devices and applications.
- Support for NETCONF: ONOS provides inherent support for NETCONF, an essential requirement for our LiFi controller design.

As for the core part of ONOS, there’s no need to modify it for our design. ONOS’s core provides the foundational functionalities, and the customisations (like the LiFi device driver) simply build on top of these. To this end, the ONOS was chosen for the LiFi controller design. Besides the Northbound and Southbound Interfaces, the LiFi device driver is a critical component that facilitates communication between the LiFI controller and the LiFi Aps. The device driver contains ONOS supported NETCONF commands, which simplifies the design of translating REST API calls into NETCONF commands.

9.3 INTERFACE SPECIFICATIONS

The innovative LiFi controller’s NBI was discussed in detail in section 4.10.1. It’s based on a RESTful API design and offers endpoints for device management, IP configuration, wireless configuration, and network configurations.

For the SBI, NETCONF is the protocol of choice. ONOS has built-in support for NETCONF, which facilitates the controller-device communication. Some of the supported NETCONF commands in ONOS are:

- **get**: Retrieve information/configuration from the device.
- **editConfig**: Modify a device’s configuration.
- **copyConfig**: Copy a particular configuration to a target source.
- **deleteConfig**: Remove a specific configuration.

By leveraging ONOS’s NETCONF command set, the LiFi controller can effectively manage and configure LiFi devices in the network. Here’s part of the driver code for configuring LiFi AP after receiving calls from user:

```
    private NetconfSession getSession(DeviceId deviceId) throws NetconfException {
        NetconfDevice device = controller.getDevicesMap().get(deviceId);
        if (device == null) {
            throw new NetconfException("Device not found for DeviceId: " + deviceId);
        }
        return device.getSession();
    }
     public String getDeviceName(DeviceId deviceId) throws NetconfException {
        NetconfSession session = getSession(deviceId);
        return session.get("/plf-lifi:lifi/interface/name");
}
    public void setDeviceName(DeviceId deviceId, String name) throws NetconfException {
        NetconfSession session = getSession(deviceId);
        session.editConfig("<name>" + name + "</name>");
}
    public String getDeviceStatus(DeviceId deviceId) throws NetconfException {
        NetconfSession session = getSession(deviceId);
        return session.get("/plf-lifi:lifi/interface/status");
 }
    public String getIpAddress(DeviceId deviceId) throws NetconfException {
        NetconfSession session = getSession(deviceId);
        return session.get("/plf-lifi:lifi/interface/ip-addr");
    }
    public void setIpAddress(DeviceId deviceId, String ip) throws NetconfException {
        NetconfSession session = getSession(deviceId);
        session.editConfig("<ip-addr>" + ip + "</ip-addr>");
    }
    public String getSsid(DeviceId deviceId) throws NetconfException {
        NetconfSession session = getSession(deviceId);
        return session.get("/plf-lifi:lifi/interface/access-point/ssid");
    }
    public void setSsid(DeviceId deviceId, String ssid) throws NetconfException {
        NetconfSession session = getSession(deviceId);
        session.editConfig("<ssid>" + ssid + "</ssid>");
    }
    public void setPassword(DeviceId deviceId, String password) throws NetconfException {
        NetconfSession session = getSession(deviceId);
        session.editConfig("<password>" + password + "</password>");
    }
```

9.4 FUNCTIONAL TEST AND VALIDATION

Testing is crucial to ensure the robustness of the LiFi controller. Various functional tests must be conducted:

- REST API Endpoints Test: The purpose of the REST API Endpoints Test is to ensure that the system behaves as expected when it receives both valid and invalid inputs. This involves verifying the return values and ensuring that the system can handle erroneous scenarios gracefully.
- NETCONF Communication: The NETCONF Communication Test aims to ensure that REST API calls are correctly translated to NETCONF commands and that the system's internal translation mechanism functions as expected.

9.5 COMPONENT INTEGRATION AND ROADMAP

The LiFi prototype built in WP3 is under procurement and will be ready in Q1 2024. Then the remaining tests are:

- Full tests on LiFi controller and LiFi AP using the multiple sets of LiFi prototypes.

- Complete the integration between PON Controller.

9.6 COMPONENT KPIS

The following KPI provide insights into the efficiency and effectiveness of the LiFi controller:

KPI	Definition	Methodology	Results
LiFi Controller Latencies	This is the time it takes for the LiFi Controller to execute different requests including: 1) retrieving information by GET request; 2) update or assign settings by PUT request.	Measured from the timestamps between request and response or timestamps between starting and completing a specific task.	Evaluation ongoing. Results to be reported when the integration with all LiFi prototypes completed.

10 LIFI AGENT

10.1 SUMMARY

The LiFi Agent acts as a central hub in the architecture of the LiFi AP. With the advancement of NETCONF capabilities, the agent provides a seamless way for the AP to interact with other components, offering a structured interface for configurations and management. The agent is designed around the NETCONF server's principles, leveraging the embedded Linux environment and utilizing the 'sysrepo', 'netopeer2-server', and associated tools for its functioning.

10.2 DESCRIPTION AND INTERNAL ARCHITECTURE

The LiFi Agent is developed as a NETCONF server on the Linux based LiFi AP. The integration revolves around using 'sysrepo' as the YANG data store and the 'netopeer2-server' as the NETCONF server component. These two main components ensure that the LiFi AP can be managed using NETCONF commands, making it interoperable with standard network management tools. Key internal components and their functions are described below:

- **sysrepo**: Acts as the YANG-based data store. It keeps track of configuration and state data in a structured manner according to the 'plf-lifi.yang' model.
- **netopeer2-server**: Provides the NETCONF server functionalities. It listens for incoming NETCONF commands, processes them, and either retrieves data from or makes changes to sysrepo accordingly.
- **sysrepo-plugin**: This is a custom-built component that listens to changes made to sysrepo. Whenever a NETCONF command leads to a configuration change, the plugin acts on the LiFi AP to enact that change.
- **Telemetry Data Source and Prometheus Exporter**: This component is responsible for gathering real-time data metrics from the LiFi AP and making them available to be exported to external Prometheus server. This functionality, including data types and their specifications, is presented in section 4.14.2.

10.3 INTERFACE SPECIFICATION

Given the NETCONF nature of the LiFi Agent, the primary interface specification is based on the 'plf-lifi.yang' model. This YANG model defines structured data entities and the relationships between them, thereby providing a clear interface for the management of the LiFi AP. Details of this interface is presented in section 4.10.2.

10.4 FUNCTIONAL TEST AND VALIDATION

To ensure the LiFi Agent operates seamlessly and delivers consistent performance, we've devised a comprehensive test and validation strategy. This strategy focuses on the NETCONF operations and the accuracy and responsiveness of the telemetry exporter.

1) NETCONF Operation testing:

- Read Operation: Retrieve specific configuration or state data from the AP and validate its accuracy.
- Update Operation: Modify existing configurations on the AP, then validate if changes have been properly applied.

2) Telemetry Exporter Testing:
   Monitor the telemetry data being exported from the AP during normal operations, capturing metrics related signal strength, link speed etc. Introduce blockage to the LiFi link and monitor telemetry data changes. Upon removal of the blockage, observe the telemetry data to ensure it returns to its baseline state.

10.5 Component integration and roadmap

The LiFi prototype built in WP3 is under procurement and will be ready in Q1 2024. Then the remaining tests are:

- Full tests on LiFi controller and LiFi AP using the multiple sets of LiFi prototypes.

- Complete the integration between PON Controller and Access Controller Roadmap:

10.6 Component KPIs

The following KPIs provide insights into the efficiency and effectiveness of the LiFi Agent:

KPI	Definition	Methodology	Results
LiFi Agent Provisioning Time	This is the time it takes for the LiFi Agent to be configured with default configuration.	While LiFi AP is powered up, measured from the timestamps between start and end of the LiFi AP get default configuration. More detail is given below.	See below.
Netconf request latency	This is the time it takes to complete different Netconf request sent by the LiFi controller.	Refer to 'LiFi controller latency' in section 11.6.	Evaluation ongoing. Results to be reported when the integration with all LiFi prototypes has completed.

LiFi Agent Provisioning Time

Initial test has been done, and the full test will be repeated when all LiFi prototypes are ready in Q1 2024. In the test, the LiFi AP is firstly powered up and running normally. The following payload is then sent to the LiFi agent as default configuration. Timestamps are recorded between the start and end of the configuration. This is then repeated for 50 times and an

```
vlan = random.randint(1, 4096)
    payload = """
    <config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <lifi xmlns="http://.purelifi.com/yang">
    <interface>
    <name>wlan0</name>
    <status>up</status>
    <access-point>
    <enabled>true</enabled>
    <mode>ap</mode>
    <ssid>LIFI""" +str(vlan)+"""</ssid>
    <security>
    <password>TESTTEST</password>
    <encryption>WPA2</encryption>
    </security>
    <vlan-id>"""+str(vlan)+"""</vlan-id>
    </access-point>
    </interface>
    </lifi>
    </config>
    """
```

The preliminary result is as follows (in ms) and the CDF of the provisioning time is shown in Figure 10.1.

```
50 Execution Times:

---------------------

[764.119, 703.4970000000001, 873.011, 902.717, 866.484, 878.086, 893.827, 892.3259999999999,
894.537, 909.798, 893.312, 867.713, 689.763, 878.305, 763.288, 677.218, 787.376,
899.4799999999999, 904.841, 882.895, 888.243, 888.211, 887.379, 876.3109999999999, 885.113,
891.397, 655.474, 860.472, 889.007, 867.418, 863.126, 739.1809999999999, 822.062, 899.787,
935.163, 912.1949999999999, 864.754, 921.479, 882.345, 889.575, 877.592, 861.662,
870.1750000000001, 878.7439999999999, 919.9359999999999, 933.496, 667.4060000000001,
750.038, 724.9060000000001, 955.3770000000001]

Average Time:

-----------------

851.6123400000002
```

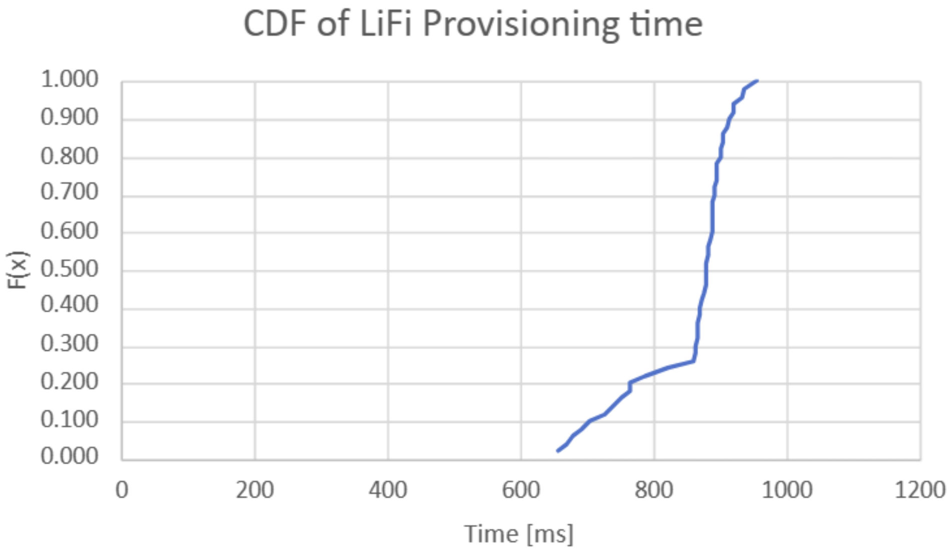


Figure 10.1: Preliminary result of LiFi Agent Provisioning Time

11 OpenROADM Agent

The OpenROADM agent is an implementation of a NETCONF server controlling optical network elements using OpenROADM device models. It's basically an evolution of the agent developed for the H2020 Metro-Haul project that was limited to ROADM devices. In B5G-Open the agent has been enhanced to cover MultiBand technology exploiting the latest OpenROADM models.

11.1 Description and internal architecture

The OpenROADM agent exploits the transAPI framework available for Netopeer, an open source implementation of the NETCONF protocol. The transAPI allows invoking call-back functions whenever an `edit-config rpc` operation performs changes on a specific branch of the configuration. Starting from this feature, the agent implements call-back functions that manage the controller requests for the creation of the interfaces that, according to the OpenROADM device model, are required for connection and optical channel setup.

To decouple the OpenROADM model processing from the action required by the underlying hardware, the agent architecture leverages on the Linux dynamic libraries subsystem to load specific drivers at runtime. The drivers are associated to circuit-packs, an OpenROADM entity used to model atomic elements inside a device that, according to the model, must have a type attribute. For the agent, every `circuit-pack-type` can have its specific driver that is loaded by the main module when an `edit-config rpc` creates the first circuit-pack of that type. The following figure shows the agent architecture applied to a ROADM architecture.

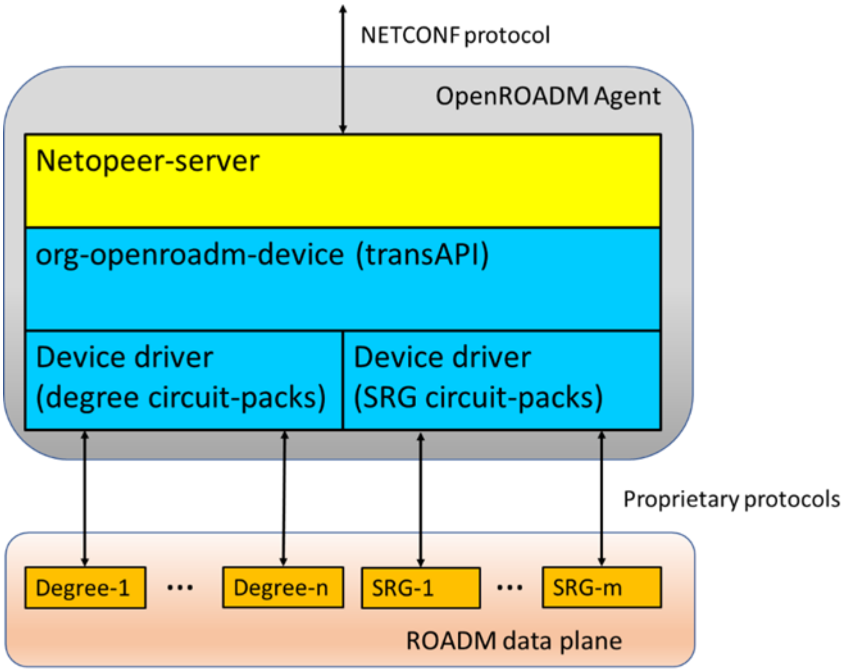


Figure 11.1: OpenROADM agent architecture

11.2 Interface Specification

The OpenROADM agent has two different interfaces: a NorthBound interface towards an SDN controller and some SouthBound interfaces towards the data plane devices.

