METRO-HAUL

METRO High bandwidth, 5G Application-aware optical network, with edge storage, compute and low latency

Grant No. 761727

Deliverable D4.2

Report on Implementations for METRO-HAUL Control for Resource and Service Management.

Editor:	Luis Velasco, UPC
Deliverable nature:	Report
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	2019-04-30
Actual delivery date:	2019-06-18
Suggested readers:	Optical Networks Researchers and Engineers; Product Managers; Architects and Technology specialists.
Version:	1.0
Total number of pages:	107
Keywords:	SDN/NFV; Network Control Plane; Multilayer and Optical Transport Services; Management and Orchestration (MANO); Disaggregated Optical Networks; Monitoring and Data Analytics; Dynamic Network Planning; Open Source

Abstract

This document (D4.2) reports the final architecture of the METRO-HAUL control, orchestration, and management (COM) system from its initial design in D4.1, after successive refinements driven by feedback gathered after implementation and integration activities, and reflecting additional choices and newly added components.

In addition, the document reports the status and assessment of the final implementation of METRO-HAUL functional components, including control plane validation of relevant performance indicators and tests. The description of each functional component in the COM includes the list of subcomponents and their status. The functional tests carried out to validate and integrate the subcomponents are described together with the testbeds used. The definition of the specific key performance indicators (KPIs) in which the component is involved are defined and measured. The interfaces connecting functional components of the COM are defined in terms of functions with regard to requirements in D4.1 and new functions. Details of the implementation of every function are reported.

The research leading to these results has received funding from the European Union Horizon 2020 Programme under grant agreement number METRO-HAUL 761727

Impressum

[Full project title] METRO High bandwidth, 5G Application-aware optical network, with edge storage, compute and low latency

[Short project title] METRO-HAUL

[Number and title of work-package] WP4

[Number and title of task] T4.1, T4.2, T4.3

[Document title] Report on Implementations for METRO-HAUL Control for Resource and Service Management.

[Editor: Name, company] Luis VELASCO, UPC

[Work-package leader: Name, company] Ramon CASELLAS, CTTC

Executive Summary

The infrastructure requires a complex Control, Orchestration, and Management (COM) system, which includes several subsystems and interfaces among them. The COM system relies on recent advances in Software Defined Networking (SDN) and Network Function Virtualization (NFV), trying to adapt existing frameworks to the specifics of the project, that is, the applicability to Metropolitan networks, the deployment of disaggregated optical networks, the importance of monitoring, telemetry and data analytics and the interest of externalizing the algorithmic aspects (network optimization, function placement, resource allocation) to dedicated subsystems.

The architecture of the METRO-HAUL COM system has evolved from its initial design reported in D4.1 and successive refinements have been made driven by feedback gathered after implementation and integration activities. This document first reports the final architecture of the COM system and includes additional choices and newly added components. Specifically, the main components include:

1. The NFV **Orchestrator** (NFVO) that performs Service Orchestration and Resource Orchestration to support the Virtual Network Functions (VNFs) and the logical links. In the context of METRO-HAUL, a **network slice** consists of a Network Service (NS) deployed using the NFVO spanning multiple nodes and network domains. The VNF placement functionalities are provided by the Back-End Module of the Network Planner.
2. The **WAN infrastructure Manager** (WIM) responsible for provisioning of connectivity paths between VNFs. The WIM architecture is hierarchical, with an SDN control per technology domain and a parent SDN controller abstracting the underlying complexity.
    a. At the optical transport layer, two approaches have been considered based on two main disaggregation models: a) **Partial Disaggregation**: Open Line System (OLS) and Multi-Vendor Transponders (TP), and b) **Full Disaggregation:** Wavelength Division Multiplexing (WDM) transport system.
    b. Regarding the **control of the Passive Optical Networks** (PON), it is realized through the adoption of an abstraction scheme which represents the PON as an OpenFlow/NETCONF-enabled switch.
3. The **Monitoring and Data Analytics** (MDA), responsible for implementing autonomic networking. The **MDA is distributed**, with MDA agents running close to the network nodes, and a big data centralized MDA controller running in the COM system.
4. The **Placement, Planning, and Reconfiguration Subsystem** (Network Planner), responsible for optimizing the resource allocation and applying different policies and strategies.
5. Two **SDN applications** for proactive soft-failure detection and de-fragmentation are included.

The document also reports the status and assessment of the final implementation of METRO-HAUL functional components, including control plane validation of relevant performance indicators and tests. The description of each functional component in the COM includes the list of subcomponents and their status. The functional tests carried out to validate and integrate the subcomponents are described together with the testbeds used. The specific key performance indicators (KPIs) in which the component is involved are defined and measured.

Finally, the interfaces connecting functional components of the COM are defined in terms of functions with regard to requirements in D4.1 and new functions. Details of the implementation of every function are reported.

Document structure

This document is structured as follows:

- **Section 1** is the introduction. We briefly present the main components of the control, orchestration and management system of the METRO-HAUL project.
- **Section 2** reports the final architecture of the METRO-HAUL COM system. The COM architecture includes: 1) the COM core platform that includes the hierarchical SDN system and the integration with compute and store; 2) the ETSI MANO/slicing integration; 3) the MDA subsystem; and 4) the Network Planner. To illustrate how external applications can perform specialized tasks and be integrated with the METRO-HAUL architecture, two external applications are included in this section.
- **Section 3** reports the status and assessment of the final implementation of the METRO-HAUL COM architecture functional components, including control plane validation of relevant performance indicators and the designed tests. The described components are: 1) the Parent Controller; 2) the Optical SDN Controller; 3) the OLS controller; 4) the SDN for Passive Optical Networks; 5) the service orchestrator and the integration with the parent controller; 6) the system for network virtualization and slicing; 7) the Monitoring and Data Analytics subsystem; and 8) the Network Planner. Additionally, this section includes the service and traffic monitoring that is composed by active and passive probes to monitor the system at packet layer. Finally, the report of the status of the two external applications is included in this section as well.
- **Section 4** is devoted to describing every interface defined between the components described in the previous section. The interfaces are described in terms of the functions that they support and how such functions have been implemented.
- **Section 5** concludes the deliverable.

List of Authors

	Name	Partner
Contributors:		
	Luis Velasco, Gabriel Junyent, Jaume Comellas, Marc Ruiz, Lluis Gifre	UPC
	Ramon Casellas, Ricardo Martínez, Ricard Vilalta, Raül Muñoz, Michela Svaluto	CTTC
	Francisco Javier Moreno, Miquel Garrich, Pablo Pavón	UPCT
	Jorge E. López de Vergara, Mario Ruiz, Luis Vaquero, José F. Zazo, Sergio López-Buedo	Naudit HPCN
	Alessio Giorgetti, Filippo Cugini	CNIT
	Annalisa Morea	NOKIA Italy
	Danish Rafique	ADVA
	Antonio D'Errico	Ericsson
	Abubakar Siddique Muqaddas	UNIVBRIS
	Oscar Gonzalez de Dios	TID
	Adrian Farrel	ODC
Checked by:		
	Luitgard Hauer	EURESCOM

Revision History

Revision	Date	Responsible	Comment
0.0	30/03/2019	Editor	Initial version, ToC
0.1	10/05/2019	Editor	First Integrated Version
0.2	04/06/2019	Editor	Second Integrated Version
0.3	05/06/2019	Editor	Third Integrated Version
0.4	06/06/2019	Editor	Fourth Integrated Version
0.5	09/06/2019	Editor	Fifth Version, editorials
0.6	10/06/2019	Editor	Sixth Version, editorials
0.7	13/06/2019	Editor	Seventh Version, editorials

Table of contents

List of Figures

List of Tables

1 Introduction

The METRO-HAUL infrastructure spans nodes residing in Central Offices (CO) in different geographic locations, where every node combines networking, processing, and storage resources. Such modular nodes are composed of different components operating at different layers and technologies, and of different vendors realizing hardware and software disaggregation. METRO-HAUL nodes implement layer 0-1 (optical domain) and layer 2 transmission and switching (frame domain), and include Edge Computing capabilities provided by a local pool of computers to instantiate Virtualized Network Functions (VNFs) with configurable amounts of processing, memory, and storage. Two specializations of the generic METRO-HAUL nodes are: a) Access Metro Edge nodes (AMEN) to interface with heterogeneous access technologies (5G and optical); and b) Metro Core Edge nodes (MCEN) nodes as gateways towards the core transport network and comprise core-oriented capabilities. The nodes are controlled by a Node Agent based on NETCONF/YANG handling the integration of such disaggregated components.

The infrastructure requires a complex Control, Orchestration, and Management (COM) system, which includes several subsystems and interfaces among them. Figure 1 summarizes the components of the COM system and the main interfaces.

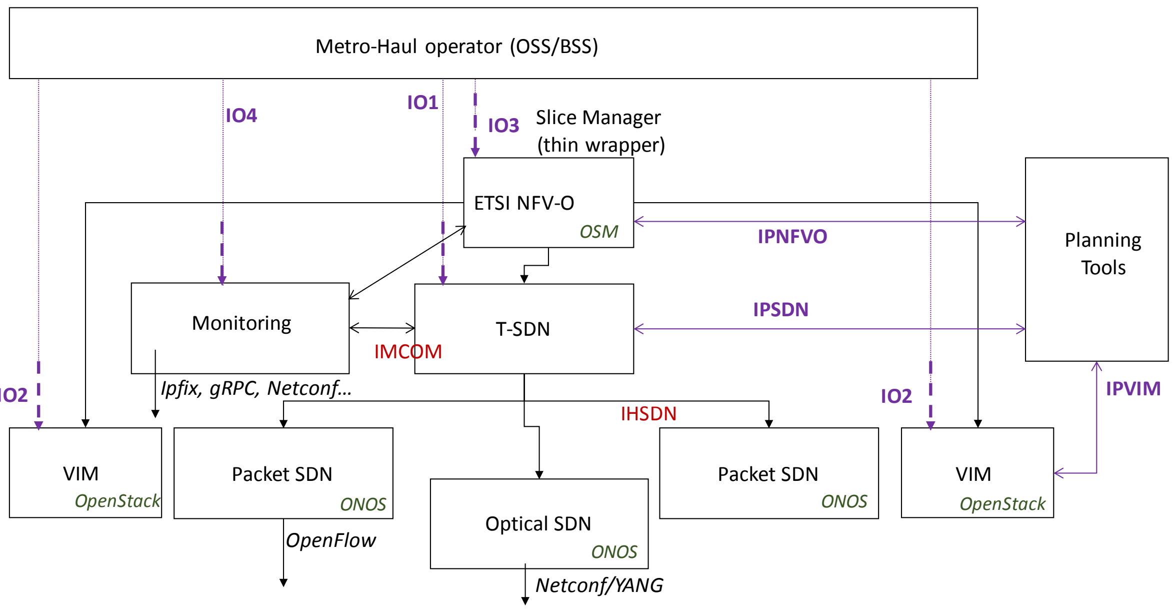


Figure 1. METRO-HAUL COM system and main interfaces

The main components include:

1. The **Network Function Virtualization** (NFV) **Orchestrator** (NFVO) that performs Service Orchestration (involving the functional split of the service into/amongst different VNFs and their logical interconnection) and Resource Orchestration dealing with the allocation of resources to support the VNFs and the logical links. In METRO-HAUL, the NFVO is used to provision Network Services (NS) that usually entail VNFs deployed in multiple; the VNF placement functionalities are provided by the back-end module of the Network Planner. The Virtual Infrastructure Manager (VIM) is the responsible for the management of the NFV

Infrastructure (NFVI) and the instantiation of the Virtual Machines (VM) of the VNFs in a single datacentre domain (AMEN/MCEN).