The NBI is a NETCONF/YANG interface implementing the OpenROADM device models. For the B5G-OPEN project, the agent will upgrade from the 2.2 to the 12.1 device model in order to incorporate all the latest enhancements dedicated to the pluggables and to the multiband technology.

The YANG models that are involved (directly or because imported by other models) are:

```
org-openroadm-device.yang
org-openroadm-network-media-channel-interfaces.yang
org-openroadm-media-channel-interfaces.yang
org-openroadm-prot-otn-linear-aps.yang
org-openroadm-port-capability.yang
org-openroadm-rstp.yang
org-openroadm-pluggable-optics-holder-capability.yang
org-openroadm-otn-odu-interfaces.yang
org-openroadm-otn-otu-interfaces.yang
org-openroadm-optical-transport-interfaces.yang
org-openroadm-optical-channel-interfaces.yang
org-openroadm-lldp.yang
org-openroadm-ethernet-interfaces.yang
```

The SBI is a proprietary interface based on function calls. A driver module implements functions to perform actions on the circuit-packs composing the device. For example, a ROADM degree can be composed of WSSes and amplifiers modelled as programmable circuit-packs and a dedicated driver can be written for configuring them. The functions that a driver can implement are the:

- **init**: called during agent start-up to set up the internal communication session between the agent and the circuit-pack and to perform initial circuit-pack setup.
- **close**: called at agent closing to free all the allocated resources.
- **get_inventory**: called when the agent needs circuit-pack inventory information.
- **get_port_operational_state**: called to get the operational state of a port.
- **get_port_media_channel_capability**: called by the agent to retrieve the optical capabilities of a device’s port, in terms of supported minimum and maximum wavelength and channel spacing.
- **make_connection**: cross-connection (spectral window) creation between circuit-pack ports.
- **delete_connection**: cross-connection removal.

To cope with circuit packs that are not programmable (such as mux-demux), all functions are optional, but the followings constraints apply:

- If **init** function is defined the close function must be present.
- If **make_connection** is defined the **init** function must be present.
- If **make_connection** is defined the **delete_connection** function must be present.

Other functions could be added in future releases of the agent.

It’s worth specifying that the agent software architecture is flexible enough to manage devices composed of circuit-packs from different vendors and allows easy implementations of emulators, since it is possible to create “dummy” drivers that perform no actions. As already did in the Metro-Haul project, this feature could be exploited to control hardware prototypes from other partners.

Discussions are ongoing with TU/e to adopt the agent for the device under development.

11.3 Functional Tests and Validations

The tests that can be done to validate the OpenROADM agent are the following:

- Startup of the netopeer-server daemon and loading of the transAPI module implementing the agent, of the OpenROADM device models and of the initial configuration data. After loading the configuration data, the agent must load all the configured device drivers (Loadable Linux libraries) and identify the implemented API functions.
- Using a NETCONF client (e.g. netopeer-cli), retrieval of configuration and state data, focusing on the optical parameters, in terms of supported spectral windows (media channel capabilities).
- Edit-config operation to create cross-connections on the different bands of a multi-band ROADM

11.4 Component Integration and Roadmap

The OpenROADM agent will be integrated with the ONOS SDN Controller for the control of multiband ROADMs and of ZR/ZR+ pluggables.

Roadmap:

Q4/2023 – Integration and validation of Multiband extensions with the ONOS SDN controller.

11.5 Component KPIs

The kind of KPI that can be applied to the agent are related to delays and latencies. The agent architecture allows it to be employed with different data-plane hardware to implement basic OpenROADM based ROADMs and transponders. This makes it difficult to define a set of measurements that can characterize all kind of devices that can be implemented. Moreover, most of the time required to perform different actions is spent by the specific data plane hardware and it results to be several order of magnitude slower than the time taken by the agent for NETCONF messages processing (tenth of seconds wrt tenth of milliseconds). Therefore, it’s clear that it would be more meaningful to characterize a device employing the agent.

That said and keeping in mind that delays involving a software module are negligible wrt delays of optical resources, it can be meaningful to characterize the agent as a standalone component (i.e. without underlying data-plane hardware) from a scalability point of view, i.e. analysing how internal resources dimensions impact on the operations of the component. The following KPIs are defined:

- Changes of time needed to load the start-up datastore (at agent initialization) wrt the number of modelled degrees, i.e. wrt to datastore dimensions. For example, how start-up times changes increasing the degrees number (e.g. 4, 6, 9 to cover the most typical ROADMs sizes).

- Time required by the ONOS SDN controller to discover the device (port capabilities) wrt the number of degrees.
- Time required to create roadm-connections as a function of the number of degrees (4, 6, 9), i.e. of the datastore dimensions. According to the OpenROADM MSA, setup of new roadm-connections requires the creation of a relationship between two Network Media Channel (NMC) interfaces that, in turn, reference related Media Channel (MC) interfaces. Therefore, the worst-case scenario is when a large number of MC and NMC interfaces (e.g. full set of 100 GHz frequency slots in the C-Band, 191.3 to 196.1 GHz) has been created on all available degrees.

KPI	**Definition**	**Methodology**	**Results**
Start-up delay	Characterize aspects related to instantiation of the agent.	Accessing the agent log, measure the time required to setup the device. Repeat for different datastore sizes (4 and 9 degrees).	See below
Discovery Latency	Time required by ONOS SDN controller to discover the ROADM and its port capabilities.	Accessing the agent log, measure the time required by ONOS to discover device ports. Repeat for different datastore sizes (4 and 9 degrees).	See below
Connection Latency	Time required to create a cross-connection between two roadm degrees. This require a configuration change that is reflected in the datastore.	Accessing the agent log, measure the time for setting a roadm-connection	See below

11.5.1 Agent start-up delay

Datastore dimensions		
4-degrees	6-degrees	9-degrees
437 ms	460 ms	477 ms

Average times over 3 executions.

As an example, here the agent log for a 9 degree datastore (only significative messages):

```
2023-06-07T10:19:14.089473773Z netopeer-server[1]: Shared memory location: /dev/shm/libnetconfshm
2023-06-07T10:19:14.089575033Z netopeer-server[1]: POSIX SHM File Descriptor: 3 (600B).
2023-06-07T10:19:14.091921831Z netopeer-server[1]: ncds_features_parse: no feature definitions found
in data model ietf-inet-types.
(...)
2023-06-07T10:19:14.434878328Z netopeer-server[1]: Datastore org-openroadm-device initiated with ID
1714636916.
2023-06-07T10:19:14.484574047Z netopeer-server[1]: *************************************
2023-06-07T10:19:14.484612868Z netopeer-server[1]: *** Initializing OpenROADM device ***
2023-06-07T10:19:14.484628413Z netopeer-server[1]: *************************************
2023-06-07T10:19:14.504511993Z  netopeer-server[1]:  Transapi  calling  callback  /A:org-openroadm-
device/A:info with op MOD.
2023-06-07T10:19:14.504558090Z netopeer-server[1]: *** ROADM Device ***
2023-06-07T10:19:14.504829072Z netopeer-server[1]: Initializing circuit-pack DEG1-AMP (dummy-amp)
2023-06-07T10:19:14.505215933Z netopeer-server[1]: Initializing circuit-pack DEG1-WSS (dummy-wss)
2023-06-07T10:19:14.505253587Z netopeer-server[1]: Initializing circuit-pack DEG2-AMP (dummy-amp)
2023-06-07T10:19:14.505411899Z netopeer-server[1]: Initializing circuit-pack DEG2-WSS (dummy-wss)
2023-06-07T10:19:14.505462868Z netopeer-server[1]: Initializing circuit-pack DEG3-AMP (dummy-amp)
```

```
2023-06-07T10:19:14.505494554Z netopeer-server[1]: Initializing circuit-pack DEG3-WSS (dummy-wss)
2023-06-07T10:19:14.505524873Z netopeer-server[1]: Initializing circuit-pack DEG4-AMP (dummy-amp)
2023-06-07T10:19:14.505772436Z netopeer-server[1]: Initializing circuit-pack DEG4-WSS (dummy-wss)
2023-06-07T10:19:14.505825798Z netopeer-server[1]: Initializing circuit-pack DEG5-AMP (dummy-amp)
2023-06-07T10:19:14.505854604Z netopeer-server[1]: Initializing circuit-pack DEG5-WSS (dummy-wss)
2023-06-07T10:19:14.505927889Z netopeer-server[1]: Initializing circuit-pack DEG6-AMP (dummy-amp)
2023-06-07T10:19:14.506025481Z netopeer-server[1]: Initializing circuit-pack DEG6-WSS (dummy-wss)
2023-06-07T10:19:14.506109435Z netopeer-server[1]: Initializing circuit-pack DEG7-AMP (dummy-amp)
2023-06-07T10:19:14.506231042Z netopeer-server[1]: Initializing circuit-pack DEG7-WSS (dummy-wss)
2023-06-07T10:19:14.506272650Z netopeer-server[1]: Initializing circuit-pack DEG8-AMP (dummy-amp)
2023-06-07T10:19:14.506303062Z netopeer-server[1]: Initializing circuit-pack DEG8-WSS (dummy-wss)
2023-06-07T10:19:14.506466161Z netopeer-server[1]: Initializing circuit-pack DEG9-AMP (dummy-amp)
2023-06-07T10:19:14.506493900Z netopeer-server[1]: Initializing circuit-pack DEG9-WSS (dummy-wss)
2023-06-07T10:19:14.506524438Z netopeer-server[1]: Initializing circuit-pack SRG1-WSS (dummy-wss)
2023-06-07T10:19:14.506719760Z netopeer-server[1]: Initializing circuit-pack SRG1-AMP (dummy-amp)
2023-06-07T10:19:14.506747786Z netopeer-server[1]: Initializing circuit-pack SRG1-CS (dummy-wss)
(...)
2023-06-07T10:19:14.508365215Z netopeer-server[1]: Starting FMON thread for org-openroadm-device data
model.
2023-06-07T10:19:14.542438216Z netopeer-server[1]: Netopeer server successfully initialized.
```

11.5.2 ONOS Discovery Latency

Datastore dimensions		
4-degrees	6-degrees	9-degrees
30.212 ms	39.842 ms	43.743 ms

It's worth explaining that the difference between the times of this test wrt. those described is sect. 6.5.4 can be explained by the fact that ONOS uses separate threads for every device so to achieve a high degree of parallelism.

11.5.3 Connection latency

This KPI has been measured on two 'flavours' of the datastore:

- A "small datastore" containing only the MC and NMC interfaces needed for creating the roadm-connections
- A "large datastore" containing a full set of 100 GHz frequency slots in the C-Band, 191.3 to 196.1 GHz, on all available degrees, resulting in additional 384, 576 and 864 interface definitions for the 4-, 6- and 9-degrees cases.

	Small datastore			Large datastore		
Connection	4-degrees	6-degrees	9-degrees	4-degrees	6-degrees	9-degrees
DEG1 – DEG2	172 ms	232 ms	305 ms	822 ms	1.286 ms	2.318 ms
DEG1 – last DEG	201 ms	244 ms	297 ms	824 ms	1.353 ms	2.493 ms

From the table, it is quite clear that the nodal degree is the main factor influencing connection setup times, while the type of the connection (i.e. if between two adjacent or two distant degrees) have negligible influence. In the "Large datastore" case, this is even more apparent because of the fact that a higher number of degrees means a larger number of MC and NMC interfaces populating the datastore.

12 OpenConfig Agent

12.1 Introduction

There will be two different implementations of the OpenConfig agent. One implemented at CTTC with the main aim of integrating it with the CTTC developed multi-band transceiver and one implemented at CNIT with the main aim of integrating it into the packet-optical SONiC based node. Since the software architecture, interfaces, and proposed KPIs are the same, both implementations are described in this section.

OpenConfig agent is an implementation of an SDN agent using NETCONF/YANG with the OpenConfig data models. It implements a subset of the data models, namely the OpenConfig platform and optical transport as well as some extensions devised in the context of B5G-OPEN to report details about the transceiver operational modes.

The software relies on ConfD free, a Tail-f/Cisco management agent software framework for network elements. It enables the industry adoption of NETCONF and YANG and provides a simple mechanism to develop SDN agents focusing on the business logic and on the actual data models and semantics.

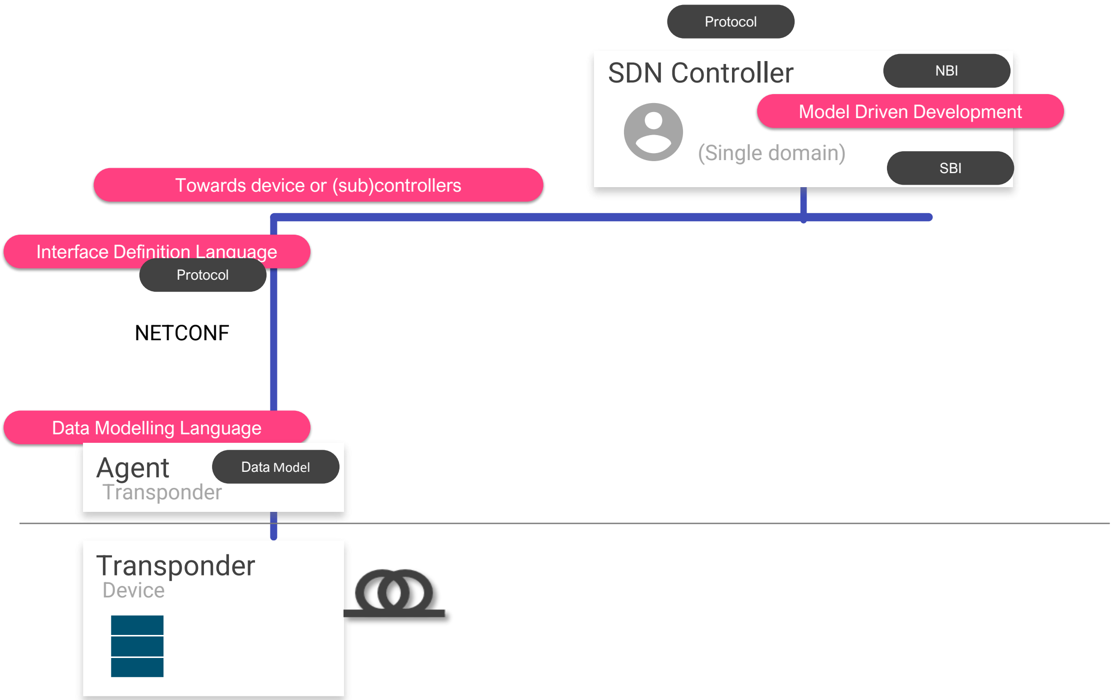


Figure 12.1: OpenConfig agent scope and macroscopic architecture

12.2 Description and Internal Architecture

OpenConfig agent relies on a ConfD process running, which implements the basic NETCONF/Yang framework. The software pre-compiles data models and keeps a Configuration Database (CDB).