2. The **WAN Infrastructure Manager** (WIM) is used by the NFVO to orchestrate network resources and it is responsible for provisioning of connectivity paths between VNFs. The WIM architecture is hierarchical and consists of a Software Defined Networking (SDN) controller for every technology domain. Two approaches have been considered for the control of the optical layer based on two main disaggregation models: the partial disaggregation that includes the Open Line System (OLS) and Multi-Vendor Transponders (TP) and the Fully disaggregated WDM transport system. The control of **Passive the Passive Optical Networks (PON)** abstracts the PON as an OpenFlow/NETCONF-enabled switch, where the physical ports the PON are represented as the logical ports of a switch. Running on top of the SDH hierarchy, the parent SDN controller abstracts the underlying complexity and presents virtualized networks to their customers.
3. The **Monitoring and Data Analytics** (MDA), responsible for implementing autonomic networking. The monitoring system has the capability to do measurements on the data plane and for generating data records that are collected and analysed by the MDA subsystem to discover patterns (knowledge) from the data. In METRO-HAUL, the MDA is distributed and consists of MDA agents that run in the network nodes and are responsible for monitoring data collection, aggregation and knowledge usage. Aggregated monitoring data is conveyed to the MDA controller where knowledge is discovered. Such knowledge can be used to issue re-configuration/re-optimization recommendations towards COM modules such as an SDN controller or orchestrator.
4. The **Network Planner**, responsible for optimizing the resource allocation in the optical metro network to effectively provision services featured by heterogeneous requirements. This task comprises the provisioning of VNF in specific METRO-HAUL nodes and the allocation of network resources.

In the context of METRO-HAUL, network slices consist of a Network Service (NS) deployed using the NFVO controlling VIM/WIM spanning multiple nodes and network domains.

The COM system relies on recent advances in on SDN and NFV. Specifically:

1) Open Source MANO (OSM) is used as an NFVO [OSM]. OSM is an open source MANO system which is based on ETSI NFV Information models.
2) Open Network Operating System (ONOS) is used as SDN controller.
3) OpenStack is used as the VIM.

Based on those existing frameworks, extensive adaptation work to the specific requirements of the project has been carried out. This includes the applicability to Metropolitan networks, the deployment of disaggregated optical networks, the importance of monitoring, telemetry and data analytics, and the interest in externalizing the algorithmic aspects (network optimization, function placement, resource allocation) to dedicated subsystems.

2 Final architecture of the control and management plane

This section reports the final architecture of the control and management plane after successive refinements driven by feedback gathered after implementation and integration activities, and reflects additional choices and newly added components.

2.1 METRO-HAUL COM Core Platform

2.1.1 Hierarchical SDN system

2.1.1.1 Parent controller and service orchestrator

As considered in [D4.1] METRO-HAUL is aligned with the IETF ACTN architecture [ACTN] for network orchestration as shown in Figure 2. ACTN can facilitate virtual network operation via the creation of a single virtualized network or a seamless service. This supports operators in viewing and controlling different domains (at any dimension: applied technology, administrative zones, or vendor- specific technology islands) and presenting virtualized networks to their customers. ONF TAPI models have been implemented as the northbound interface (NBI) for the SDN controller. Virtual network aspects will be reported in next deliverable D4.3.

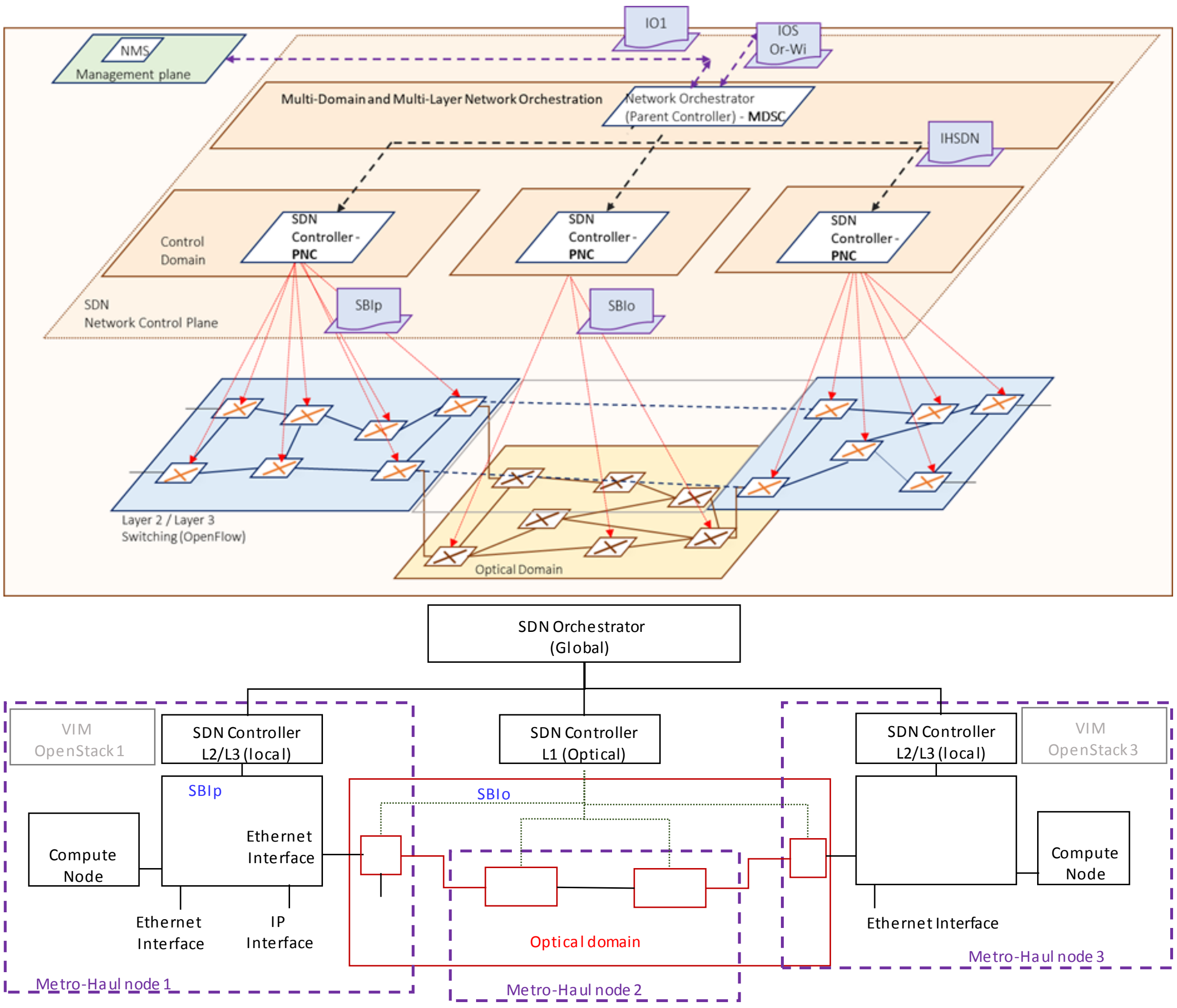


Figure 2. Network Orchestration Architecture.

2.1.1.2 Control of the Optical Layer

In the project, we have considered two approaches based on two main disaggregation models.

Partial Disaggregation: Open Line System (OLS) and Multi-Vendor TP. In this approach (Figure 3), the disaggregation applies to the Digital-to-WDM (DtoWDM) adaption layer (i.e., to the TPs) whose lifecycle is decoupled from that of a mono-vendor and proprietary Analogue WDM (A-WDM) transport layer [Ric18]. The A-WDM layer remains a proprietary black box analogue transport system supporting Optical Channels from external TPs as client signals. An OLS-NBI API is needed to configure and report events from the OLS. Note in Figure 3.A) an Open Line System is part of a partial disaggregated WDM transport system: the OLS and controller are from a single vendor (1-2); TPs may be in pairs from the same supplier (3), or from mixed suppliers (4); the WDM Transport Controller interfaces directly with the TPs (5), and through an NBI (7) to the OLS. The Single Wavelength Interface (SWI) needs to be standardized (6). Figure 3.B) shows an alternative partial disaggregated WDM transport system: OLS and WDM controller are proprietary from a single vendor (1-2); TPs may be in pairs from the same supplier (3) or from mixed suppliers (4); the proprietary WDM Transport Controller interfaces directly with TPs using a standard SBI (5). The Single Wavelength Interface (SWI) (6) and SBI (5) need to be standardized.

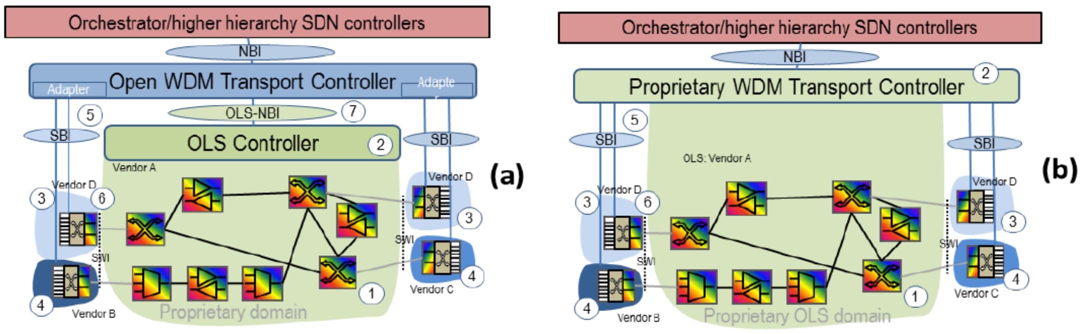


Figure 3. A) OLS as part of a partial disaggregated WDM transport system. B) Alternative partial disaggregated WDM transport system.

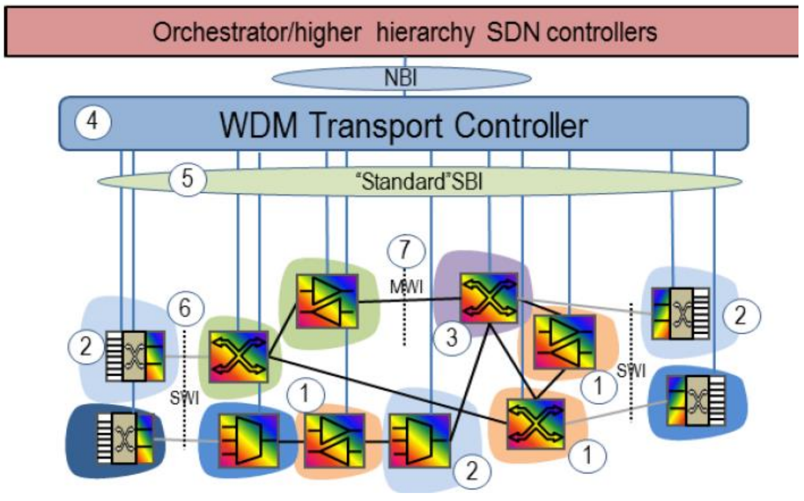


Figure 4. Fully disaggregated WDM transport system.

The **Fully disaggregated WDM transport system** is shown in Figure 4 [Ric18]. Optical Network Elements (O-NEs) from both the A-WDM and DtoWDM layers are potentially purchased from different vendors, leaving interworking at the control and data plane to the system integrator. Therefore, most of the control intelligence is moved to the WDM controller (necessarily vendor agnostic) which becomes the most critical element of the whole chain, having also to face all of the analogue transmission issues (equalization, transient suppression, etc.). In the figure, a fully disaggregated WDM transport system is shown: O-NEs can be from the same (1-2) or from different suppliers. No separation between DtoWDM and A-WDM layers exists. A standard SBI (5) is needed to simplify the direct control of the whole WDM System by the controller (4). Both Single Wavelength (6) and Multi Wavelength Interfaces (7) need standardization.

Regardless of the actual deployments, we can consider the SDN control of the optical layer as a single component of the COM system, provided the right abstractions and interfaces are defined. In particular, it is useful to consider that the METRO-HAUL architecture for the control of the disaggregated Optical Network relies on a centralized SDN controller and the use of YANG/NETCONF as data modelling language and protocol, without excluding deployments combining the aforementioned choices.

The SDN Controller is responsible for the centralized control of the optical infrastructure; exports a North Bound Interface to applications, to instantiate services. The services are described using YANG models; optical devices (Optical Network Elements) are configured using a NETCONF/YANG protocol and data models; and network connectivity is provisioned in the optical layer between transponders (with optional recovery mechanisms).