Open Optical Terminals in general cover transponders, switchponders, muxponders, etc. with the ability to switch and multiplex multiple client signals into optical signals. The agent deals with uniform components hierarchy, multiplexing stages and Cross-connection logic discovery and Optical channel configuration (Frequency, power and operational mode).

The actual logic is implemented as a second process that connects to the ConfD daemon via dedicated sockets. This process is written in C++ and implements different classes for interacting with the ConfD engine. MAAPI is a C API which provides full access to the ConfD internal transaction engine. The CDB API can read (committed) configuration from the CDB and has functions like cdb_set_value for operational (state) data only. With MAAPI it is possible to create or attach to existing transaction and access configuration data in the CDB. The modifications will be then propagated at commit time of the transaction.

Notably, the agent takes care of the following aspects:

- Notification of changes in the configuration database: in this sense, the actual SDN agent may react and configure the hardware accordingly. In particular, it registers appropriate call-backs for changes in the configuration of Optical Channels, including the actual frequency, transmit optical power and operational mode.

- It relies on CDB API and MAAPI from ConfD to write on the operational data store, in such a way that operational data can be written to the ConfD database based on the status of the hardware. In particular, the agent MUST report the composition of the actual device in terms of components and subcomponents and reflect configuration changes (e.g., config/frequency) into state values (e.g., state/frequency).

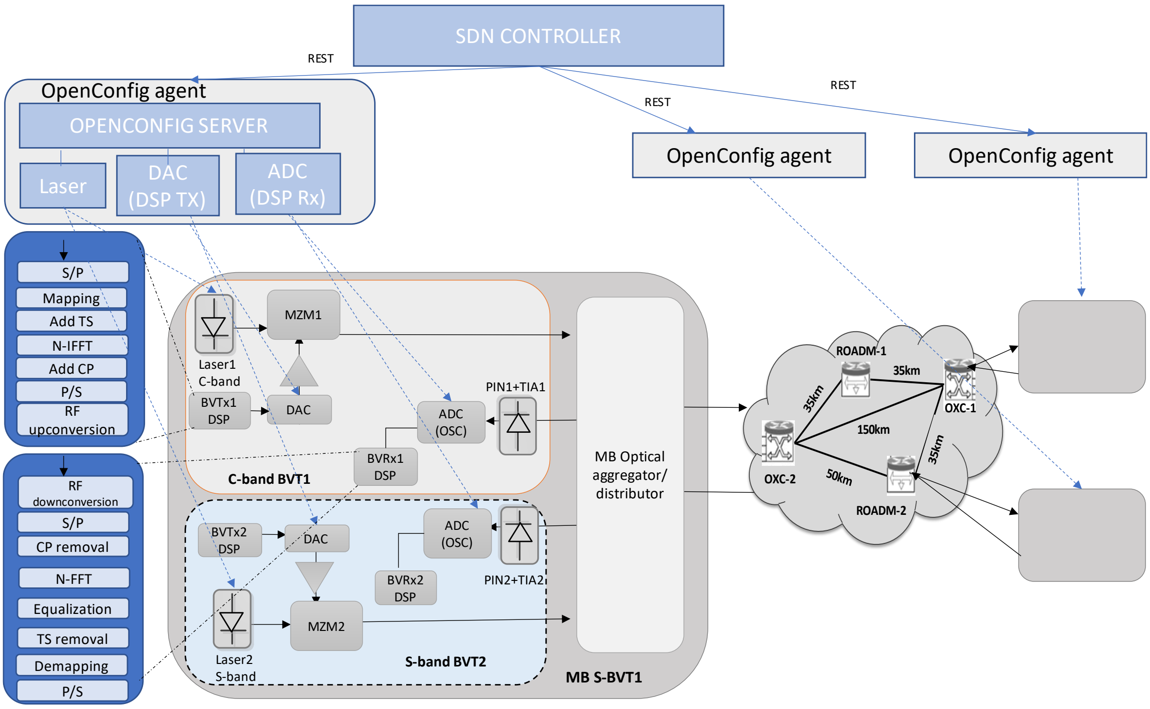


Figure 12.2: SDN Controlled SBVT

12.3 Interface Specification

The interfaces of the component are specified in Section 2.1, on the usage of OpenConfig for optical terminal devices. The agent relies on NETCONF/YANG as the basic framework, with a subset of the supported data models, mainly:

`openconfig/optical-transport/openconfig-transport-types@2017-08-16.yang`

`ietf/ietf-interfaces@2014-05-08.yang`

`openconfig/interfaces/openconfig-interfaces@2017-07-14.yang`

`openconfig/types/*.yang`

`openconfig/platform/*.yang`

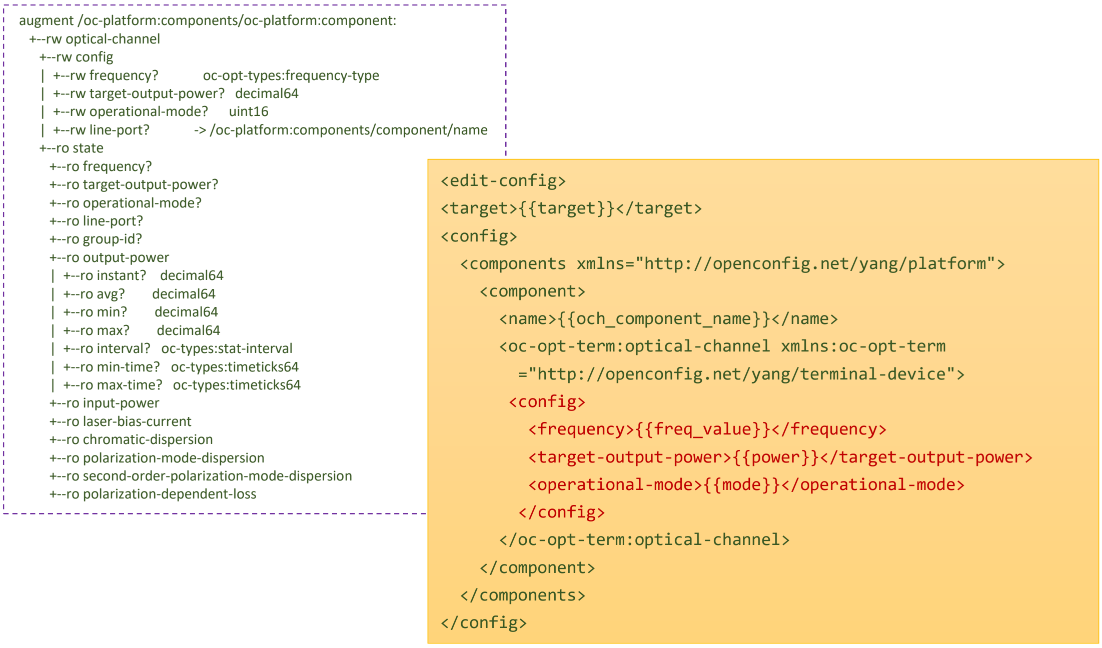
`openconfig/optical-transport/*.yang`

In particular, the agent also implements the following extensions, detailed previously in the document

`b5gopen/openconfig-terminal-device-property-types.yang`

`b5gopen/openconfig-terminal-device-properties.yang`

The set of functional tests and workflows that are in scope of B5G-OPEN are: i) device discovery (NETCONF GET) and ii) OpenConfig Component discovery (NETCONF GET). The figure below shows an example of exchange to configure the frequency, target output power and operational mode.



```
augment /oc-platform:components/oc-platform:component:
  +--rw optical-channel
    +--rw config
    |  +--rw frequency?           oc-opt-types:frequency-type
    |  +--rw target-output-power?   decimal64
    |  +--rw operational-mode?     uint16
    |  +--rw line-port?           -> /oc-platform:components/component/name
    +--ro state
      +--ro frequency?
      +--ro target-output-power?
      +--ro operational-mode?
      +--ro line-port?
      +--ro group-id?
      +--ro output-power
      |  +--ro instant?   decimal64
      |  +--ro avg?       decimal64
      |  +--ro min?       decimal64
      |  +--ro max?       decimal64
      |  +--ro interval?   oc-types:stat-interval
      |  +--ro min-time?   oc-types:timeticks64
      |  +--ro max-time?   oc-types:timeticks64
      +--ro input-power
      +--ro laser-bias-current
      +--ro chromatic-dispersion
      +--ro polarization-mode-dispersion
      +--ro second-order-polarization-mode-dispersion
      +--ro polarization-dependent-loss
```

```
<edit-config>
<target>{{target}}</target>
<config>
  <components xmlns="http://openconfig.net/yang/platform">
    <component>
      <name>{{och_component_name}}</name>
      <oc-opt-term:optical-channel xmlns:oc-opt-term
        ="http://openconfig.net/yang/terminal-device">
       <config>
         <frequency>{{freq_value}}</frequency>
         <target-output-power>{{power}}</target-output-power>
         <operational-mode>{{mode}}</operational-mode>
        </config>
      </oc-opt-term:optical-channel>
    </component>
  </components>
</config>
```

Figure 12.3: configuration of a SBVT Optical Channel using OpenConfig

12.4 Functional Tests and Validations

The tests that have been carried to validate the component are the following:

- Startup of the ConfD daemon and loading of initial operational and configuration data

- Retrieval of the datastore of the NETCONF agent.

- Retrieval of components of the OpenConfig terminal device, including focusing on the optical channel augment.

- Retrieval of the characteristics and current state of an Optical Channel components

- Dynamic configuration of an optical channel attributes. This can be done as an emulated device, or integrated with CTTC S-BVT and SONiC-based packet-optical nodes (i.e., pluggable configuration).

- Characterization of a given Operational Mode (e.g., mode-id 100) in terms of B5G-OPEN physical impairment validation.

12.5 Component Integration and Roadmap

The OpenConfig agent is integrated with the ONOS SDN Controller, for the control of sliceable BVTs.

Roadmap:

Q1/2023 – Software availability of the agent, considering a basic implementation focusing on aspects such as component discovery, optical channel configuration and operational mode characterization based on B5G-OPEN extensions.

Q3/2023 – Integration with ONOS SDN controller and functional tests. This requires the ONOS SDN controller to connect to the OpenConfig agent and discover basic blocks and operational modes.

Q3-Q4/2023 – Performance evaluation and evaluation/refinement of KPIs of the component.

12.6 Component KPIs

The considered set of component KPIs that can be measured independently:

- *Instantiation delay and footprint*.
- *Discovery latency*.
- *Operational Mode characterization*: measure the time and the control plane overhead (in terms of bytes, and throughput) it takes for an SDN controller to discover the details of a given operational mode
- *Transaction delay*: the time it takes to send a configuration change, and this is reflected in the datastore. The focus shall be to change an Optical channel frequency, power, and operational model. This KPI will be evaluated with and without hardware.

KPI	Definition	Methodology	Results
Instantiation Delay	Characterize aspects related to instantiation of the agent, as well as aspects related to memory usage.	Measure the time to launch the application and agent.	See below

Discovery Latency	SDN controller to discover the components of the transceiver upon a NETCONF get operation.	Measure the time and the control plane overhead (in terms of bytes, and throughput).	See below
Operational Mode Characterization	Obtain the parameters of a given Operational Mode given its mode-id	Measure the time and the control plane overhead (in terms of bytes, and throughput) the details of a given operational mode	See below
Transation Delay	The time it takes to send a configuration change, and this is reflected in the datastore. The focus shall be to change an Optical channel frequency, power, and operational model. This KPI will be evaluated with and without hardware.	Measure the time for a simple transaction (Note: without hardware configuration)	See below

12.6.1 Instantiation Delay

Measure the time to execute the following script

```
#!/bin/bash

sudo ./bin/confd --verbose -c etc/confd/confd.conf

# Load Configuration Data - l:load o:ignore operational data
echo "Loading config data..."
./bin/confd_load -l -o -e ./openconfig/openconfig-data.xml

echo "Loading operational data..."
# Load operational data
./bin/confd_load -C -e ./openconfig/openconfig-operational.xml

# Load basic access rules (merge)
echo "Loading access rules..."
./bin/confd_load  -m -l ./var/confd/cdb/aaa_init.xml

echo "Running agent"
/opt/confd/agent/build/ocagent
```

```
time ./b5gopen_time1.sh
Loading config data...
Loading operational data...
Loading access rules...

real    0m1,234s
user    0m0,020s
sys     0m0,017s
```

Measure the time to launch the application specific C++ agent that maps the ConfD frontend to hardware configuration.

```
[2023-09-13 15:06:26.363601]: (agent) : calling confd_init
(...)
```

```
[2023-09-13 15:06:26.373322]: Connecting CDB data socket for config data retrieval: 127.0.0.1
[2023-09-13 15:06:26.373589]: Connecting MAAPI socket for transactions: 127.0.0.1
[2023-09-13 15:06:26.373763]: Subscribing to CDB events 127.0.0.1
[2023-09-13 15:06:26.373809]: (subscribe) : setting up subscriptions
[2023-09-13 15:06:26.374392]: (pre_poll) : Pre-Poll reading configurations
[2023-09-13 15:06:26.374632]: (get_components_list) : CARD-A-In
(...)
[2023-09-13 15:06:26.377165]: (get_optical_channel_configuration) :
[agent::get_optical_channel_configuration] [OCh OCH-A-Out-1] 191500000 launch power 0 using mode 0
[2023-09-13 15:06:26.377316]: (get_optical_channel_configuration) : [OCh OCH-A-Out-1] validated
191500000
[2023-09-13 15:06:26.377983]: (cdb_write_operational) : [agent::cdb_write_operational] power: 0
[2023-09-13 15:06:26.378015]: (cdb_write_operational) : [agent::cdb_write_operational] freq.:
191500000
[2023-09-13 15:06:26.378028]: (cdb_write_operational) : [agent::cdb_write_operational] mode.: 0
[2023-09-13 15:06:26.378533]: (get_optical_channel_configuration) :
[agent::get_optical_channel_configuration] [OCh OCH-A-Out-2] 191500000 launch power 0 using mode 0
[2023-09-13 15:06:26.378731]: (get_optical_channel_configuration) : [OCh OCH-A-Out-2] validated
191500000
[2023-09-13 15:06:26.379378]: (cdb_write_operational) : [agent::cdb_write_operational] power: 0
[2023-09-13 15:06:26.379398]: (cdb_write_operational) : [agent::cdb_write_operational] freq.:
191500000
[2023-09-13 15:06:26.379410]: (cdb_write_operational) : [agent::cdb_write_operational] mode.: 0
[2023-09-13 15:06:26.379500]: (pre_poll) : Pre-Poll reading operational modes
[2023-09-13 15:06:26.380075]: (get_operational_modes) : mode 100
```

The initial startup has taken 8ms, which is several orders of magnitude faster than the overall agent startup time.