METRO-HAUL is aligned with three relevant worldwide open initiatives: OpenROADM [OpenROADM], OpenConfig [OpenConfig], and ODTN [ODTN]. METRO-HAUL considers OpenROADM and OpenConfig to model the hardware devices. The baseline architecture applies a single SDN controller for the whole optical domain, without precluding refinements of the architecture including a deployment model with multiple co-ordinated controllers, e.g., for scalability reasons.

2.1.1.3 Control of the PON

Control of the PONs is realized through the adoption of an abstraction scheme which represents the PON as an OpenFlow/NETCONF-enabled switch. Under this abstraction, the physical ports the PON interconnect to the network infrastructure, i.e. the upstream port of the Optical Line Terminal (OLT) and the downstream ports of the Optical Network Units (ONU), are portrayed as the logical ports of a switch.

The proposed abstraction scheme offers two advantages: a) it hides the PON-specific details related to forwarding and control/management operations; b) vendor-specific configuration commands are automatically translated and executed by the PON components. Therefore, the forwarding command messages are exchanged by means of the standard OpenFlow protocol, while (re)configuration is realized by means of the NETCONF protocol.

Under this scheme, the PON is operated as an OpenFlow-controlled L2 switch. The novelty of this approach is that it is a truly plug-and-play solution in the sense that standard protocols are used; there is no need for OpenFlow protocol extensions or upgrades to the already-deployed SDN control/management tools. Operators may simply connect the OpenFlow-enabled PON infrastructure into their network and integrate the provided YANG model into the NETCONF infrastructure. Moreover, this approach allows the operator to capitalize on the increased penetration of OpenFlow/NETCONF in other network segments (WAN/Metro).

Further, this scheme is extended to support PON slicing since several OpenFlow-controlled (virtual) switches can be abstracted over the same physical Gigabyte Passive Optical Network (GPON). Under this mode, each of these switches can dynamically control a subset of the GPON resources. Thus, a PON is partitioned into several logical PONs where each one consists of a logical-OLT (having a slice of the physical OLT) and from several logical-ONUs that may include any combination of ONUs, ONU ports, or even ONU port slices.

2.1.2 Compute and Storage Integration

METRO-HAUL follows the ETSI NFV MANO [NFV] architecture, which can be defined as an architecture and deployment model around the idea of replacing dedicated network appliances — such as routers and firewalls — with software implementations (guests) running on common shared hardware (hosts), becoming Virtualized Network Functions (VNFs). The benefits have been well established, including lower costs, replacing dedicated appliances with shared servers, use of capacity on demand with efficient resource usage, reduction of operational costs with fewer appliances to deploy and maintain, enabling migrations), support of on-demand and pay-as-you-go deployment models, and enabling innovation by making it easier to develop and deploy network functions.

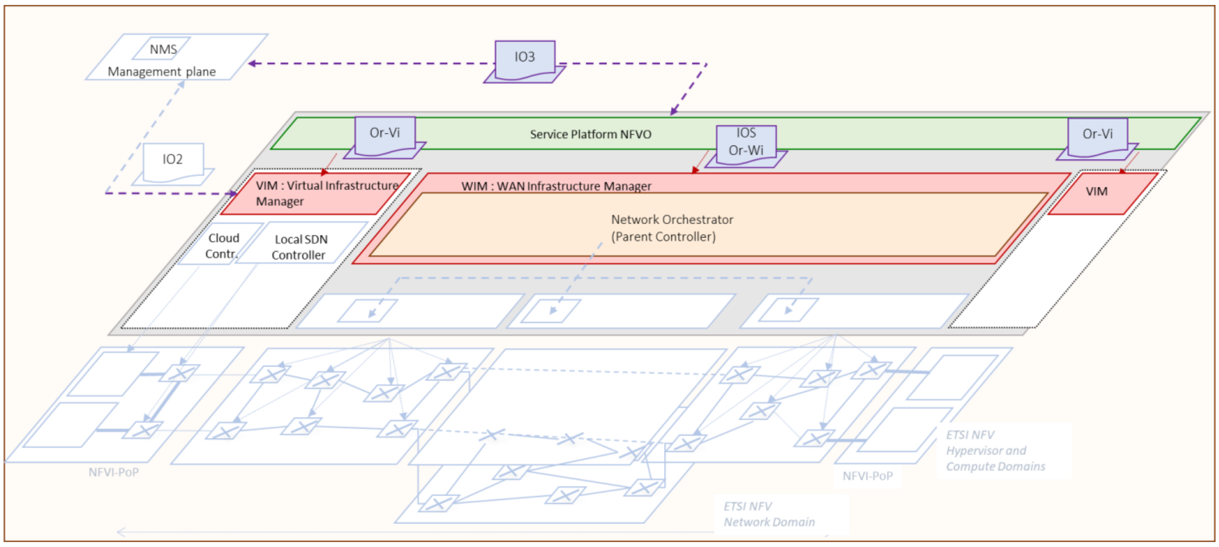


Figure 5. Integration with Computing and Storage, ETSI/NFV MANO

The ETSI NFV architecture defines the NFVI deployed across multiple points of presence (NFVI-PoP) for supporting the instantiation of VMs, along with the Management and Orchestration (MANO) subsystem, which deals with the orchestration of VNFs and how to deploy them as components of the so-called Network Services.

The following section details the METRO-HAUL implementation of the elements that are key to the architecture (Figure 5): the NFVO, the WIM, and the VIM. The NFVO performs Service Orchestration and Resource Orchestration. Service Orchestration is the part of service instantiation that involves the functional split of the service into/amongst different VNFs – that may be managed by different managers (VNFMs) by different vendors – and their logical interconnection (called VNF forwarding graphs). Resource Orchestration deals with the allocation of resources to support the VNFs and the logical links. The WIM is responsible for the provisioning of connectivity paths between VNFs. The VIM is responsible for the management of the NFVI and the instantiation of the VMs of the VNFs in a single datacentre domain (AMEN/MCEN). VIM components are used “as is” in the scope of METRO-HAUL, not being the subject of research.

2.2 ETSI MANO / Slicing Integration

In METRO-HAUL, an ETSI-based Management and Network Orchestration (MANO) system, which is essentially an NFV orchestrator (NFVO), is used to deploy Network Services (NS) realizing the concept of NFV. Each NS is a combination of one or more Virtual Network Functions (VNFs) interconnected by logical links. In the ETSI NFV world, each network slice instance is an NS encompassing multiple physical locations, where each VNF of the NS may be hosted in different PoPs/datacentres and the links interconnecting the VNFs may span multiple network segments. Furthermore, the NFVO is responsible for the overall end-to-end deployment of the Network Slice. This is depicted in Figure 6. As noted in D4.1:

Within the scope of METRO-HAUL, the main focus of a network slice is the ETSI NFV Network Services, that is, a set of interconnected VNFs, across the METRO-HAUL infrastructure (see, for example, Figure 6 and Figure 7).

An NFVO orchestrates compute resources using Virtual Infrastructure Managers (VIMs), where the NFVO deploys VNFs using the VIM. The VIM in turn, deploys the VNFs in Virtual Machines (VMs) in the compute nodes under its control. In addition, to deploy the network between the VNFs, the NFVO utilizes a WAN Infrastructure Manager (WIM). The WIM may use a combination of SDN controllers to deploy the network over various packet and optical domains as shown in Figure 6 interconnecting the PoPs where the compute resources are hosted. In D4.1, architectural considerations were presented regarding multiple NFVO instances. However, in METRO-HAUL, we utilize a single NFVO instance to orchestrate network slices.

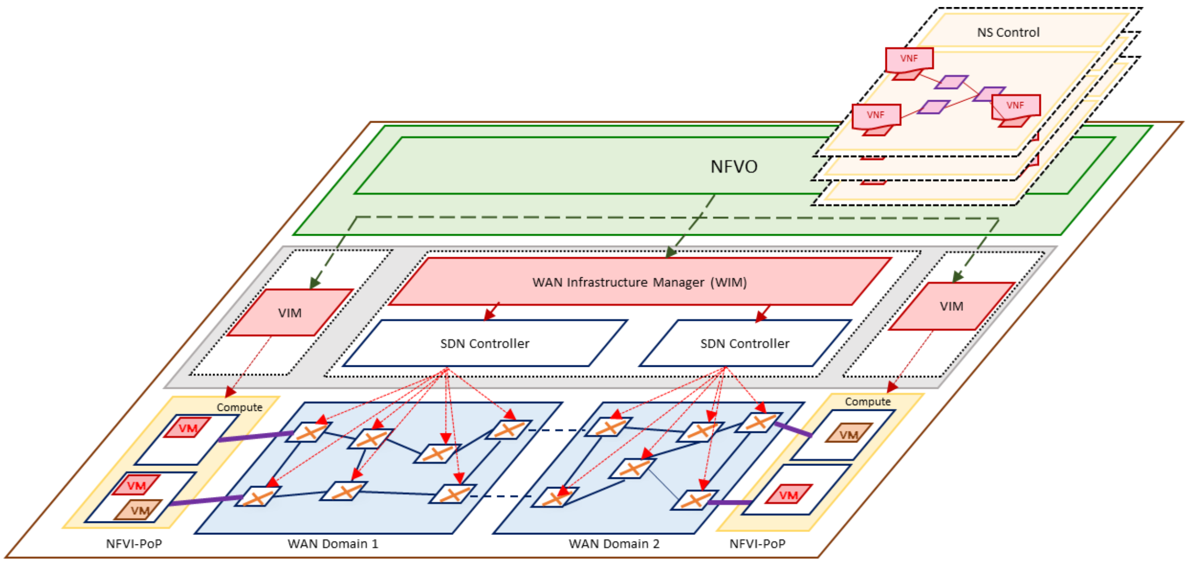


Figure 6: Network Slicing using the integrated SDN/NFV framework.

Figure 7 further elaborates on the Network Service (NS) as a METRO-HAUL slice where the VNFs are deployed as VMs in NFVI-PoPs and the links interconnecting the VNFs are spanning multiple packet and optical domains. In this way, an NS is provisioned as a slice of the overall NFVI.

Within the scope of the METRO-HAUL project, Open Source MANO (OSM) is used as an NFVO [OSM]. As an operator-led community, OSM offers a production-quality open source MANO stack that meets the requirements of commercial NFV networks. To enable connectivity between VNFs at different PoPs, OSM has been extended, as part of release 5, to have the capability to orchestratea network between different PoPs using the WIM.

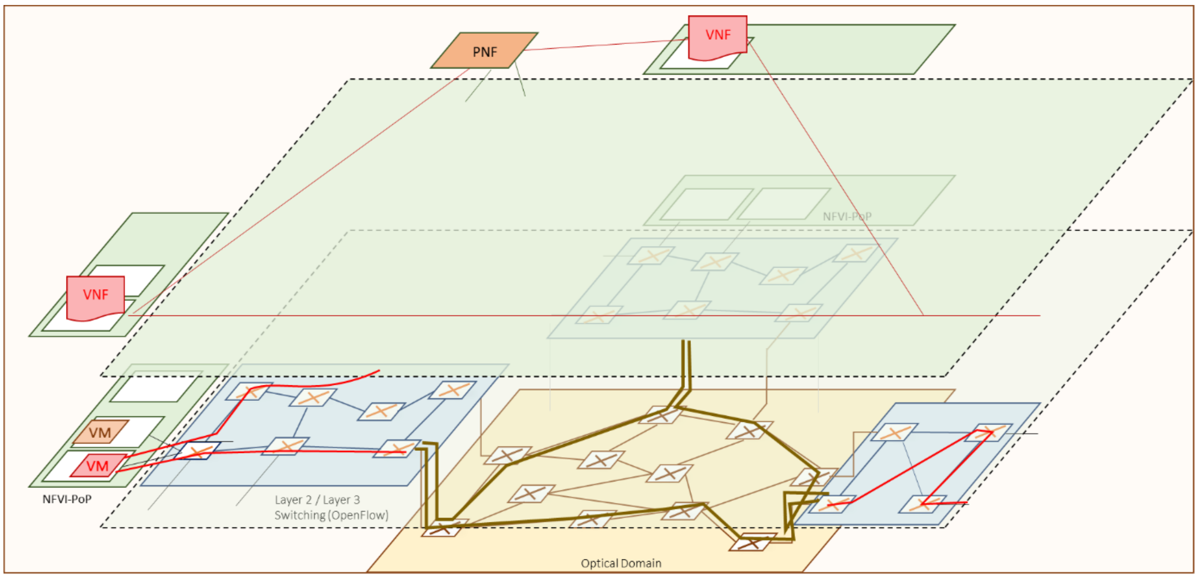


Figure 7. ETSI NFV Network Service as canonical METRO-HAUL 5G slice

Furthermore, a planning tool performs the VNF placement by selecting the location of the VNFs of an NS to be in multiple AMENs and MCENs while instantiating a NS. More details about this can be found in Section 2.4.

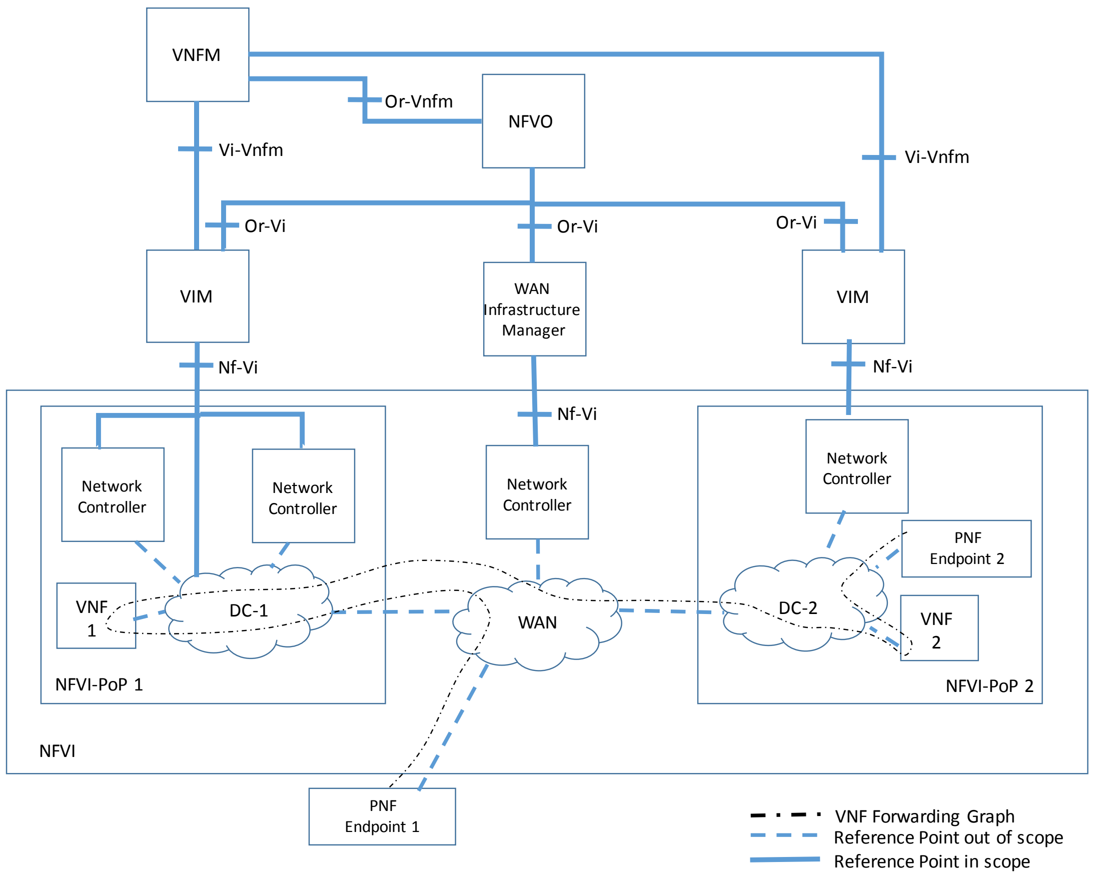


Figure 8.WIM in ETSI MANO architecture

2.2.1 WAN Infrastructure Manager (WIM)

In the ETSI NFV architecture, the WIM is used by the NFVO to orchestrate network resources, interconnecting the multiple PoPs as shown in Figure 6 and Figure 8. Different VNFs, which are running on compute resources at geographically distributed PoPs as part of an NS, are interconnected by the network which is provisioned by the WIM. As mentioned before, the WIM uses multiple SDN controllers to control the network interconnecting the PoPs as the network may be segmented into multiple SDN domains.

The overlay network interconnecting the VNFs (as part of an NS) on different AMENs/MCENs is provisioned by the WIM. OSM has been extended to use a WIM.

2.2.2 Placement options

The placement functionalities are provided by the Back-End Module of the Network Planner. In the previous deliverable [D4.1], two preliminary algorithms were proposed to address the planning and provisioning algorithms: VNF Placement and Scaling Optimizer (NPSO), and Network Resource Allocation Optimizer (NRAO). These approaches have evolved to widen the scope for placement options. Algorithms hosted in the Back-End module include:

a. A network service /service chain allocation algorithm
b. VNF placement algorithm
c. Network Resource and Wavelength Allocation (RWA) algorithm.

Algorithm a) is in an advanced stage of development and already included in the Net2Plan framework; algorithms b) and c) are currently under development and integration. All these algorithms adopt network optimization techniques, are oriented to connection and service-chain real-time provisioning, and exchange information with the Front-end using the NIW library integrated in Net2Plan. They are implemented in Java. The second module incorporates algorithms oriented to predictive periodical or off-line network re-optimization, they make use of machine learning (ML) techniques and are developed in Python.

Therefore, the user has several degrees of freedom in choosing the algorithms that will actually be used for a specific demo or a specific application. Moreover, it will be possible to compare the results of different algorithms on the same problem so as to facilitate the evolution of the computation methods. For instance, we will be able to compare the single-step approach (integrated VNF placement + RWA) to the simpler two-steps (first VNF placement, then RWA).

We consider four attributes of the implemented algorithms, namely: VNF Reactive, Network Reactive, VNF Proactive, and Network Proactive (see Figure 12). A given algorithm, e.g., #4, performs functions associated with VNF and Network planning in a pro-active way. The algorithms provided by the partners of the project can be split into six different functionalities, according to the type of optimization they implement, i.e., they compute the configuration either in reactive or proactive modes in terms of VNFs, network resources, or both.

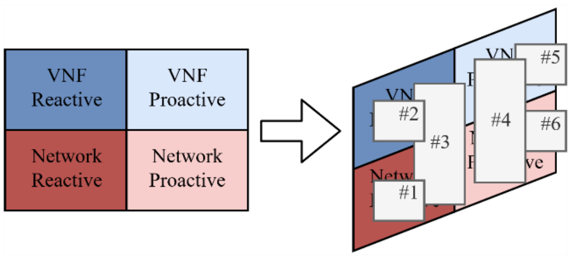


Figure 9: Placement, planning, and reconfiguration algorithmic view.

2.3 Monitoring and Data Analytics Subsystem

2.3.1 Architecture

The overall architecture of the monitoring and data analytics (MDA) subsystem together with the related interfaces is presented in Figure 10. As shown, the MDA includes *MDA agents* running close to the network nodes, and a big data centralized *MDA controller* running in the control and management plane.

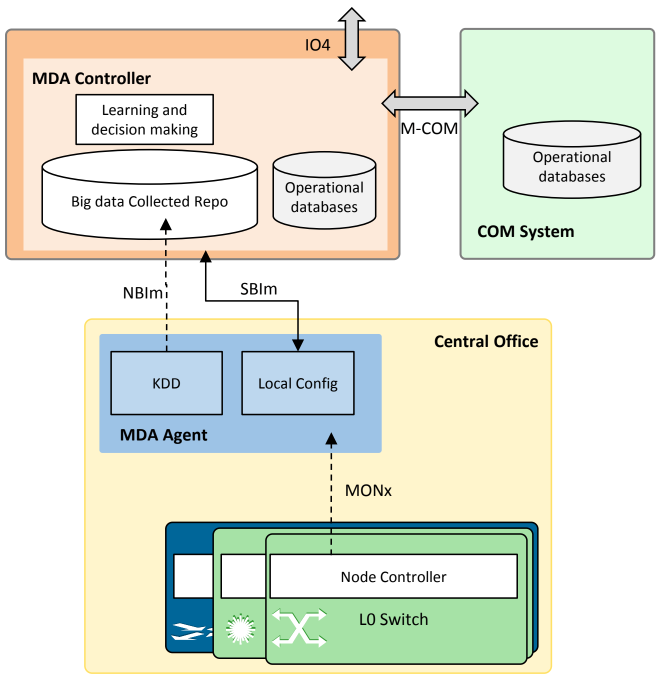


Figure 10. Overall METRO-HAUL monitoring architecture for autonomic networking

MDA agents are *multi-node* agents that collect monitoring data records from configured OP in the nodes using the monitoring of packet devices (MONp) and monitoring for optical devices (MONo) interfaces (collectively denoted as *MONx*). Data can be used for Knowledge Discovery from Data (KDD) to proactively implement local control loops to tune parameters in the network devices and to notify the MDA controller about network anomalies and degradations. The MDA agent has been conceived to support multilayer disaggregated scenarios. The number of MDA agents in a network may vary depending on the size of such network, geographical extension, and/or any other criteria.

The MDA controller collates measurements from the active Observation Points (OP) in the network through the Northbound monitoring (*NBIm)* interface and stores them in a (big data) repository. In addition, the Southbound monitoring (*SBIm)* interface is used for monitoring configuration. In addition, an NBI interface, named *IO4*, is provided for external systems to access monitoring data.

The protocol used to implement the SBIm interface for monitoring configuration is RESTCONF [RFC8040] that it is based on an extended YANG data model, whereas the IPFIX protocol [RFC6313], [RFC7011], has been implemented for the NBIm interface for monitoring purposes and extended by defining new templates, specifically to convey measurements from the optical data plane. However, different protocols for monitoring and telemetry might be considered (e.g., gRPC).

Data analytics algorithms in the MDA controller can use measurements as well as notifications from the MDA agents to discover knowledge. Such knowledge can be used to make predictions and to detect anomalies and degradations before they negatively impact on network performance. Such predicted events can be notified to the SDN controller in advance together with a *recommended action* to guide the SDN controller. The recommended action is a suggestion that the SDN controller can follow or just ignore and apply its own policies. As an example, BER degradation can be anticipated in a lightpath before any threshold is exceeded by analysing the evolution of the BER measured in the receiver: this is notified to the SDN controller together with the recommended action of re-routing the lightpath. The notification to the SDN controller might trigger a network re-configuration, hence closing the loop and adapting the network to the new conditions. In consequence, the MDA Controller to COM System interface (the *M-COM interface)*, is required to coordinate the SDN controller and the MDA controller, as well as for synchronization of operational data bases.

2.3.2 Workflows

Figure 11 presents two basic workflows: *i*) *Label Switched Path (LSP) set-up and monitoring / telemetry activation*; and *ii*) *degradation detection and network reconfiguration*. These basic workflows are used for the functional tests to validate the MDA subsystem in the Section 3.7. Each message exchanged is identified with a number that is relative to the workflow it belongs to: note that numbers are reset in every workflow.

The LSP set-up and monitoring/telemetry activation workflow is triggered by a request to the SDN controller (message 0). After computing the specific routing algorithm, the SDN controller sends configuration messages (1) to the node controllers along the route of the LSP specifying a set of parameters, e.g., the spectrum allocation in the case of a Layer 0 LSP. When the LSP has been finally set up, a notification is sent to the MDA controller (3) through the M-COM interface containing the route of the LSP, configuration parameters and identifier.

Next, the MDA controller sends requests to the SDN controller for the creation and activation of a subset of OPs for the LSP (4). Those requests are processed in the SDN controller and subsequent requests are sent to the node controllers, where the specific IP address of the MDA agent that will process the data samples is included (5). Once created, the MDA controller issues requests to the MDA agents to create the OPs, and local KDDs create the specific OP handlers to collect, analyse, and aggregate measurements (6).