12.6.2 Discovery Latency

```
/opt/confd/bin/netconf-console -u USER -p PASSWD  --host=127.0.0.1 --port=830
--get --with-defaults='explicit' -x "/components/component"

time `./netconf_components | wc `

real    0m0,375s
user    0m0,046s
sys     0m0,047s
```

469 lines

12.6.3 Operational Mode Characterization

```
Example:
/opt/confd/bin/netconf-console -u USER -p PASS  --host=127.0.0.1 --port=830  -
-get -x "/operational-modes"
```

Output:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <data>
    <operational-modes xmlns="http://example.net/yang/openconfig-terminal-device-properties">
      <mode-descriptor>
        <mode-id>100</mode-id>
        <state>
          <mode-id>100</mode-id>
          <mode-type xmlns:oc-opt-term-prop-types="http://example.net/yang/openconfig-terminal-
device-property-types">oc-opt-term-prop-types:TRANSCEIVER_MODE_TYPE_EXPLICIT</mode-type>
        </state>
        <explicit-mode>
```

```
          <operational-mode-capabilities>
            <state>
              <modulation-format>oc-opt-term-prop-types:MODULATION_FORMAT_QPSK</modulation-format>
              <bit-rate xmlns:oc-opt-types="http://openconfig.net/yang/transport-types">oc-opt-
types:TRIB_RATE_40G</bit-rate>
              <baud-rate>20000000000.0</baud-rate>
              <optical-channel-spectrum-width>50.0</optical-channel-spectrum-width>
              <min-tx-osnr>40.0</min-tx-osnr>
              <min-rx-osnr>33.0</min-rx-osnr>
              <min-input-power>0.0</min-input-power>
              <max-input-power>9.0</max-input-power>
              <max-chromatic-dispersion>200.0</max-chromatic-dispersion>
              <max-differential-group-delay>1.0</max-differential-group-delay>
              <max-polarization-dependent-loss>0.12</max-polarization-dependent-loss>
            </state>
            <fec>
              <state>
                <fec-coding>oc-opt-term-prop-types:FEC_HD</fec-coding>
                <coding-overhead>7.0</coding-overhead>
                <coding-gain>8.53</coding-gain>
                <pre-fec-ber-threshold>0.000000000001</pre-fec-ber-threshold>
              </state>
            </fec>
            <penalties>
              <penalty>
                <parameter-and-unit xmlns:oc-opt-term-prop-types="http://example.net/yang/openconfig-
terminal-device-property-types">oc-opt-term-prop-types:CD_PS_NM</parameter-and-unit>
                <up-to-boundary>800.0</up-to-boundary>
                <state>
                  <parameter-and-unit xmlns:oc-opt-term-prop-
types="http://example.net/yang/openconfig-terminal-device-property-types">oc-opt-term-prop-
types:CD_PS_NM</parameter-and-unit>
                  <up-to-boundary>800.0</up-to-boundary>
                  <penalty-value>10.0</penalty-value>
                </state>
              </penalty>
              <penalty>
                <parameter-and-unit xmlns:oc-opt-term-prop-types="http://example.net/yang/openconfig-
terminal-device-property-types">oc-opt-term-prop-types:PMD_PS</parameter-and-unit>
                <up-to-boundary>0.77</up-to-boundary>
                <state>
                  <parameter-and-unit xmlns:oc-opt-term-prop-
types="http://example.net/yang/openconfig-terminal-device-property-types">oc-opt-term-prop-
types:PMD_PS</parameter-and-unit>
                  <up-to-boundary>0.77</up-to-boundary>
                  <penalty-value>1.0</penalty-value>
                </state>
              </penalty>
              <penalty>
                <parameter-and-unit xmlns:oc-opt-term-prop-types="http://example.net/yang/openconfig-
terminal-device-property-types">oc-opt-term-prop-types:PDL_DB</parameter-and-unit>
                <up-to-boundary>0.12</up-to-boundary>
                <state>
                  <parameter-and-unit xmlns:oc-opt-term-prop-
types="http://example.net/yang/openconfig-terminal-device-property-types">oc-opt-term-prop-
types:PDL_DB</parameter-and-unit>
                  <up-to-boundary>0.12</up-to-boundary>
                  <penalty-value>1.0</penalty-value>
                </state>
              </penalty>
            </penalties>
            <filter>
              <state>
                <pulse-shaping-type>oc-opt-term-prop-types:OFF</pulse-shaping-type>
              </state>
            </filter>
          </operational-mode-capabilities>
          <optical-channel-config-value-constraints>
            <state>
              <min-central-frequency>191675804</min-central-frequency>
              <max-central-frequency>205281058</max-central-frequency>
              <grid-type xmlns:oc-opt-term-prop-types="http://example.net/yang/openconfig-terminal-
device-property-types">oc-opt-term-prop-types:FLEX</grid-type>
              <adjustment-granularity xmlns:oc-opt-term-prop-
types="http://example.net/yang/openconfig-terminal-device-property-types">oc-opt-term-prop-
```

```
types:G_50GHZ</adjustment-granularity>
              <min-channel-spacing>50.0</min-channel-spacing>
              <min-output-power>0.0</min-output-power>
              <max-output-power>25.0</max-output-power>
            </state>
          </optical-channel-config-value-constraints>
        </explicit-mode>
      </mode-descriptor>
    </operational-modes>
  </data>
</rpc-reply>
```

```
real    0m  0,323s
user    0m  0,147s
sys     0m  0,025s
```

this particular reply contains

- 84 lines (XML encoded, pretty printed)
- 4837 characters

12.6.4 Transaction Delay

```
/opt/confd/bin/netconf-console -u USER -p PASSWD  --host=127.0.0.1 --port=830
--db=running --edit-config=payload.xml
```

Request:

```
<components xmlns='http://openconfig.net/yang/platform'>
        <component operation='merge'>
                <name>OCH-A-Out-1</name>
                <oc-opt-term:optical-channel  xmlns:oc-opt-term='http://openconfig.net/yang/terminal-
device'>
                        <oc-opt-term:config>
                                <oc-opt-term:frequency>195300000</oc-opt-term:frequency>
                                <oc-opt-term:target-output-power>10</oc-opt-term:target-output-power>
                                <oc-opt-term:operational-mode>100</oc-opt-term:operational-mode>
                        </oc-opt-term:config>
                </oc-opt-term:optical-channel>
        </component>
</components>
```

Response:

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="1">
  <ok/>
</rpc-reply>
```

```
real    0m  0,268s
user    0m  0,010s
sys     0m  0,009s
```

13 SONIC-BASED PACKET OPTICAL NODE

13.1 COMPONENT ARCHITECTURE

The Software for Open Networking in the Cloud, i.e., SONiC, [SONIC] is considered as the Network Operating System (NOS) to be deployed on packet-optical IPoWDM nodes operated in metro/aggregation networks. Within the B5G-OPEN project, SONiC has been extended with several components provided in the form of docker containers: (1) a REST-based interface to get status and perform configurations of optical pluggables and other node's interfaces as described in Sec. 4.8 (by CNIT); (2) a docker container running the OpenConfig agent; (3) a docker container exposing P4 runtime interface toward the packet controller. Furthermore, in B5G-OPEN we rely on the available SONiC BGP implementation.

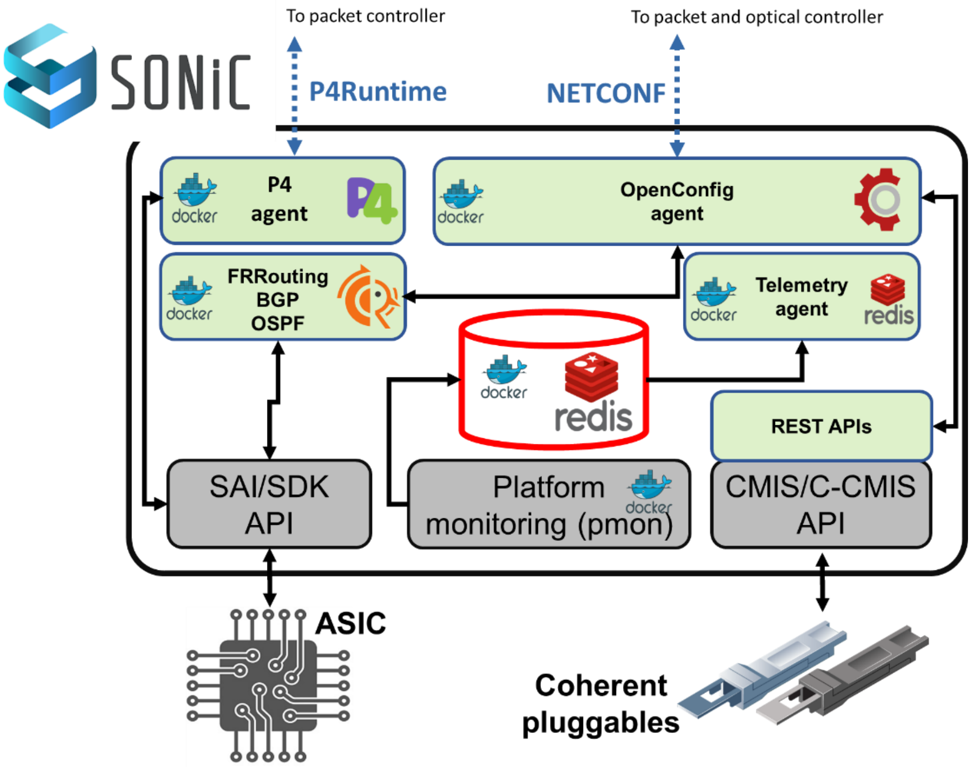


Figure 13.1: internal architecture of the packet optical node.

13.2 INTERFACE SPECIFICATIONS

13.2.1 Toward the SDN optical controller

From OpenConfig agent to the SDN optical controller: the OpenConfig agent exposes a YANG-based model to the SDN optical controller that is consumed through NETCONF.

The OpenConfig agent, inherited from the metro-haul project, has been deployed in the SONiC-based device and extended on its SBI to map the commands received by the SDN optical controller toward the REST APIs docker deployed in the NOS, see Sec. 4.8 for details on this interface interface. For example, when the SDN controller requires to activate a coherent pluggable the PUT method is used.

13.2.2 Toward the SDN packet controller

Two different interfaces are exposed toward the packet controller.

From P4 agent toward SDN packet controller: we have deployed on the SONiC-based device a container running the software switch BMW2 that exposes on its NBI the P4 runtime interface. Such container is configured to acquire a set of the physical interfaces of the device and includes them in the P4 pipeline. Validation of this interface is documented in [Gio23].

From BGP FRRouting application toward the SDN packet controller: for this interface case we rely on the implementation of the BGP FRRouting application available in the reference SONiC distribution, that applies the received layer 3 configurations to the device interfaces. Such configurations are sent by the packet controller to the OpenConfig agent that, in turn, translates it into FRRouting commands. Specifically, on top of the FRRouting application we have developed the /Sonic/bgp/… REST endpoints that can be invoked by the OpenConfig agent or even directly by the packet controller. Then the FRRouting app interacts with the SAI/SDK APIs provided by the NOS to read and write the packet interface configuration.

13.2.3 Toward the telemetry system

A docker container including the telemetry agent can be deployed on the SONiC-based device and integrated with the REDIS database residing in the device that acts as source of data. Once agent deployment the interface toward the centralized telemetry system is the same described in Se. 4.7, based on gRPC.

13.2.4 SBI toward SONiC native features

The several agents deployed on the SONiC NOS interface with the system calls using the REST interfaces illustrated in Sec. 4.8. Specific endpoints have been deployed to be used by the several agents.

- Endpoints /Sonic/Interface are used by the FRRouting app to configure IP properties.
- Endpoints /Sonic/InterfaceConfig are used by the FRRouting app to configure the port speed
- Endpoints /Sonic/InterfaceStatus are used by the FRRouting app to retrieve interface state
- Endpoints /Sonic/bgp are used by the FRRouting app to configure BGP
- Endpoints /Sonic/TransceiverConfig are used by the OpenConfig agent to configure and monitor the coherent optical pluggables

However, each endpoint can also be directly invoked externally, and represents therefore a configuration interface of the SONiC-based device. For example, the packet controller could directly invoke the endpoints for performing IP of BGP configurations.

These REST endpoints are built upon different interfaces offered by the SONiC NOS. As an example, the /Sonic/TransceiverConfig operates on top of the CMIS/C-CMIS APIs to actually read and write the coherent pluggable configuration; while the /Sonic/Interface endpoints interact with the SAI/SDK APIs to read and write the packet interface configuration.

13.3 Component integration

The following components are integrated with the SONiC-based packet-optical node:

- The agents deployed on the NOS, i.e., the P4 agent and the OpenConfig agent. Such agents are deployed on the device in the form of docker containers and exposes their interfaces toward the packet and the optical controllers.
- The optical controller is integrated by means of the interfaces exposed by the OpenConfig agent.
- The packet controller is integrated through the utilization of the FRRouting application deployed on the SONiC NOS and through the direct utilization of the REST endpoints described in Sec. 4.8 and Sec. 16.2.4.

13.4 Functional validation

The performed functional validation concentrated on the multi-layer traffic switching. Specifically, a packet traffic has been generated and injected on packet interfaces, routed coherent modules, and finally received by the traffic generator to collect statistics. The considered testbed is reported below including: a Spirent traffic generator, an EdgeCore whitebox and one Lumentum ROADM.

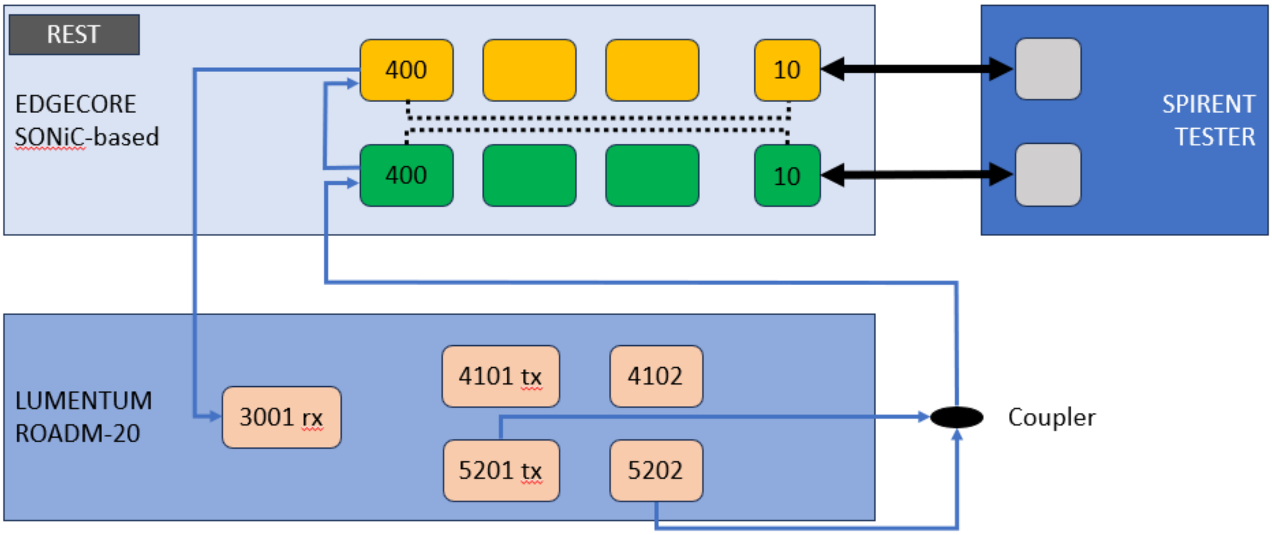


Figure 13.2: testbed environment for functional validation.