Once OPs are activated, monitoring/telemetry data is received and can be analysed to discover patterns. The KDD process workflow starts when new measurements are received in the MDA agent (message 1); monitoring data records are analysed by the corresponding process in a KDD application and, when a certain pattern is detected, e.g., a degradation, the MDA agent notifies the MDA controller (2). With such notifications, the MDA controller can make decisions, including suggesting a reconfiguration to the SDN controller (3), which can be carried out afterwards (4).

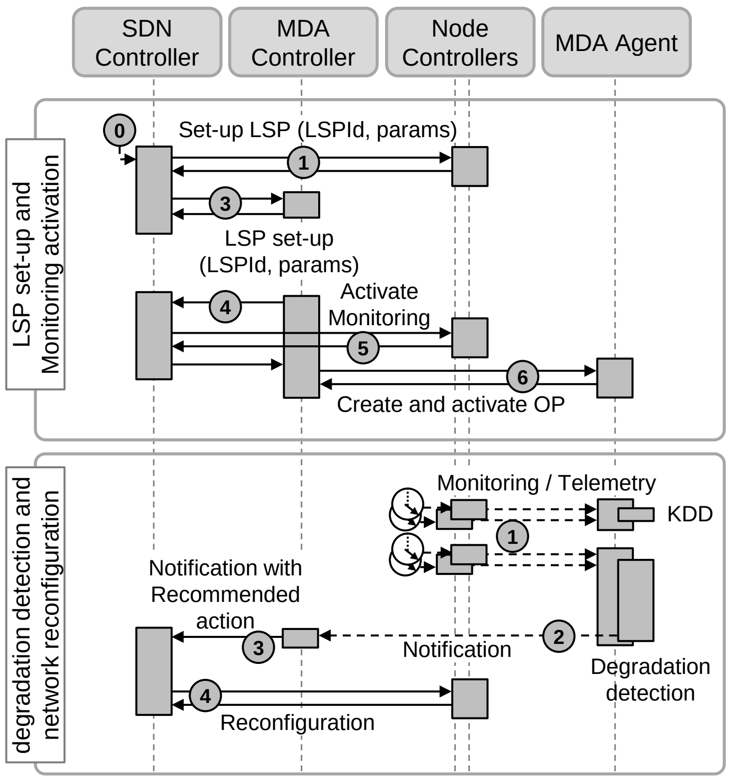


Figure 11. Basic workflows supported by the MDA subsystem

2.4 Placement, Planning, and Reconfiguration Subsystem

The aim of the Placement, Planning, and Reconfiguration Subsystem (alias, in short, Network Planner) is to optimize the resource allocation in the optical metro network in order to effectively provision services featured by heterogeneous requirements. This task comprises the provisioning of Virtual Network Functions in specific computing nodes (e.g., the mini datacentre distributed in the metro network) and network resources allocation.

The traffic generated by end-user-oriented services (such as Smart Factories, Live TV distribution, Content Delivery Network, etc.) have different requirements in terms of bandwidth, delay and QoS. For this reason, the intelligent optimization algorithms of the optimization module will apply different policies and strategies for the aforementioned services in order to optimize VNFs and network resources in advance, reacting to specific events. For all the cases, however, the optimizer will need data coming from the data plane (optical layer, layer 2, and IP layer) through the front-end interfaces, to learn the state of the network. Figure 12 shows the METRO-HAUL Service Platform, where the dashed red circle highlights the Placement, Planning, and Reconfiguration Subsystem. The Network Planner subsystem is composed of two main modules: the Front-end and Back-end modules. They are explained further in Section 3.8 including how they interact to implement planning and to provide solutions to the METRO-HAUL control plane.

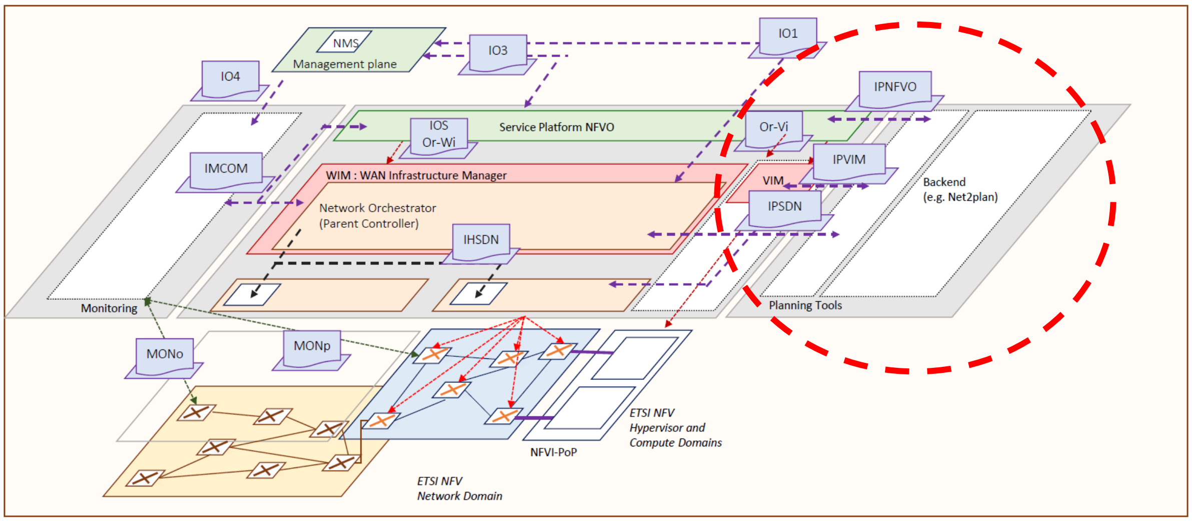


Figure 12. METRO-HAUL unified service platform. Planning tool module.

The open-source Net2Plan tool has been chosen as the common framework for materializing the placement, planning, and reconfiguration of the VNFs, IT and network resources. The specialized aforementioned algorithms will be used or developed in order to optimize the network and IT resources.

2.5 Applications

2.5.1 SDN application for proactive soft-failure detection

As stated in [D4.1], network management requires the need for improved asset management to reduce downtime and improve resource usage. Network faults may include cooling unit failure, laser degradation, subsystem control unit failure, etc. Early detection of equipment failure states and consequent remedial actions can prevent network downtime and enable scheduled preventive maintenance. A lot of commercial equipment tolerates some errors until automatically tearing down the connection when system thresholds are exceeded. While a restoration procedure could be initiated to recover the affected traffic, it would be desirable to anticipate such events and taking relevant actions.

The SDN Application (Figure 13) for proactive soft-failure detection is an analytics application enabling cognitive network assurance through proactive soft-failure detection. In particular, we cater to real-life network fault use cases and identify them.

As shown in Figure 13, from bottom to top, optical transport network configuration and telemetry information from the optical domain is extracted inside the Network Management System and made available to the ONOS controller through a RESTCONF interface. Telemetry information is provided northbound to the application for proactive soft-failure detection. The machine learning framework consists of several stages including data access through the REST API, pre-processing features, and neural networks for computational mapping of inputs to failure probability, and post-processing to trigger intents. The communication with the optical devices is done through ADVA's NMS, which exposes an experimental TAPI interface.

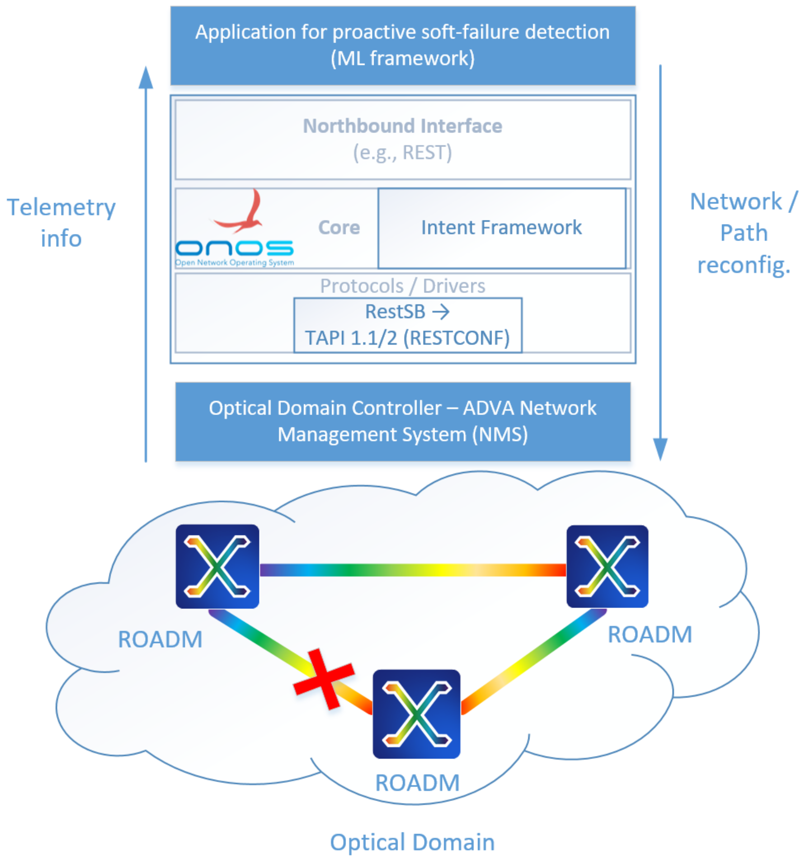

Application for proactive soft-failure detection (ML framework)
Northbound Interface (e.g., REST)
Core
Intent Framework
Protocols / Drivers
RestSB →
TAPI 1.1/2 (RESTCONF)
Optical Domain Controller – ADVA Network Management System (NMS)
Telemetry info
Network / Path reconfig.
ROADM
ROADM
ROADM
Optical Domain

Figure 13: SDN application for proactive soft-failure detection.

2.5.2 De-fragmentation application

The de-fragmentation application operates at the optical layer directly interacting with the Optical SDN controller. Specifically, the application is composed of two main components: the de-fragmentation tool (developed by Nokia as a software tool operating externally to the controller) and an extended set of REST APIs (developed as an internal ONOS application) that extends the features of the Optical SDN controller (Figure 14). The extended APIs have been merged in the official ONOS distribution in the form of an ONOS application with name "optical-rest".

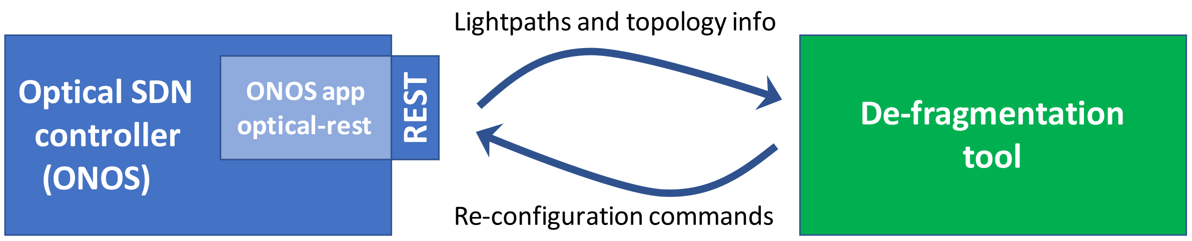

Optical SDN controller (ONOS)
ONOS app optical-rest
REST
Lightpaths and topology info
Re-configuration commands
De-fragmentation tool

Figure 14: Main architecture of the De-fragmentation app.

This section describes the basic behaviour of the de-fragmentation tool and the extended features that have been introduced in ONOS to enable the interaction with the tool.

- **External de-fragmentation tool**: the tool includes a web interface that takes inputs from the user (e.g., maximum fragmentation threshold, maximum reconfiguration frequency, etc.). Specifically, the tool proceeds as follow: the tool periodically retrieves the current network status (e.g., network topology, currently established lightpaths) from the ONOS controller and estimates the current fragmentation value, if the maximum fragmentation threshold is surpassed a defragmentation operation is triggered (i.e., optimization routine runs). If current lightpaths can be re-arranged to achieve better fragmentation status, the new configurations are sent from the tool to the ONOS controller (i.e., set up lightpaths on a new optical channel, OCh, and tear down the previous OCh configurations). The optimization routine is a component (developed in C++) that estimates the possibility of rearranging the OCh in the network so as to reduce the overall OMS/network fragmentation.