During the functional validation only one switch was available at the CNIT premises. For this reason, we have created two distinct VLANs on the switch (green and yellow in the figure) where each VLAN includes one 10 Gbps interface (used to connect the Spirent traffic generator), and three QSFP-DD slots, where one of those slots is equipped with a 400 Gbps coherent optical pluggables. The coherent pluggables are connected back-to-back, so that traffic incoming on the yellow VLAN is transmitted to the green VLAN.

Traffic is generated by the tester and injected on the yellow VLAN; the yellow coherent pluggable forwards the traffic to the green VLAN. The Lumentum ROADM-20 is inserted in between the two coherent pluggables (only in one of the two directions). This allow us to dynamically change the optical path (e.g., using exit port 5201 or 5202), indeed we are also interested to understand the duration of traffic disruption generated by this kind of events (i.e., change of optical path by keeping the same optical channel).

13.5 Roadmap

White box equipped with 400 ZR/ZR+ transceivers are currently available at CNIT, TIM, and TID labs. Integration performed with OpenConfig agent has been performed at CNIT, including the functional validation. The measurements reported in Sec. 16.6.1 on the ZR/ZR+ performance have been executed at TIM. The measurements reported in Sec. 16.6.2 for the estimation of the spectrum width of the channel generated by the coherent pluggables has been performed at CNIT exploiting also the box provided by TIM.

At the time of writing, the key features for the packet-optical node have been introduced and validated with bilateral integrations. In the remaining of WP4, the ongoing activities are:

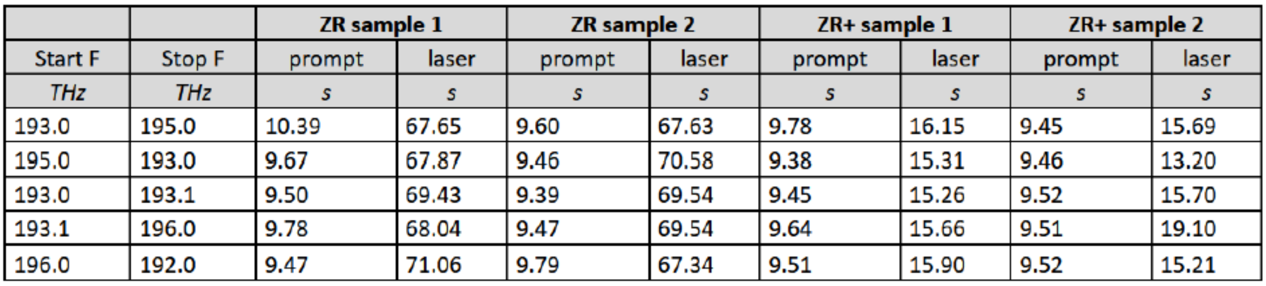
- test optical coherent pluggables performance in traffic recovery scenarios.
- automate the generation of the OpenConfig YANG model.
- integration with the telemetry agent.

13.6 KPIs

The considered KPIs concentrate on the performance of the coherent optical pluggables installed in a whitebox Edgecore 32x 400GbE QSFP56-DD switch, system ONIE, chipset BCM56880 Trident 4 12.8 Tb/s equipped with 400ZR and 400ZR+ coherent pluggable modules. The white box runs the open-source NOS SONiC_20220819.

13.6.1 ZR and ZR+ frequency configuration

The performance of 400ZR/ZR+ transceivers has been evaluated in terms of configuration time. Specifically, the experiment consists in the variation of the central frequency, it starts with an end-to-end connectivity established using the central frequency value “Start F” that is modified to the value “Stop F” [Morro23]. The frequency shifts have been performed considering both nearby frequencies and distant frequencies in order to check whether the reconfiguration time depends on the frequency gap to be covered. For each experiment, two intervals have been collected: the first (named prompt in the table) is the time interval taken by SONiC to receive the command and return the prompt after a frequency change command; the second (named laser in the table) is the time interval passed from the command issue and the signal detection at the RX.



		ZR sample 1		**ZR sample 2**		**ZR+ sample 1**		**ZR+ sample 2**	
Start F	Stop F	prompt	laser	prompt	laser	prompt	laser	prompt	laser
THz	*THz*	*s*	*s*	*s*	*s*	*s*	*s*	*s*	*s*
193.0	195.0	10.39	67.65	9.60	67.63	9.78	16.15	9.45	15.69
195.0	193.0	9.67	67.87	9.46	70.58	9.38	15.31	9.46	13.20
193.0	193.1	9.50	69.43	9.39	69.54	9.45	15.26	9.52	15.70
193.1	196.0	9.78	68.04	9.47	69.54	9.64	15.66	9.51	19.10
196.0	192.0	9.47	71.06	9.79	67.34	9.51	15.90	9.52	15.21

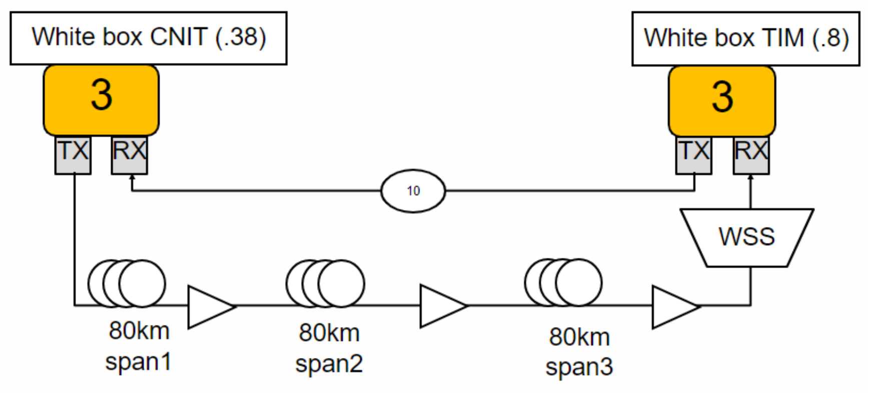
Figure 13.3: control performance of coherent pluggable modules in IPoWDM whitebox.

As a first remark it is important to say that a time of around one minute is required Analysing the results collected using the two classes of pluggable transponders (e.g., the ZR and ZR+ modules), the modules present considerable differences. Specifically, ZR+ modules guarantee a faster laser activation time. Moreover, the table shows that the (re)-configuration time interval does not depend on the gap between the starting and stop frequencies. In addition, the collected results show that modules of the same type present similar behavior, showing a stable and uniform configuration time.

13.6.2 ZR+ frequency slot evaluation

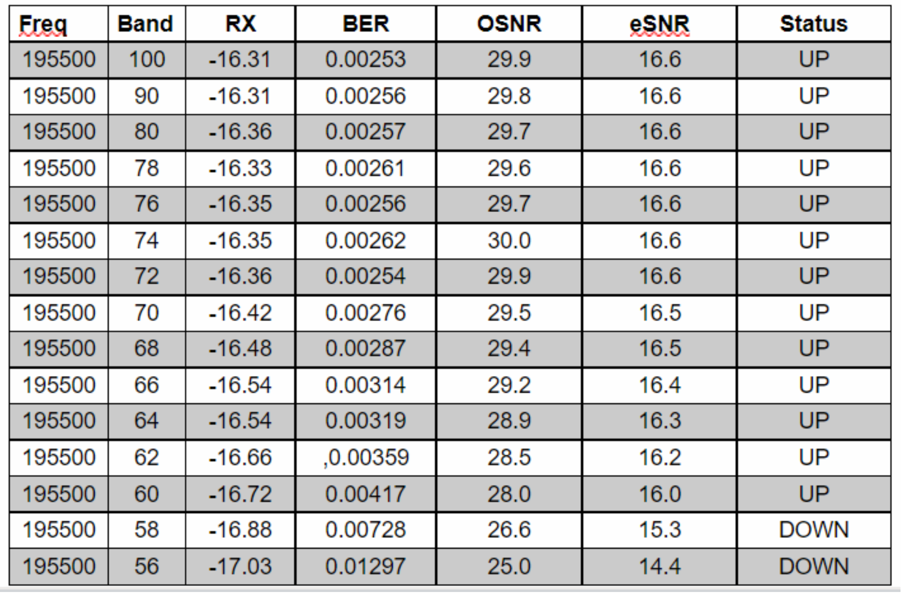
The CMIS driver of the coherent pluggables allows to configure the channel using two different grids namely: 100 GHz and 75 GHz. For example, with the 100 GHz grid if the specified central frequency does not exactly match the frequency grid, the system automatically round the specified central frequency to match the grid.

In this environment, we performed a set of experiments to understand which is the actual width of the of the 400 Gbps optical channel. For this experiment we have used the setup illustrated in the figure below, using coherent optical pluggables provided by Cisco (model 400G QSFP-DD High-Power).



In the illustrated setup, the WSS is initially configured at the proper central frequency and 100 GHz of width. The width is gradually narrowed, while monitoring the QoT of the channel on the receiver side (i.e, BER, OSNR, eSNR and STATUS of the interface). The following table illustrates the experimental results.

Table 6: Experimental results of the ZR+ pluggable evaluation



Freq	Band	RX	BER	OSNR	eSNR	Status
195500	100	-16.31	0.00253	29.9	16.6	UP
195500	90	-16.31	0.00256	29.8	16.6	UP
195500	80	-16.36	0.00257	29.7	16.6	UP
195500	78	-16.33	0.00261	29.6	16.6	UP
195500	76	-16.35	0.00256	29.7	16.6	UP
195500	74	-16.35	0.00262	30.0	16.6	UP
195500	72	-16.36	0.00254	29.9	16.6	UP
195500	70	-16.42	0.00276	29.5	16.5	UP
195500	68	-16.48	0.00287	29.4	16.5	UP
195500	66	-16.54	0.00314	29.2	16.4	UP
195500	64	-16.54	0.00319	28.9	16.3	UP
195500	62	-16.66	,0.00359	28.5	16.2	UP
195500	60	-16.72	0.00417	28.0	16.0	UP
195500	58	-16.88	0.00728	26.6	15.3	DOWN
195500	56	-17.03	0.01297	25.0	14.4	DOWN

From the results illustrated in the table, it is clear that the QoT of the connectivity does not degrade up to the width of 68 GHz, and the channel remains up till the width of 60 GHz. This result will help to save spectrum resources in the network, because the optical SDN controller could allocate a spectrum of 6 slots in the flexible grid (i.e., 75 GHz).

14 OpenROADM Agent

The Adtran OpenROADM agent is creating OpenROADM Netconf APIs for Adtran FSP 3000 CloudConnect devices. Depending on the cards in a single card NE, multi-card Ne or multi-shelf Ne, the agent is creating multiple northbound APIs for a single managed network element (NE). These NETCONF APIs correspond to the RDM (ROADM), XPDR (Transponder, Muxponder, Flexponder, colored Pluggables), or ILA (inline amplifier) data models of OpenROADM.

The OpenROADM agent was developed in funded research projects OTB-5G+ and AI-NET-PROTECT, and extended in B5G-OPEN. The agent is based on several open-source projects and software engineering concepts:

- OpenDaylight libraries at its core
- MD-SAL/ADVAnced code generation from YANG models using Java annotations
- Clean, declarative, event-based, functional, asynchronous APIs for YANG-bound data flow
- CPython-integration via Jep for interactive data analysis and automation scripting, including support for Jupyter notebooks
- completely Gradle-managed application life-cycles

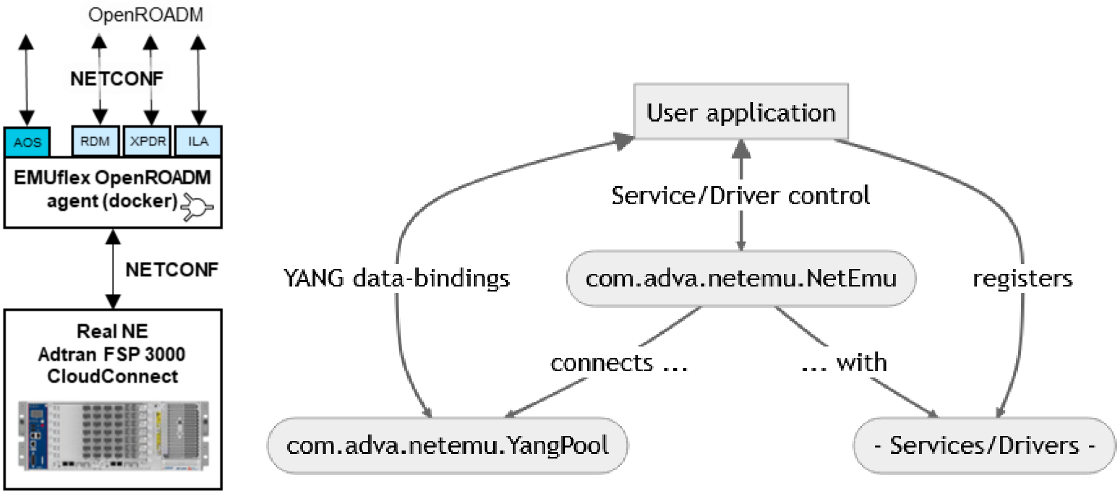
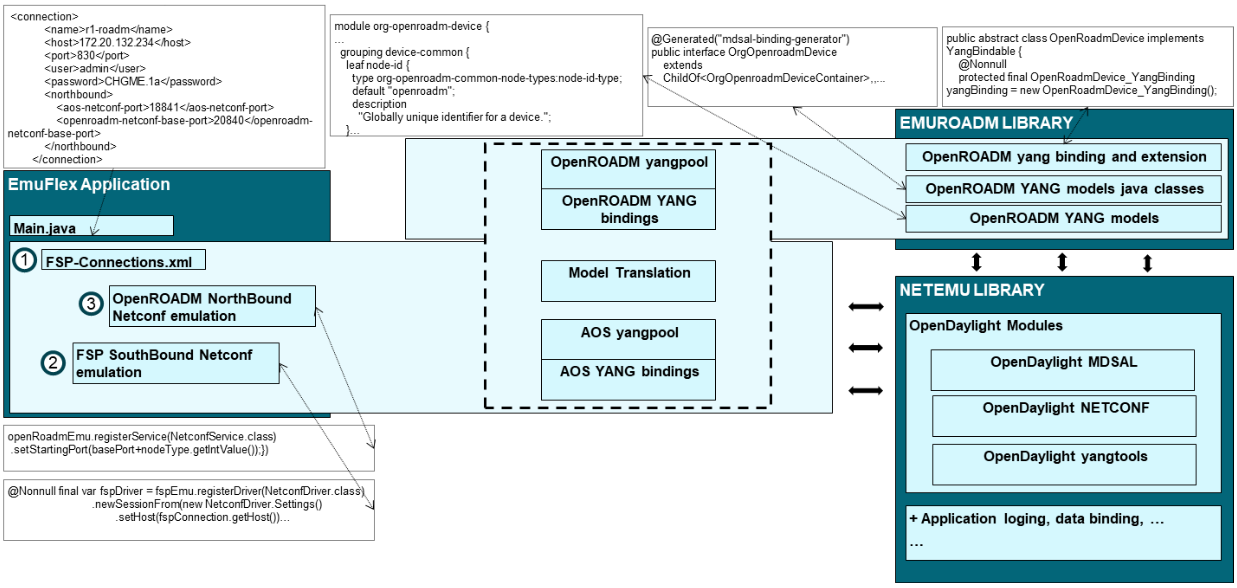


Figure 14.1: OpenROADM agent architecture.

The OpenROADM agent is based on a library named NetEmu. NETEMU. NETEMU's API design draws heavily from the modular structure of CESNET's sysrepo/netopeer2 C libraries and tools for NETCONF/YANG application development, which has already been described.

NETEMU is using open-source component from OpenDaylight, namely MD-SAL, the NETCONF, and yangtools. In addition to NETEMU, a EMUROADM library was developed as helper library for OpenROADM yang data models. The agent itself is based on the EmuFlex application which creates connections to one or multiple NEs as configured in the FSP-connections.xml file, generates Adran Operating System (AOS) yang data store, the corresponding OpenROADM data bindings, and allows the definition of data bindings.



```
<connection>
        <name>r1-roadm</name>
        <host>172.20.132.234</host>
        <port>830</port>
        <user>admin</user>
        <password>CHGME.1a</password>
        <northbound>
            <aos-netconf-port>18841</aos-netconf-port>
            <openroadm-netconf-base-port>20840</openroadm-netconf-base-port>
        </northbound>
    </connection>
```

```
module org-openroadm-device {
...
 grouping device-common {
   leaf node-id {
     type org-openroadm-common-node-types:node-id-type;
     default "openroadm";
     description
       "Globally unique identifier for a device.";
   }...
```

```
@Generated("mdsal-binding-generator")
public interface OrgOpenroadmDevice
    extends
    ChildOf<OrgOpenroadmDeviceContainer>,,...
```

```
public abstract class OpenRoadmDevice implements YangBindable {
    @Nonnull
    protected final OpenRoadmDevice_YangBinding yangBinding = new OpenRoadmDevice_YangBinding();
```

```
openRoadmEmu.registerService(NetconfService.class)
 .setStartingPort(basePort+nodeType.getIntValue());})
```

```
@Nonnull final var fspDriver = fspEmu.registerDriver(NetconfDriver.class)
                            .newSessionFrom(new NetconfDriver.Settings()
                                .setHost(fspConnection.getHost())...
```

Figure 14.2: OpenROADM implemented over NetEmu

15 AI/ML MODELS FOR PSD AND POWER MANAGEMENT

This component is a machine learning application towards augmented optical networks and is called “Automatic power correction”. Optical network is like photography before autofocus, where you needed to find a sweet spot of the optical lens before taking a picture otherwise it leaded to a blurry photography. The nonlinear behavior of optical transmission makes picking the best operating power tricky as there is a sweet spot to find between linear and nonlinear effects. It is the so-called nonlinear threshold. This sweet spot allows to operate in the weakly nonlinear regime.

Monitoring linear and nonlinear noise is quite complex in a live network which makes it attractive for machine learning applications. Our idea is to monitor an optical power spectral density which contains both linear and nonlinear impairments and the amount of linear and/or nonlinear effects gives a different shape of this optical spectrum. For extreme conditions like pure linear transmission regime or high nonlinear regime, a human eye can recognize the different shapes and classify them. For more intermediate regimes (i.e. where optical network operates), it is actually very difficult to distinguish for the human eye, but artificial neural network (ANN) is performing very well. In addition, the ANN can predict the optimal power correction to be applied to operate in the sweet spot of the nonlinear threshold.

The component leverages 3 main steps:
- monitoring an optical power spectral density of a given lightpath
- using telemetry to send it to the monitoring database
- AI/ML application for automated power reconfiguration, which is the main component module.

15.1 COMPONENT ARCHITECTURE

The automatic power correction component is a management plane module which allows to optimize a given ligthpath configuration. It works very closely with the physical layer as it influences the launch power of the transmitted signal. It can provide recommendations showing the power correction to be applied and the expected gain, then let the end user trigger the optimization reconfiguration.

The component itself is an AI/ML application that processes the optical power spectrum density (PSD) to predict an estimated power correction ($\widehat{\Delta P}$). To learn how to connect the inputs to targets, the ANN needs a lot of training examples in different operating conditions. We provided these examples by conducting many lab experiments. Once the neural network has been trained, we can start testing it. The ANN works with 1 hidden layer with 10 neurons and 1 output layer with 1 neuron. A neuron is a sequence of two operations: a weighted sum followed by a nonlinear manipulation. It is a fully connected ANN meaning that all neurons of the following layer take as input all neurons from the preceding layer. The hidden layer employs sigmoid function, and the output layer uses the identity function.

We implement a pre-processing step to normalize the PSD in the range [0, 1]. Figure 15-1 shows the architecture of the component.

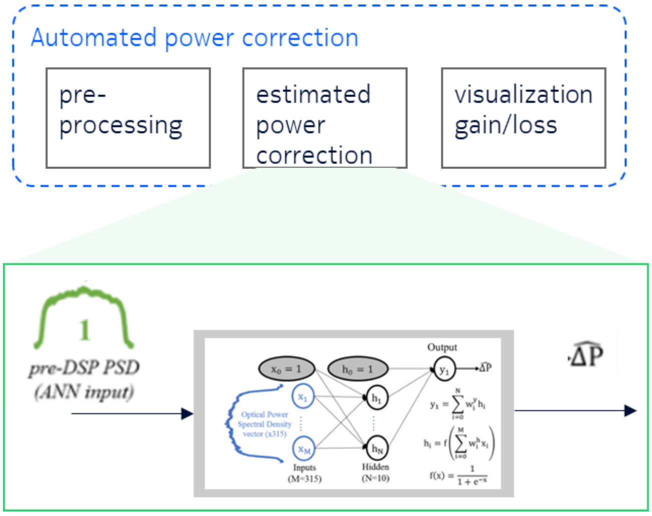

Automated power correction
pre-processing
estimated power correction
visualization gain/loss
1
pre-DSP PSD (ANN input)
Output
$x_0 = 1$
$h_0 = 1$
y_1
$\widehat{\Delta P}$
x_1
h_1
Optical Power Spectral Density vector (x315)
x_M
h_N
Inputs (M=315)
Hidden (N=10)
$y_1 = \sum_{i=0}^{N} w_i^y h_i$
$h_i = f\left(\sum_{i=0}^{M} w_i^h x_i\right)$
$f(x) = \frac{1}{1+e^{-x}}$
$\widehat{\Delta P}$

Figure 15-1: architecture of the automatic power correction component

Then, it can expose the expected gain. For instance, close to the nonlinear threshold, the expected gain can be up to 1dB gain in signal-to-noise ratio and a negligible loss in transmission quality.

15.2 Interfaces

The PSD component will be used as a standalone component. However, it is integrated with an optical mesh network to collect monitored PSD data and a telemetry database to feed the automatic power correction component, as shown in the figure below.

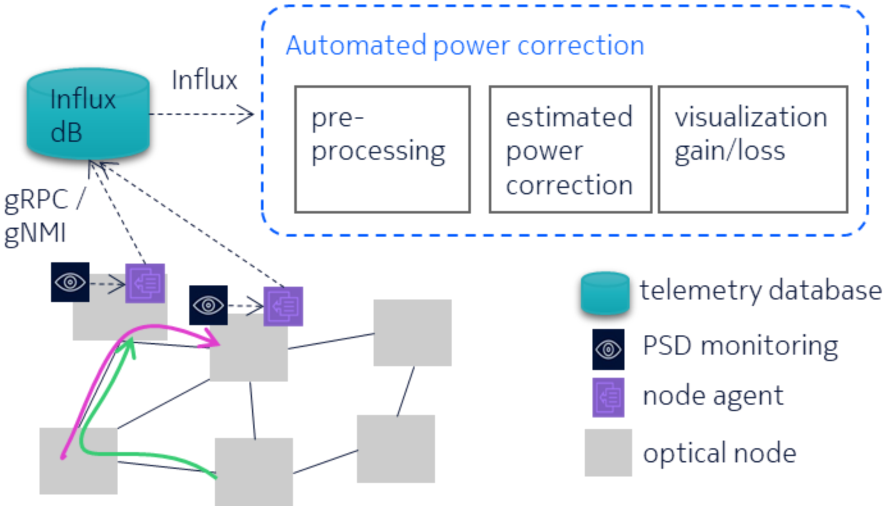


Figure 15-2: interconnection of the automatic power correction component with optical mesh network and telemetry database

As illustrated examples, the green and pink lightpaths in Figure 15-2 are leveraging the component to optimize their performance. Each of them has the following workflow. First, an optical spectrum density is monitored in the optical node thanks to an optical spectrum analyzer or equivalent hardware. Then, the optical node agent (responsible for communicating with the control and management plane) sends to the telemetry database the monitored PSD via a gRPC/gNMI interface. Once the database is populated, the automatic power correction component can read from it and can provide recommendations for power reconfiguration with the associated gain.

15.3 Functional validation

To validate the component, the following tests could be performed:

Test	Description
Training and validation of AI/ML	Generate one (or more) set of measurements or simulations with different operating conditions (e.g. different propagation lengths, fiber characteristics, channel loading, launched power, etc.)
Accuracy validation	Evaluate the error between known optimal powers and estimated ones
Component validation	Validate the AI/ML component by visualizing the expected SNR gain after applying the power correction

15.4 Component integration

This component will be used as a standalone component. Nevertheless, it is integrated with a physical layer mesh optical networks using telemetry to populate the database with optical power spectral density. This is needed for the functional validation of the component described in the previous subsection.

15.5 Roadmap

- Q3/2023: 2nd version of the PSD component, assessing compression rate vs accuracy.
- Q2/2024: 3rd version of the PSD component, evaluating performance in a networking environment.

15.6 Component KPIs

KPI	Definition	Methodology
Gain/loss of performance	The difference between the signal-to-noise ratio before and after using this AI/ML component.	2 SNR measurements
Scalability in terms of number of input points	The number of points required for the optical PSD to get a given accuracy	Setup scenarios to reduce the PSD size, e.g. scenario #1 could take 1 point every x samples; scenario #2 could take y samples located at the edges.
Compression rate	The complexity reduction of the ANN architecture (without retraining)	Setup scenarios

The gain / loss of performance of the component is investigated and the results are shown in Figure 15-3. The power correction can provide up to 1dB SNR gain and down to 0.15 dB SNR loss. We notice that the loss is close to the nonlinear region threshold and has negligible performance impact. In contrast, if we operate in the linear or nonlinear region, there is a subsequent gain to obtain by applying the power correction.

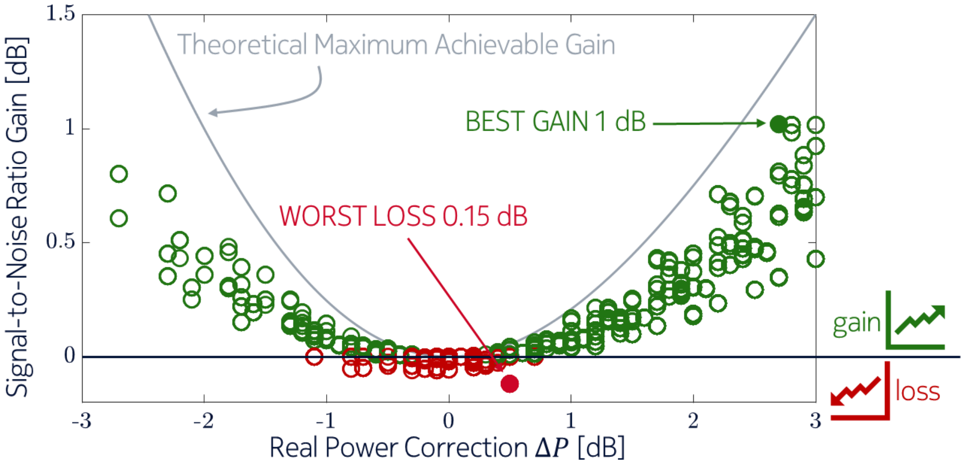


Figure 15-3: SNR gain/loss performance

We investigated the scalability of the component. To reach this goal we defined different scenarios to play with the number of inputs. A first scenario is to compress the input PSD by taking fewer points equally spaced. We investigated 3 different configurations with either 20, 100 and 315 input PSD points. As a figure of merit, we defined the error as the difference between the true power and the predicted output power. We plot in Fig. 15-4 the standard deviation of the error as a function of the number of input PSD points. As expected, when the number of PSD points is low, the standard deviation increases. For 20 PSD points, the error standard deviation is 1.67dB while for 315 PSD points it can decrease down to 0.41 dB. To maintain a reasonable performance and to stay close to the nonlinear threshold area, we can go down to 100 PSD points which gives 1dB error standard deviation for the investigated line configurations.