- **Extended ONOS controller REST APIs** (optical-rest ONOS app): the REST APIs of the ONOS controller have been extended to enable the interaction with the de-fragmentation tool. Specifically, new APIs have been developed to allow GET/PUSH/DELETE operations on lightpaths (i.e., optical intents). The GET operation provides the list of all established lightpaths specifying the path and the utilized OCh, the PUSH operation enables establishment of a new lightpath specifying both a suggested path and a suggested OCh, and the DELETE operation allows to tear-down a specific lightpath.

3 Final implementation of the METRO-HAUL COM functional components

This section reports the status and assessment of the final implementation of the METRO-HAUL control and management architecture functional components, including control plane validation of relevant performance indicators and the designed tests.

3.1 Parent Controller

3.1.1 Description

The Parent Controller is the component responsible for managing both the optical and packet controllers, providing the multi-layer view of the METRO-HAUL Network. The parent controller receives VPN service requests from the VNF orchestrator (OSM). The North-Bound interface is based on RESTCONF, using the YANG Data Model for Layer 2 Virtual Private Network (L2VPN) Service Delivery defined in RFC 8466, also known as L2SM [RFC8466]. The implementation covers the creation of L2VPNs with point-to-point topologies, supporting both full-port and dot1q-vlan-tagged interfaces, and bandwidth/delay constraints. Multiple L2VPNs can be requested over the same physical port (bearer). The L2SM YANG model is implemented via a set of REST operations, with read and write capabilities.

In terms of southbound interfaces, the parent controller is connected to the Optical Controller using the TransporTAPI version 2.1, using the topology model to retrieve information about the connectivity of the optical layer, and the service model to request new optical connections between transponder ports. The parent controller is also connected to packet controllers. In addition, a packet network emulator based on Open Virtual Switches (OVS) has been developed to be able to test with OSM.

3.1.2 Sub-Components

L2SM Server: The L2SM RESTCONF server is the main sub-component of the Parent Controller. It implements the logic of handling L2VPN service requests (create, read, modify and delete L2VPN Services). A set of REST Calls are implemented to fulfil the request of the OSM WIM Plugin.

It has been generated using Swagger codegen with an Open-API specification of the LS2M YANG model. The OpenAPI specification defines the set of calls that need to be implemented by the REST server. The mapping of YANG to OpenAPI is performed using a dedicated plugin. All the operations made by the server are reflected in a L2 Service database.

Topology Module: The parent controller maintains a multi-layer topology which contains:

- Packet topology (read from packet controller)
- Optical topology (read from ONOS)
- Multilayer topology
    - Packet links include the uuid of the services of the optical layer
    - Service end points include the identification needed by OSM Plugin: site_id and bearer

- Line end points (ports of the packet switches connecting to the optical networks) include the uuid of the transceiver client side (DSR) end point.

Database: In order to store the set of L2VPN Services that are created and the mapping to network resources, a MongoDB based Database us used. The database contains the list of available sites to request connection. It also stores the multi-layer view of the network (based on optical and packet controller's information), as well as the deployed L2VPNs. The server reads and writes the database when it is necessary to update the state of the components of the network. In order to facilitate the interaction with the server, the data is directly stored in JSON.

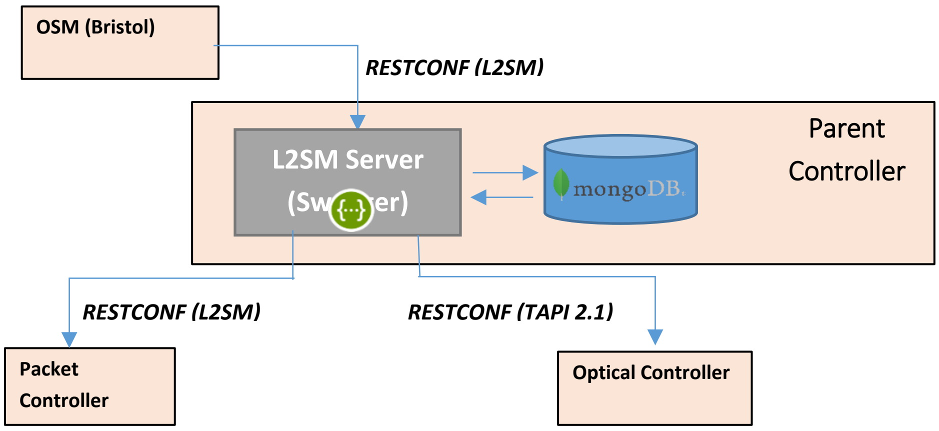


Figure 15. Parent Controller

3.1.3 Status

At the time of writing, a first version of the MongoDB database is fully operative. It is able to register all the operations carried out by the LS2M Server (Swagger based), saving the state of the network (all the sites and services that are available and their parameters).

The TAPI topology is retrieved from the ONOS controller and successfully correlated with the packet topology (which is currently loaded using the IETF Topology JSON model). The parent controller maintains the multilayer topology with the uuids used by OSM, its relation to the packet layer, and the correspondence between packet devices, interfaces, and optical connections.

On the other hand, the L2SM Swagger Server implements all the operations requested from OSM. Also, it implements the response to OSM with different status codes according to the result of the operation.

In order to test the packet layer, an emulator based on OVSs has been created. It enables to emulate a L2 VLAN based VPN to interconnect datacentres.

3.1.4 Related testbed and platform

In order to test the performance of the developed Parent Controller, some tests have been made:

- Manual tests performed on the server and database.
- Tests of the OSM Plugin.
- Tests of the TAPI of the Optical Controller

3.1.5 Functional Tests

Manual tests performed on the server: Some manual tests have been executed. These tests were developed to check the functions of the server and its connection with the database. These tests where developed using the command-line interface to call the functions of the server in order to check all these functions have a good performance.

Tests of OSM WIM Plugin – parent Controller: Once the OSM plugin was developed, it was also used to do a cross-test with the server. In this way, using scripts to call the plugin (simulating the OSM), it should execute the given orders on the server and obtain the pretended results from the server, which should manage in the right way with the petitions from the simulated OSM. A set of tests was also performed with the OSM located at University of Bristol.

Tests of the TAPI of the Optical Controller: A set of tests has been executed to retrieve the optical topology for a context and to create a service in the Photonic layer. Both DSR and Photonic layers have been successfully performed.

3.1.6 KPIs

KPI Description	L2VPN Creation with Optical creation
Context	The Parent Controller manages the operations received from OSM and it should configure the network following these requests. The worst-case scenario is when there is a need to set up an optical connection in addition to configuring the packet layer. The following latency components are included: - Creation of the L2VPN Service (only interaction with database) - Configuration of L2VPN end points (writing in database + configuration of endpoints and packet network via packet controller) - Retrieval of updated optical topology - Multi-layer path computation - Configuration via TAPI RESTCONF of the optical connection Note that three calls are needed from OSM to configure the full service.
Target	<10s
Assessment	Measurement of the average time that is used to process a full L2VPN creation, including all the calls from OSM.
Relationship to project KPIs	**MH2**. METRO-HAUL E2E PtP connection set-up time **MH3**. Set-up time of network service slice across METRO-HAUL **MH4**. Capacity of METRO-HAUL controller

3.1.7 Availability

All the developed code is available online on the METRO-HAUL gitlab repository:

https://gitlab.com/METRO-HAUL/parent-controller/parent-sdn-controller

3.1.8 Interfaces

The interfaces implemented in this component are the IETF L2SM (to allow the communication between OSM and the Parent Controller), and TAPI (to retrieve the optical topology and request the creation of optical services).

The services and the multi-layer topology are stored in a MongoDB Data Base and the communication with the database is done by MongoDB Wire Protocol.

3.2 Optical SDN Controller

3.2.1 Description

The SDN controller for the optical network is used to control devices that are either OpenConfig or OpenROADM based. The NBI for the SDN controller is based on TAPI version 2.1. The implementation covers the connectivity and topology models. Topology models are read-only, to allow topology discovery by other subsystems by means of REST operations (it is also possible to discover additional devices if their YANG data model is known and registered). The software can accommodate hybrid deployments in which the control of the transceivers is decoupled from the control of the Open Line System.

The SBI is based on NETCONF over SSH transport. Current implementation supports theOpenConfig device model for platform and the terminal device or OpenROADM device version 2.2. There is also an implementation for an SBI based on TAPI RESTCONF for an OLS.

3.2.2 Sub-components

The main sub-components are

- **ONOS METRO-HAUL application.** This is the application that implements the main logic, that is able to parse requests coming from the NBI and proceed to configure the devices.
- **ONOS OpenConfig Terminal device driver**. This is the driver part, which maps high level instructions (flow model rules) to the specific instructions required to configure the device as defined by its data model.
- **ONOS OpenROADM device driver**. Similarly, for devices that follow the OpenROADM device model, the driver implements such aforementioned mapping.
- **ONOS Lumentum ROADM device driver**. Given the availability of this hardware component, METRO-HAUL has also provided an implementation of a driver that does the mapping based on the vendor-provided configuration operations.
- **TAPI 2.1 North Bound Interface (NBI).** This is the implementation of the interface that allows an end user (operator) to retrieve the topology or to request connectivity services to the SDN controller. It’s the main interface that complements the SDN controller GUI.
- **TAPI as SBI towards an OLS controller.** For the different disaggregation models, this SBI also allows the SDN controller to delegate some of the functions to the Open Line System (OLS)

3.2.3 Status

At the time of writing, the METRO-HAUL target implementations are mostly complete, only pending extensions based on feedback collected during regression testing, integration testing, and projectwide integrations with other components. In particular, support for OpenROADM is complete and demonstrated. This includes implementation of a connection cache, communications with the OpenROADM device to retrieve data from the device driver, and implementation of the DeviceDiscovery behaviour, the CrossConnect behaviour, the LambdaQuery, and the flowRule programmability. This enables ONOS to control the device as a ROADM in the optical model, to retrieve the status of the internal connections, and to program cross-connections upon request following the OpenROADM device driver. Similarly, the OpenConfig Terminal Device support for behaviours of LambdaQuery, FlowProgrammable, and DeviceDiscovery. It supports an OpenConfig TerminalDevice and Optical Transport models. There is also an implementation of a TAPI SBI enabling discovery of Service Interface Points (SIPs) from a TAPI-enabled OLS controller, to be used in a hierarchical setting with an OLS controller. Finally, there is the export of TAPI topology context in NBI and basic processing of TAPI Connectivity Requests Remote Procedure Calls (RPCs), as previously mentioned.

3.2.4 Related testbed and platform

Most of the source code is open source and has been submitted and integrated into the main ONOS repository. The functional component has been tested in partners' testbeds (CTTC / Telefonica / CNIT / TIM) as well as in several public demonstrations and events. In particular let us mention the following events, publications, and demos (non-exhaustive list):

Initial description and validations

R. Casellas et al “METRO-HAUL: SDN control and orchestration of disaggregated optical networks with model-driven development”, ICTON2018

R. Casellas et al “METRO-HAUL: Supporting Autonomic NFV Services over Disaggregated Optical Networks”, EUCNC2018

Most recent demonstrations

R. Morro et al. “Automated End to End Carrier Ethernet Provisioning over a Disaggregated WDM Metro Network with a Hierarchical SDN Control and Monitoring Platform” ECOC 2018.

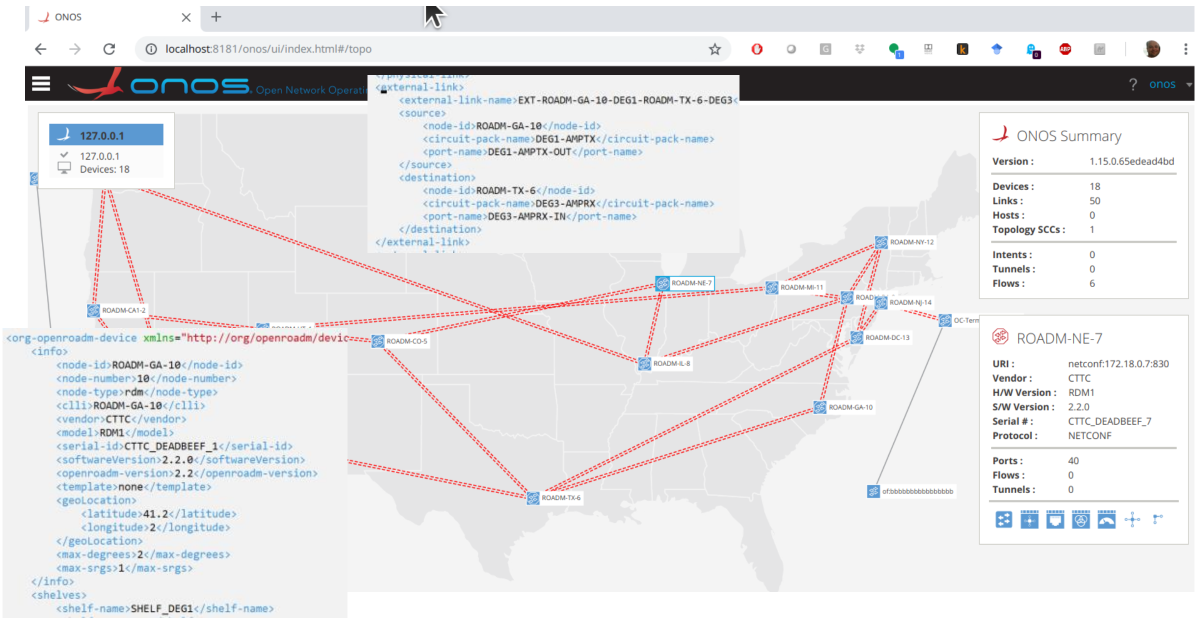
OFC2019 Demo A. Campanella et al. “ODTN: Open Disaggregated Transport Network. Discovery and control of a disaggregated optical network through open source software and open APIs”.

OFC2019 Demo P. R. Esmenats et al, “Autonomic NFV Network Services on Top of Disaggregated Optical Metro Networks”

OFC2019 Demo S. Troia et al “Dynamic Virtual Network Function Placement over a Software-Defined Optical Network”

3.2.5 Functional Tests

The main functional test is related to the provisioning and removal of an end to end connection between terminal devices. For example, for an experimental evaluation, we consider an NSFNet topology with 14 OpenROADM devices (emulating the physical infrastructure) and with 42 links (see Figure 16). A partition/OVN is defined selecting 10 nodes and a subset of the OpenROADM degrees, resulting in a sub-graph of the original topology. Virtual agents are allocated in Docker containers in such a way that devices in the same OVN belong to the same IP subnet. A TAPI request is sent to the OVN SDN controller for the Network Media Channel (NMC). The experiment is triggered by posting a RESTCONF Remote Procedure Call (RPC) requesting a TAPI connectivity service. Due to space reasons, we do not detail the previous tests related to device discovery, device configuration and integrated and regression tests part of the development process.



```
<external-link>
    <external-link-name>EXT-ROADM-GA-10-DEG1-ROADM-TX-6-DEG3<
    <source>
        <node-id>ROADM-GA-10</node-id>
        <circuit-pack-name>DEG1-AMPTX</circuit-pack-name>
        <port-name>DEG1-AMPTX-OUT</port-name>
    </source>
    <destination>
        <node-id>ROADM-TX-6</node-id>
        <circuit-pack-name>DEG3-AMPRX</circuit-pack-name>
        <port-name>DEG3-AMPRX-IN</port-name>
    </destination>
</external-link>
```

```
<org-openroadm-device xmlns="http://org/openroadm/devic
    <info>
        <node-id>ROADM-GA-10</node-id>
        <node-number>10</node-number>
        <node-type>rdm</node-type>
        <clli>ROADM-GA-10</clli>
        <vendor>CTTC</vendor>
        <model>RDM1</model>
        <serial-id>CTTC_DEADBEEF_1</serial-id>
        <softwareVersion>2.2.0</softwareVersion>
        <openroadm-version>2.2</openroadm-version>
        <template>none</template>
        <geoLocation>
            <latitude>41.2</latitude>
            <longitude>2</longitude>
        </geoLocation>
        <max-degrees>2</max-degrees>
        <max-srgs>1</max-srgs>
    </info>
    <shelves>
        <shelf-name>SHELF_DEG1</shelf-name>
```

ONOS Summary

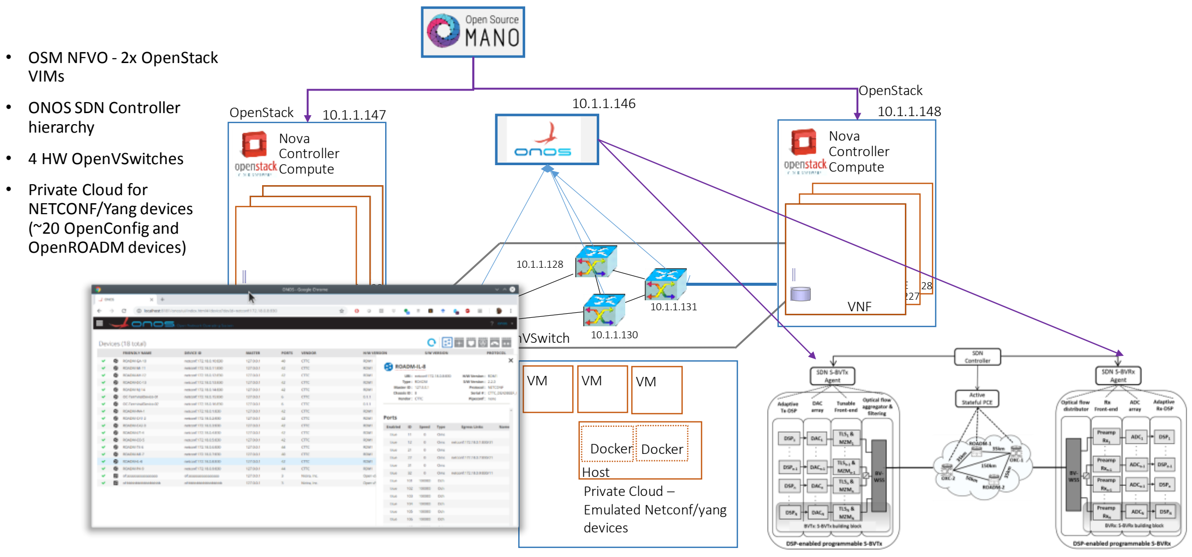
Version :	1.15.0.65edead4bd
Devices :	18
Links :	50
Hosts :	0
Topology SCCs :	1
Intents :	0
Tunnels :	0
Flows :	6

ROADM-NE-7

URI :	netconf:172.18.0.7:830
Vendor :	CTTC
H/W Version :	RDM1
S/W Version :	2.2.0
Serial # :	CTTC_DEADBEEF_7
Protocol :	NETCONF
Ports :	40
Flows :	0
Tunnels :	0

Figure 16. Optical SDN Controller example scenario and functional tests.

As stated, the component has been validated in several partners' testbeds. However, a significant part of the development is being done at the CTTC testbed, shown in Figure 17. This same testbed is being used for the demonstration to be reported in WP5.



• OSM NFVO - 2x OpenStack VIMs
• ONOS SDN Controller hierarchy
• 4 HW OpenVSwitches
• Private Cloud for NETCONF/Yang devices (~20 OpenConfig and OpenROADM devices)
Open Source MANO
OpenStack
10.1.1.147
10.1.1.146
OpenStack
10.1.1.148
Nova Controller Compute
Nova Controller Compute
onos
10.1.1.128
10.1.1.131
10.1.1.130
VSwitch
VNF
VM
VM
VM
Docker
Docker
Host
Private Cloud – Emulated Netconf/yang devices
SDN S-BVTx Agent
SDN Controller
SDN S-BVRx Agent
DSP-enabled programmable S-BVTx
DSP-enabled programmable S-BVRx

Figure 17. CTTC Testbed for optical SDN Controller integration and validation.

3.2.6 KPIs

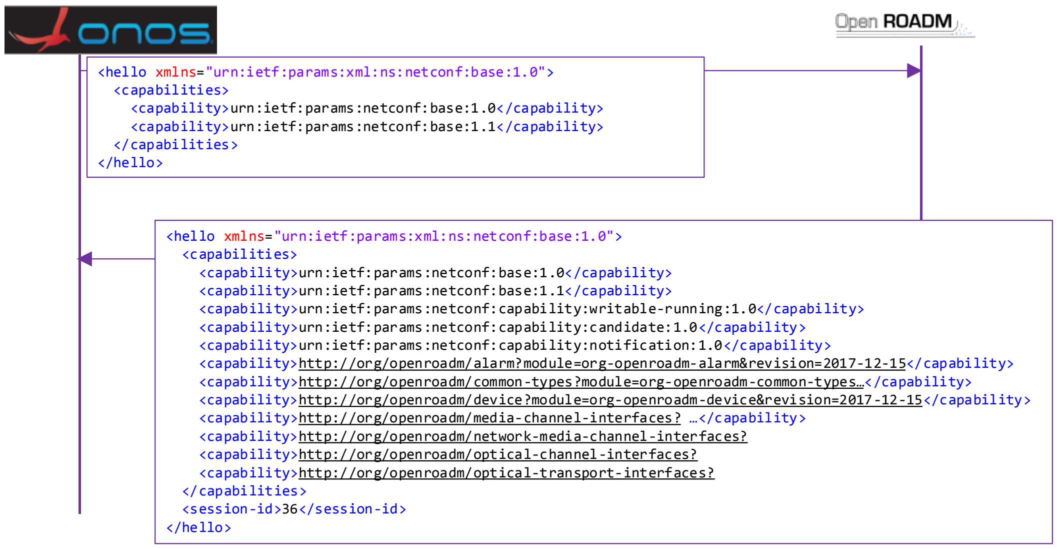
The KPIs for the SDN controller are related to the project KPIs MH1, MH2, and MH3 (since the optical controller is a part of the service setup delay) as well as MH4 (capacity of the METRO-HAUL SDN optical controller). The following table provides a synthesis of the KPIs.

KPI Description	**Setup Delay in the optical network** SDN-based management framework enabling fast configuration time in the Optical Layer to set up or reconfigure services handling 5G applications. This KPI is composed of the following elements: - Control plane latency and optical node reconfiguration delay - Time required to instantiate a network connection through the optical layer.
Context	The following assumptions are part of the context in which the KPI is assessed: - A single controller ONOS instance is running in a Linux Server with medium to high level hardware (i7, 32 Gb RAM, ..) - Virtual Containers are deployed emulating hardware devices. Devices will model OpenROADM nodes and AMEN/MCEN nodes with the same conditions as with real hardware. - The SDN Controller uses a NETCONF session over SSH transport towards the devices. - A dedicated network supports communication between the Controller and the devices, with a minimum of 10 or 100 Gb/s Ethernet. This KPI applies across multiple use cases. This KPI will be measured in a Control Plane testbed, with a server running the ONOS controller and multiple servers supporting instantiated containers.
Target	< 1 min with real hardware. In the case of ONOS controller in the scope of this deliverable, the target is met (with emulated hardware is of the order of milliseconds).
Assessment	The methodology involves the following steps: - Allocate network topologies - Configure the SDN Controller with the credentials - Measure the time it takes to setup a service.
Relationship to project KPIs	Related to MH1, MH2, MH3

KPI Description	**Control plane bandwidth and throughput** Control plane bandwidth and throughput required to set up a service across the network, between client ports of 2 transceivers.
Context	As previous KPI
Target	Feasible, assuming a dedicated control plane network to interact with the devices that is supported over a VPN or Ethernet LAN, with a minimum of 10 or 100Gb/s interfaces.
Assessment	The methodology involves the following steps: - Allocate network topologies - Configure the SDN Controller with the credentials - Measure the exchanged messages and control plane throughput
Relationship to project KPIs	Related to MH1, MH2, MH3 and MH4