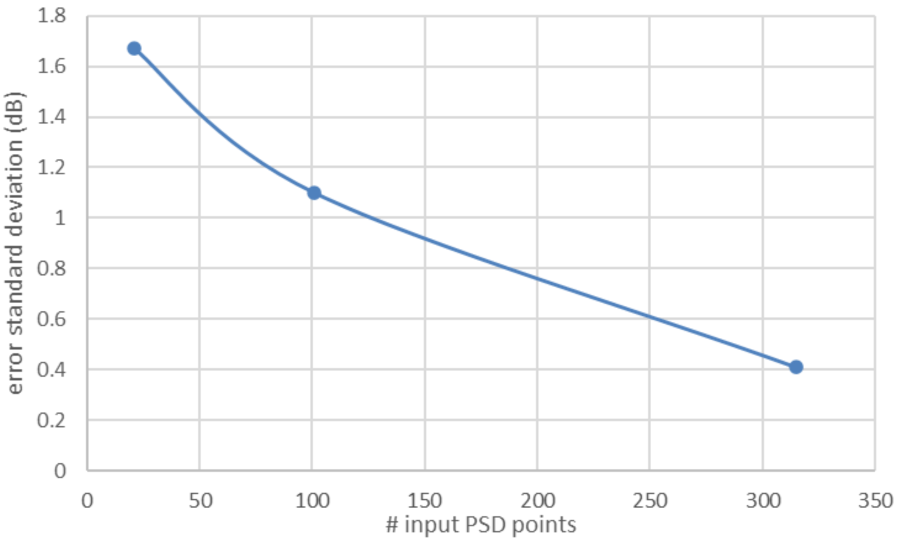


Figure 15-4: investigation of scalability of the component

A second scenario is to input only a part of the PSD. Since the nonlinearities are visible on the edge of the spectrum, one can think that removing the points corresponding to the center of the spectrum will not impact too much the performance of the component. This assumption will be investigated next.

16 Telemetry System

16.1 Introduction

B5G/OPEN distributed telemetry system integrates measurement and event data collection and supports intelligent data aggregation nearby data collection, so agents receive and analyze measurements before sending to a centralized manager.

16.2 Internal Workflow and interfaces

A detailed architecture of the proposed telemetry system is presented next where the internal architecture of telemetry agents inside node agents and the telemetry manager is shown.

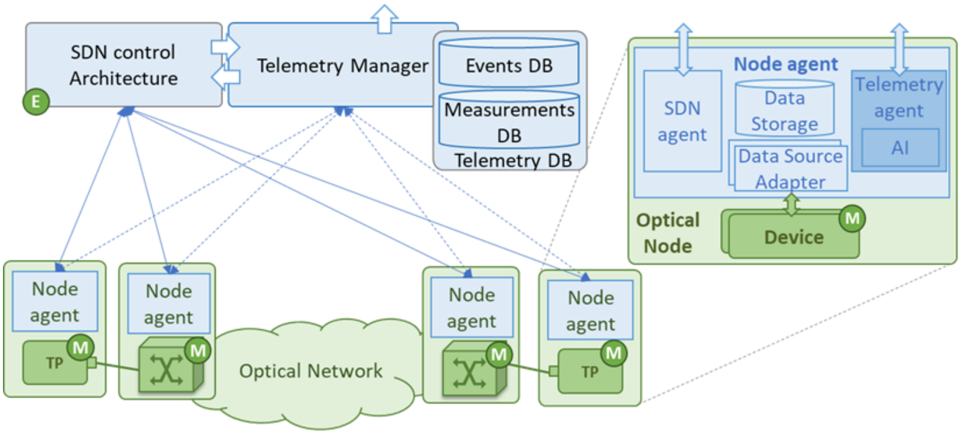


Figure 16.1: Proposed telemetry architecture

Let us describe a typical telemetry workflow valid for a wide range of use cases. The node agent includes modules (denoted data sources) that gather telemetry data from observation points in the optical nodes. Examples include optical spectrum analysers (OSA) in the ROADMs and data from digital signal processing, e.g., optical constellations, in the TPs. A telemetry adaptor has been developed, so data sources can export collected data to the telemetry system; specifically, the adaptor receives raw data from the data source and generates a structured JSON object, which is then published in the local Redis DB (labelled 1 in the figure). The periodicity for data collection can be configured within a defined range of values. A number of algorithms can be subscribed to the collected measurements. In this example, let us assume that only one algorithm is subscribed, which processes the measurements locally. Such processing might include doing: i) no transformation on the data (null algorithm); ii) some sort of data aggregation, feature extraction or data compression; or iii) some inference (e.g., for degradation detection). The output data (transformed or not) are sent to a gRPC interface module through the Redis DB (not shown in the figure) (2), which conveys the data to the telemetry manager. Because gRPC requires a previous definition of the data to be conveyed, our implementation defines a unique message of type *bytes*, which allows generalization of the telemetry data to be conveyed. Note that, although such encoding could largely increase the volume of data to be transported, intelligent data aggregation performed by telemetry agents could reduce such volume to a minimum.

In the telemetry manager, the data are received by a gRPC interface module that publishes them in the local Redis DB, so subscribed algorithms can receive them. The algorithms in the telemetry manager can implement functions related to data aggregation, inference, etc. Once processed, the output data is published in the local Redis DB (4) and can be stored in the telemetry DB (5) and/or be exported to external systems (6). Interestingly, algorithms in the telemetry manager can communicate with those in the telemetry agents using the gRPC interface (7-8). Examples of such communication include parameter tuning, among others.

16.3 Functional Validation

The following tests have been carried out to validate the telemetry system:

a. Generate measurements using the data source and the telemetry adaptor and verify that:
   1. they are received by the selected algorithm in the telemetry agent;
   2. they are sent to the telemetry manager and stored in the measurements DB.

b. Generate events with the SDN controller and verify that they are stored in the events DB.

16.4 Component Integration

The Telemetry system can collect measurements from any data source provided that they implement the telemetry adaptor. Examples of data sources for measurements are transponders, ROADMs, and OSAs. Examples of data sources for events are controllers and agents. In addition, the telemetry manager integrates also with any other management system using external delivery systems, like Kafka [KAFKA].

16.5 Roadmap

The Telemetry system has already been demonstrated at OFC 2023, where we showed integration with network devices from Nokia and Adtran, as well as with the CTTC's SDN controller.

16.6 Component KPIs

KPI	**Definition**	**Methodology**	**Results**
Optical Constellation Data Compression with Autoencoders	Compression technique applied to optical constellation measurements.	Measurements from the original and the processed sample are taken in the Telemetry Agent Samples are injected in the Telemetry Node with a defined rate.	625:1 with reconstruction error < 2%
Spectrum Dimensionality Reduction with Feature Extraction	Dimensionality Reduction technique applied to spectrum measurements.	Measurements from the original and the processed sample are taken in the Telemetry Agent	Compression rate for spectrum using features extraction: 7.5:1

		Samples are injected in the Telemetry Node with a defined rate.	

Optical Constellation Data Compression with Autoencoders

This KPI is related with the data compression while using streaming telemetry. To be able to handle the high process requirements of streaming telemetry and in particular with Optical Constellation, some compression techniques have to be applied. For this reason, we propose to use a type of neural network called AE that has two components: the *encoder*, which maps input data into a lower-dimensional *latent* space, and the *decoder*, which gets data from the latent space and reconstructs the original data back.

Results show a compression ratio of 625:1 with a reconstruction error that is negligible. To put some numbers, the AE *encoder* receives a sample of 10,000 symbols and generates a latent space Z of 32 symbols. This significantly size reduction enables the use of less demandant telemetry systems and less storage requirements to process and store all the data.

Spectrum Dimensionality Reduction with Feature Extraction

This KPI is also related to data compression but in this case spectrum samples and using a dimensionality reduction technique called Feature Extraction. This technique is intended to generate a set of features $\Phi(M)$ that characterize a measurement sample M. For instance, a sample of a 75 GHz channel that is represented with a list of 75 pairs assuming a GHz granularity, is processed to generate a set with 13 features. This features then can be used on failure detection without having to convey the whole spectrum sample.

17 MESARTHIM – FAILURE MANAGEMENT USING A SNR DIGITAL TWIN

The performance of optical devices can degrade because of aging and external causes like, for example, temperature variations. Such degradation might start with a low impact on the Quality of Transmission (QoT) of the supported lightpaths (soft-failure). However, it can degenerate into a hard-failure if the device itself is not repaired or replaced, or if an external cause responsible for the degradation is not properly addressed. MESARTHIM compares the QoT measured in the transponders with the one estimated using a QoT tool. Those deviations can be explained by changes in the value of input parameters of the QoT model representing the optical devices, like noise figure in optical amplifiers and reduced Optical Signal to Noise Ratio in the Wavelength Selective Switches. By applying reverse engineering, MESARTHIM estimates the value of those modelling parameters as a function of the observed QoT of the lightpaths.

17.1 WORKFLOW AND INTERFACES

Among the effects degrading the QoT within optical systems, we consider degradations arising from ROADMs and In-Line OAs, where a ROADM consists of WSSs and OAs. Both building blocks face aging and non-ideal conditions. For example, although OAs are considered robust devices, they also suffer time-varying effects, which might increase over time due to the aging of the amplifier building blocks. In addition, the Noise Figure (NF) is frequency-dependent and, as the allocation of the spectrum might become time-dependent. Therefore, the NF can be modelled as a time-frequency variation. The pump lasers of the EDFAs also present degradation, which can be adjusted thanks to internal control loops, but which still reduces the EDFA efficiency. For what concerns the WSSs, they might suffer temperature-dependent variations, which might lead to frequency shift over time; furthermore, individual channels can drift as well, and both effects can be highly detrimental in terms of QoT. In the context of this work, we consider gradual time-varying device degradations on OA and add/drop (A/D) WSSs in the ROADMs. Specifically, we consider that soft-failures can be explained by one of the following events in the modelling parameters: a) NF increase; b) maximum optical output power (P-max) decrease; and c) Optical SNR (OSNR) degradation caused by frequency drifts of the WSSs due to temperature fluctuation.
Our proposed architecture for soft-failure analysis is illustrated in Figure 17.1. The Telemetry manager stores a replica of the operational databases (DB) that are synchronized from the network controller. In addition, it collects measurements from the telemetry agents and stores them in the Telemetry DB. These measurements are used by MESARTHIM to: i) estimate those modelling parameters related to optical devices (resources); ii) analyse the evolution of the measured SNR and that of the modelling parameters to detect any degradation as soon as it appears; and iii) determine the severity of the degradation based on the foreseen impact on the performance of the lightpaths.
Figure 17.1 also sketches the MESARTHIM methodology implemented in the telemetry system. Specifically, the following building blocks can be identified: (1) the Surveillance block that analyses the SNR measurements and the value of modelling parameters to detect any meaningful degradation (e.g., by threshold crossing); (2) the Localization block that localizes the soft-failure; (3) the Find Modelling Configuration block that finds the most likely value of the modeling parameters of a given resource, so it results into SNR values of the lightpaths being supported by such resources similar to those that have been actually measured; (4) the soft-failure Identification block that, assuming a resource has been localized as the source of the soft-failure, finds what is the modeling parameter responsible for such failure; and (5) the Severity

Estimation block that estimates whether and when the soft-failure will degenerate into a hard-failure. In addition, two internal repositories are used: i) the Device Modeling Config DB with the evolution of the value of modeling parameters along time for every resource; and ii) the Network Diagnosis DB that stores historical data for analysis purposes. The MESARTHIM manager coordinates those blocks to achieve intelligent QoT analysis, as well as manages the interface with the QoT tool.

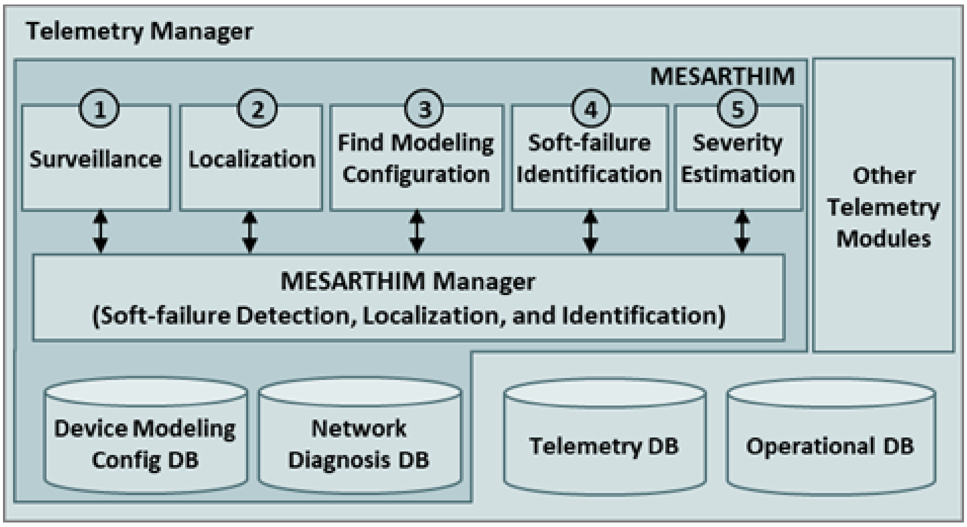


Figure 17.1: Mesarthim architecture and its relationship with the Telemetry System

17.2 FUNCTIONAL VALIDATION

The following tests have been carried out to validate Mesarthim:

- Verify that MESARTHIM gets the route of the right lightpath from the LSP DB
- Verify that MESARTHIM gets the right samples from the telemetry DB
- Verify that MESARTHIM is able to find the right configuration of parameters for different lightpaths.

17.3 COMPONENT INTEGRATION

MESARTHIM runs as part of the telemetry system, and it access:

- the telemetry measurements DB to analyse constellation samples.
- the Operational DB to get the route of the lightpaths.
- an external QoT tool

The results of the analysis are stored in an internal DB.

17.4 ROADMAP

OCATA has been already integrated with the telemetry system.

17.5 COMPONENT KPIS

KPI	Definition	Methodology	Results
Estimate the most likely modelling configuration	Relative average error of the modelling parameters estimation < 8%.	Test experimentally on a lightpath system with several spans. Produce degradations originated by optical filtering and fibre attenuation.	At each step, the module was able to explain the increment in the SNR of the lightpath by a reduction in the bandwidth in the related WSS or the NF of the OA. $R^2 > 0.986$
Anticipation of soft failures	> 15% through the estimation of modelling parameters w.r.t. SNR analysis.	Produce degradations for which the magnitude gradually increases with time	P-max degradation anticipated 15%, NF degradation 45% and WSS degradation 27%
Severity estimation	Severity estimation anticipation > 40%	Produce degradations for which the magnitude gradually increase with time	P-max estimation 42.8%, NF 62.8% and WSS 49.6%

18 OCATA - DIGITAL TWIN FOR THE OPTICAL TIME DOMAIN

OCATA is a deep learning-based digital twin for the optical time domain that is based on the concatenation of deep neural networks (DNN) modelling optical links and nodes, which facilitates representing lightpaths. The DNNs model linear and nonlinear noise, as well as optical filtering. Additional DNN-based models extract useful lightpath metrics, such as lightpath length, number of optical links and nonlinear fibre parameters. OCATA exhibits low complexity, thus making it ideal for real-time applications.

18.1 WORKFLOW AND INTERFACES

Figure 18.1 overviews the considered network architecture and will be used for describing the main workflow; for the sake of simplicity, only the directly involved elements, like a lightpath, a node controller and a sandbox are detailed, whereas other components have been omitted to better highlight the key concepts involved in this work. Lightpath *i* is considered as the entity under analysis, which is represented as a sequence of optical components (Tx, ROADMs, links, and Rx) supporting that lightpath.

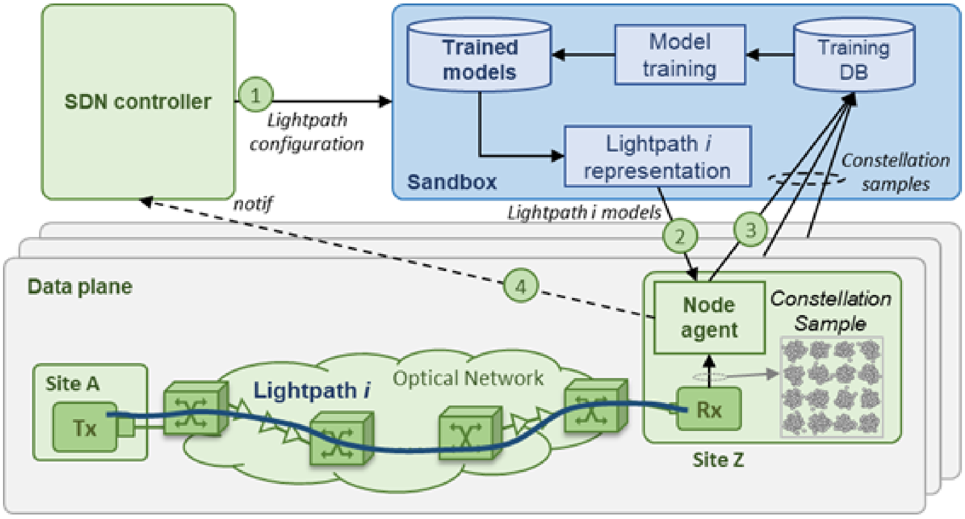


Figure 18.1: OCATA Reference Network Architecture

Figure 18.2 details the internal architecture of the sandbox domain and node agents. At set-up time, the SDN controller solves the Routing, Spectrum, and Transponder Assignment before configuring the involved devices to establish the lightpath. Then, after the lightpath is provisioned, the sandbox domain receives the lightpath's configuration from the SDN controller (labelled 1 in Figure 18.1), including its route on the optical network and some metrics. This configuration is used in the sandbox domain to set up an accurate representation of that lightpath to be set in the Rx agent (2). Such representation is defined as a sequence of pre-trained DNN-based models that emulate the behaviour of each individual optical component that the optical signal traverses. The role of the models is different depending on the physical element they characterize (Figure 18.2 (a)). For instance, the model for the Tx characterizes the output signal according to its specifications, whereas the models for intermediate elements (ROADMs and fibre links) propagate forward a set of features related to the signal's constellation. Specifically, intermediate components introduce distortion on the constellation as a result of LI and NLI noise. Finally, the Rx model receives the constellation features and performs additional actions before returning the output of the model. Additionally, any relevant change affecting the lightpath during its lifetime, e.g., path rerouting, needs to be notified to the sandbox, so as to adapt the lightpath's representation and avoid misleading diagnosis due to mismatch between the physical lightpath and its models.

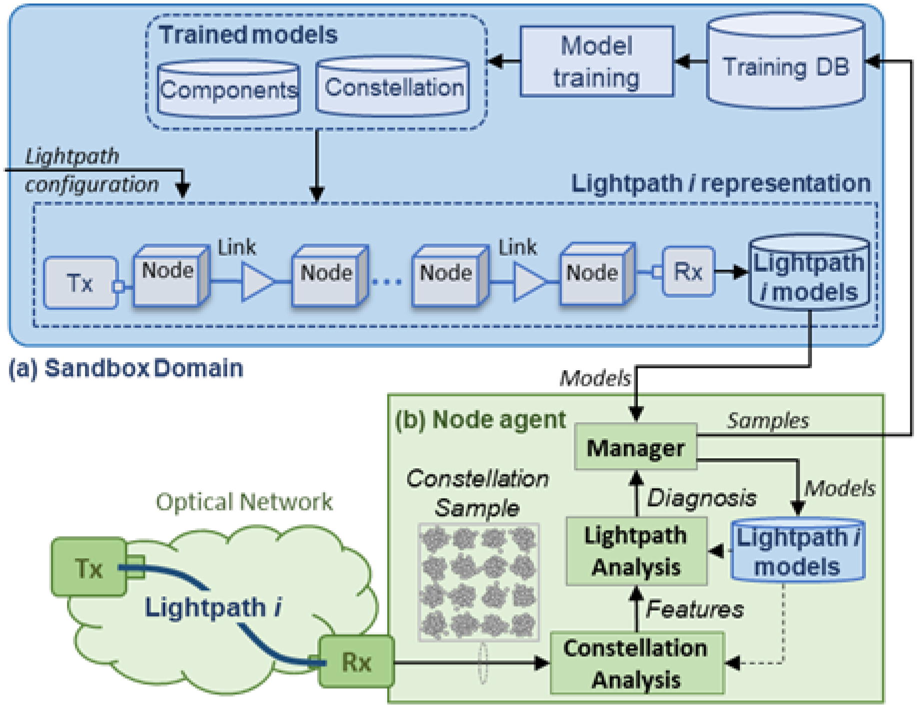


Figure 18.2: Details of sandbox (a) and node agent (b)

Both constellation and lightpath analysis require from models that characterize the monitored lightpath. Thus, anomaly detection based on comparing observed features and expected ones coming from lightpath's models can be carried out at the Rx. Note that this scheme highly reduces the amount of data to be sent to the centralized elements (3), as well as its computational demand for real-time data analysis purposes. Once the lightpath is set-up and the lightpath models are set in the Rx agent, they are used for analysis. With a predefined frequency, e.g., updated every 1s, the Rx samples the received constellation and gathers n IQ symbols. The sample is then processed by the constellation analysis block in the Rx agent (Figure 18.2 (b)). The aim of this block is to extract a set of relevant constellation features that facilitates posterior analysis, as well as compressing constellation data to be used for multiple purposes, such as model training. These features are obtained by means of both supervised and unsupervised statistics and ML-based techniques. Next, the lightpath analysis block processes the features extracted from the received constellation and analyses key lightpath's configuration metrics, such as length and/or power configuration. The result of this analysis produces a diagnostic report highlighting, e.g., whether some of the metrics does not follow the expected behaviour. The diagnostic report is processed by the manager block that implements a set of rules and generates notifications to the SDN controller depending on the diagnosis (4).

18.2 Functional Validation

The following tests have been carried out to validate OCATA:

- Verify that OCATA gets the route of the right lightpath from the LSP DB
- Verify that OCATA gets the right constellation samples from the telemetry DB
- Verify that OCATA determines the lightpath length and compares to that stored in the LSP DB.

18.3 Component Integration

OCATA components run inside the telemetry system and access: i) The telemetry measurements DB for model training purposes; ii) The LSP DB to get the route of the lightpaths and iii) The samples in the node agent from transponder data sources. The results of the analysis are stored in an internal DB.

It is expected to perform integration with the SDN controllers during the Year 3 of the project, to evaluate its applicability in a set of use cases.

18.4 Roadmap

OCATA has been already integrated with the telemetry system. A control loop integrating the telemetry system, with OCATA and the SDN controller is to be defined by the end of the second year of the project.

18.5 Component KPIs

KPI	**Definition**	**Methodology**	**Results**
Lightpath modelling error	Error between the optical constellations generated with OCATA with respect to real ones for the same lightpath configuration.	Test for different lightpath configurations, i.e., number of hops and total distance.	We observed negligible μ prediction errors (max error < 2%) independently of the link length. In contrast, σ max error is around 30% for low σ values although decreases when path length increases, becoming under 15%, which is, in general, a good enough performance to validate the OCATA models.
Running time	Reduction of running time > 2 orders of magnitude with respect to Split-Step Fourier Method (SSFM) simulation	Test for different lightpath configurations, i.e., number of hops and total distance. Measure the time needed to generate optical constellations (2048 symbols)	The experiments were performed on an Intel Core i5 CPU @ 2.67GHz computer with 8 GB RAM and running Windows 10 64-bit. The results show much better scalability of OCATA, which runs over 3 orders of magnitude faster than SSFM. In addition, the running time is only dependent on the number of hops and not on the distance as in SSFM.

19 CONCLUSIONS

This deliverable reflects the work done in WP4 during the second year of the project. A significant effort has been addressed to design the interfaces and protocols in order to enable the different components of the control plane to interwork. Selected interface functionality has been implemented and validated, and the different components are available for major integrations in the scope of WP5.

All tasks have contributed to the deliverable. The focus of T4.1 is on the implementation of SDN-based technologies and solutions to operate on packet-optical nodes based on SONiC and using coherent pluggable modules. Experimental activities have been also performed on data plane devices running SONiC to understand traditional protocol features and configurations. Task 4.2 deals with the design, development and validation of a generalized orchestration and control plane system for the B5G-OPEN multiband optical networks infrastructure, able to deploy and to manage the lifecycle of services. Developments are consolidated. These include SDN control for multiband optical networks with externalized physical layer impairment validation and the ONOS SDN controller has preliminary support for optical amplifiers, multi-band, flexible grid, ROADM intents and interfacing toward TAPI module and interfaces have been properly defined and implemented.

Finally, in Task *AI Empowered Network Operating System for Autonomous and Zero-Touch Networking*, the hierarchical and distributed telemetry system has been demonstrated, as well as algorithms for data processing, using sources of realistic data have been identified and now are available for testing different algorithms. The optical time domain digital twin (OCATA) has been tested using available experimental data. The digital twin allows the generation of expected optical signals after crossing optical devices, which allows modelling lightpaths and OCATA models are being used for knowledge sharing.

20 References

[CMIS] “CMIS” [Online]. Retrieved April 27, 2022, https://www.oiforum.com/wp-content/uploads/OIF-CMIS-05.2.pdf

[ENP] “e-Lighthouse Network Planner, from https://e-lighthouse.com/products/networkplanner

[EuCNC] B5G-OPEN, EuCNC Demo, June, 2023, Gothenburg, Sweden

[GioDemo23] A. Giorgetti, A. Sgambelluri, F. Cugini, E. Kosmatos, A. Stavdas, J. M. Martinez-Caro, P. Pavon, O. Gonzalez De Dios, R. Morro, L. Nadal, R. Casellas "Modular Control Plane Implementation for Disaggregated Optical Transport Networks with Multi-band Support", ECOC2023

[Gon22] O. Gonzalez de Dios et al, "MANTRA Whitepaper. IPoWDM convergent SDN architecture - Motivation, technical definition & challenges", Telecom Infra Project, August 2022, [Online] https://cdn.brandfolder.io/D8DI15S7/at/n85t9h48bqtkhm9k7tqbs9fv/TIP_OOPT_MANTRA_IP_over_DWDM_Whitepaper_-_Final_Version3.pdf

[GRPC22] gRPC Network Management Interface (gNMI) [online], Retrieved October 5, 2022, from https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md

[K8s] Kubernetes [Online]. Retrieved October 5, 2022, from https://kubernetes.io/docs/home/

[K8sAPI] Kubernetes API [Online]. Retrieved October 13, 2022, from https://kubernetes.io/docs/reference/kubernetes-api/

[KAFKA] [Online] https://kafka.apache.org/

[Metro-Haul] Metro-Haul "Definition of Use Cases, Service Requirements and KPIs", Project Deliverable D2.1, February 2018

[MUST] MUST Optical SDN Controller NBI Technical Requirements Document TIP OOPT PG - Version: 1.1 [Online] Retrieved October 5, 2022 from https://cdn.brandfolder.io/D8DI15S7/at/sp6tgqcpjp8rgsshf8pvmwpg/TIP_OOPT_MUST-Optical-SDN-Controller-NBI-Technical-Requirements-v11_FINAL_GREEN_ACCESS.pdf

[NSv16] "Framework for IETF Network Slices - draft-ietf-teas-ietf-network-slices" [Online]. Retrieved December 16, 2022, from https://datatracker.ietf.org/doc/draft-ietf-teas-ietf-network-slices/

[OCATA] M. Ruiz, D. Sequeira, and L. Velasco, "Deep Learning -based Real-Time Analysis of Lightpath Optical Constellations [Invited]," IEEE/OPTICA Journal of Optical Communications and Networking (JOCN), vol. 14, pp. C70-C81, 2022.

[Ofc23] Pol González, Ramon Casellas, Jose Pedreño-Manresa, Fabien Boitier, Behnam Shariati, Johannes K. Fischer, Marc Ruiz, Jaume Comellas, and Luis Velasco, "Distributed Architecture Supporting Intelligent Optical Measurement Aggregation and Streaming Event Telemetry", OFC2023

[OFCDemo] Pol González, Ramon Casellas, Jose Pedreño-Manresa, Fabien Boitier, Behnam Shariati, Johannes K. Fischer, Marc Ruiz, Jaume Comellas, and Luis Velasco, "Distributed Architecture Supporting Intelligent Optical Measurement Aggregation and Streaming Event Telemetry", OFC2023

[OIF] “OIF” [Online] https://www.oiforum.com/

[ONF] Open Networking Foundation [Online]. Retrieved October 5, 2022, from https://opennetworking.org/

[ONL] "Open Network Linux" [Online] http://opennetlinux.org/

[ONOS] "ONOS" [Online], https://opennetworking.org/onos/, https://github.com/opennetworkinglab/onos

[ONOSREST] "ONOS REST APIs" [Online] https://wiki.onosproject.org/display/ONOS/Appendix+B%3A+REST+API

[OpenConfig] OpenConfig, "OpenConfig web site," [Online]. Available: http://www.openconfig.net. GitHub: https://github.com/openconfig.

[OpenROADM] OpenROADM [Online], http://openroadm.org/

[P4] “P4” [Online]. Retrieved October 5, 2022, https://p4.org/

[Pav15} P. Pavon-Marino and J. L. Izquierdo-Zaragoza, “Net2plan: An open source network planning tool for bridging the gap between academia and industry,” IEEE Netw, vol. 29, no. 5, pp. 90–96, Sep. 2015, doi: 10.1109/MNET.2015.7293311.

[RFC8345] A. Clemm et al, "A YANG Data Model for Network topologies", https://datatracker.ietf.org/doc/html/rfc8345

[SONIC] "SONiC" [Online] https://sonic-net.github.io/SONiC/

[TAPI2.1.3] Transport API 2.1.3 [Online], Retrieved 15 december at https://github.com/OpenNetworkingFoundation/TAPI/releases/tag/v2.1.3

[TR-547] TAPI Reference Implementation Agreement TR-547 [Online] https://opennetworking.org/wp-content/uploads/2021/12/TR-547-TAPI_ReferenceImplementationAgreement_v1.1.pdf