KPI Description	**MH4. Capacity of the METRO-HAUL controller** The KPI defines the number of supported optical devices controlled by a single SDN Controller instance, and it is related to the number of NETCONF sessions that can be managed by such controller. This KPI is directly related to the processing capacity of the controller. It is assumed that the number of managed devices has a direct impact on relevant metrics such as the latency in configuring operations across the network, or the control plane overhead associated to managing such number of devices.
Context	Same as previous KPIs
Target	Target: control of 10-100 nodes (AMENs/MCENs, i.e., basically Open Disaggregated ROADMs). The output of this KPI will qualify in which conditions the target is met.
Assessment	The methodology involves the following steps: - Allocate network topologies with an increase number of network elements (nodes). - Configure the SDN Controller with the credentials (IP addresses, ports) of the elements. - Measure the time it takes until the NETCONF sessions are active, after the capability exchange and the initial device and topology discovery.
Relationship to project KPIs	This is MH4. Local tests verify that the KPI is met in the considered scenarios

Network Operations and Service Workflow



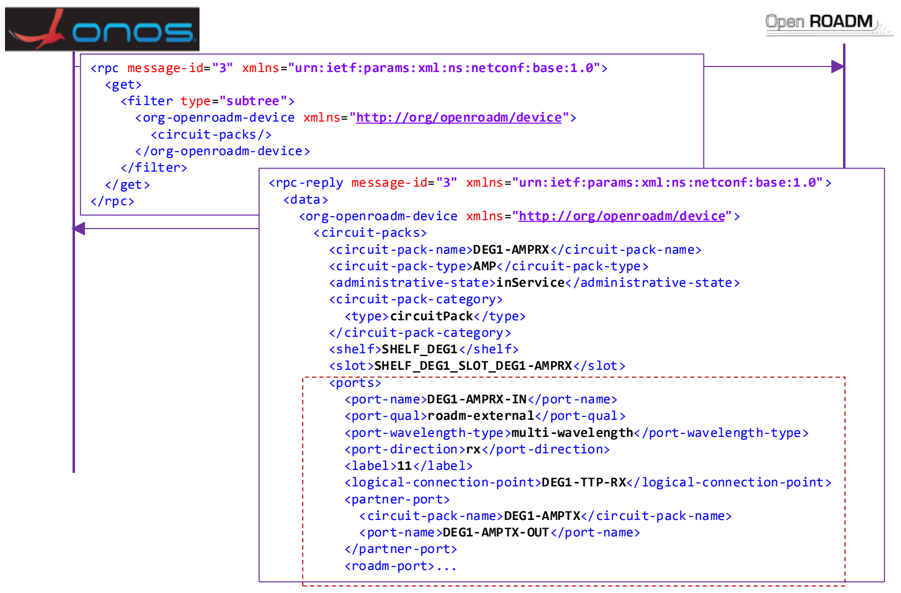
```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
  </capabilities>
</hello>
```

```
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
    <capability>http://org/openroadm/alarm?module=org-openroadm-alarm&revision=2017-12-15</capability>
    <capability>http://org/openroadm/common-types?module=org-openroadm-common-types…</capability>
    <capability>http://org/openroadm/device?module=org-openroadm-device&revision=2017-12-15</capability>
    <capability>http://org/openroadm/media-channel-interfaces? …</capability>
    <capability>http://org/openroadm/network-media-channel-interfaces?
    <capability>http://org/openroadm/optical-channel-interfaces?
    <capability>http://org/openroadm/optical-transport-interfaces?
  </capabilities>
  <session-id>36</session-id>
</hello>
```

Figure 18. Initial Capability Discovery between the SDN controller and the OpenROADM

Network Discovery: From the point of view of network operation, the procedure is as follows: first, the network operator configures the SDN controller (in our case the implementation is based on ONOS) with the list of devices and the NETCONF credentials. This means IP addresses, NETCONF ports (default is 830), user credentials, etc.

The SDN controller establishes a persistent NETCONF session with the device. Let us focus on the OpenROADM devices. First, there is an initial capability exchange, in which the NETCONF client (ONOS) discovers what models and features are supported (Figure 18 and Figure 19) using a HELLO. Next, the SDN controller issues <get> and <get-config> messages as needed to retrieve info about the devices. Typically, a <get> operation with a subtree filter of <org-openroadm-device><info/> allows retrieving basic data to add the device into the SDN controller device manager. Similar operations on the circuit packs and ports are used to retrieve internal connectivity and to discover port capabilities (Figure 18). For example, we query the list of circuit packs and their ports, e.g. the DEGREE1 AMPRX and see the DEG1-AMPRX-IN, this is a multi-wavelength port, external, and its partner port is DEG1-AMRTX-OUT. We can also see a logical connection point, the DEG1 Trail Termination Point for RX for that degree.



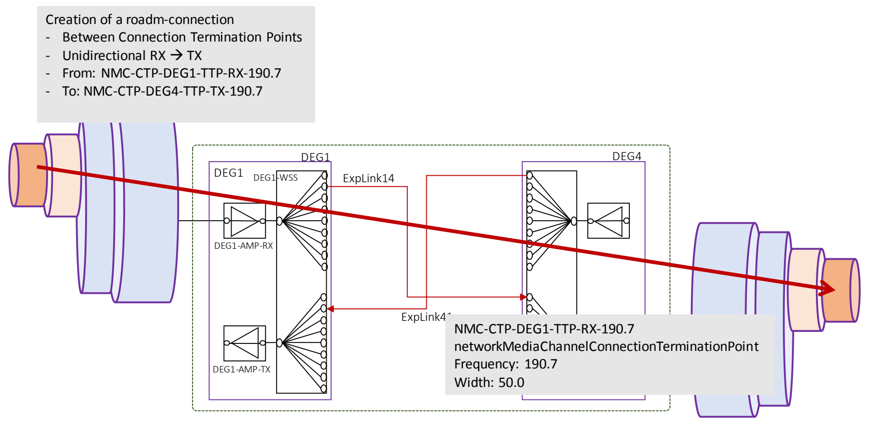
```
<rpc message-id="3" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <org-openroadm-device xmlns="http://org/openroadm/device">
        <circuit-packs/>
      </org-openroadm-device>
    </filter>
  </get>
</rpc>
```

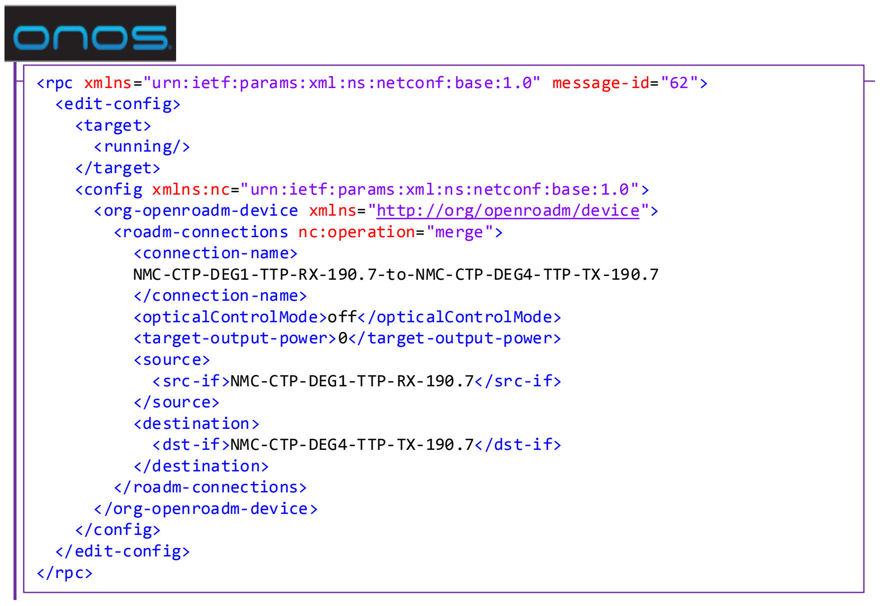
```
<rpc-reply message-id="3" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <org-openroadm-device xmlns="http://org/openroadm/device">
      <circuit-packs>
        <circuit-pack-name>DEG1-AMPRX</circuit-pack-name>
        <circuit-pack-type>AMP</circuit-pack-type>
        <administrative-state>inService</administrative-state>
        <circuit-pack-category>
          <type>circuitPack</type>
        </circuit-pack-category>
        <shelf>SHELF_DEG1</shelf>
        <slot>SHELF_DEG1_SLOT_DEG1-AMPRX</slot>
        <ports>
          <port-name>DEG1-AMPRX-IN</port-name>
          <port-qual>roadm-external</port-qual>
          <port-wavelength-type>multi-wavelength</port-wavelength-type>
          <port-direction>rx</port-direction>
          <label>11</label>
          <logical-connection-point>DEG1-TTP-RX</logical-connection-point>
          <partner-port>
            <circuit-pack-name>DEG1-AMPTX</circuit-pack-name>
            <port-name>DEG1-AMPTX-OUT</port-name>
          </partner-port>
          <roadm-port>...
```

Figure 19. NETCONF <get> operation to retrieve the list of circuit packs of the OpenROADM.

If the device supports it, we can retrieve external links to discover how OpenROADMs are inter-connected. This allows the SDN controller to construct a network topology view. If this is not supported, other means of topology discovery need to be defined, including, if need be, manual provisioning at the SDN controller. The result is that the SDN controller is aware of the network topology and end-to-end services may be requested, using, for example, the GUI.

Service Creation: In order to request a service, the user or operator retrieves the list of available Service Interface Points (SIPs) using the TAPI NBI and proceeds to request a connectivity service between a pair of SIPs (for a point-to-point connection). The request can specify if it applies, e.g., to a digital signal between two transceiver client ports or, alternatively, a network media channel between two transceiver line ports (or two OLS ports for an OLS controller). The SDN controller maps those SIPs to node edge points and performs a routing and spectrum assignment process that finds the k-shortest paths between the devices, and performs first fit spectrum allocation. Once completed, flow and forwarding rules are configured at each device: for the Terminal Device, a logical channel association is instantiated within the device between a client (transceiver) port and an optical channel component bound to the line port of the device. For each of the OpenROADM devices across the path, a ROADM internal connection is requested: i) OTS and OMS (optical transport and optical multiplex) interfaces are created within each degree (if not already existing), ii) Supporting Media Channel and NMC interfaces are created, iii) A roadm-connection object is created.


Creation of a roadm-connection
- Between Connection Termination Points
- Unidirectional RX → TX
- From: NMC-CTP-DEG1-TTP-RX-190.7
- To: NMC-CTP-DEG4-TTP-TX-190.7
DEG1
DEG1
DEG1-WSS
DEG1-AMP-RX
DEG1-AMP-TX
ExpLink14
ExpLink41
DEG4
NMC-CTP-DEG1-TTP-RX-190.7
networkMediaChannelConnectionTerminationPoint
Frequency: 190.7
Width: 50.0



```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="62">
  <edit-config>
    <target>
      <running/>
    </target>
    <config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
      <org-openroadm-device xmlns="http://org/openroadm/device">
        <roadm-connections nc:operation="merge">
          <connection-name>
          NMC-CTP-DEG1-TTP-RX-190.7-to-NMC-CTP-DEG4-TTP-TX-190.7
          </connection-name>
          <opticalControlMode>off</opticalControlMode>
          <target-output-power>0</target-output-power>
          <source>
            <src-if>NMC-CTP-DEG1-TTP-RX-190.7</src-if>
          </source>
          <destination>
            <dst-if>NMC-CTP-DEG4-TTP-TX-190.7</dst-if>
          </destination>
        </roadm-connections>
      </org-openroadm-device>
    </config>
  </edit-config>
</rpc>
```

Figure 20. Creation of a ROADM connection between two Connection Termination Points.

For example, to create a unidirectional express connection supporting a 50GHz signal centred at 190.7 GHz from degree 1 to degree 4 (Figure 20), the SDN controller proceeds as follows. First, assume that the OTS and OMS interfaces are configured at the ROADM degrees; these interfaces correspond to the lower layers of the OTN model; OTS corresponds to the Optical Transmission Section, e.g., between amplified fibre sections and OMS to the Optical Multiplex section. Next, create Media Channel (MC) interfaces (via an <edit-config> operation) over the OMS interfaces (by convention named MC-TTP-DEG1-TTP-RX-190.7 and MC-TTP-DEG1-TTP-RX-190.7, with a min-freq: 190.675 and max-freq: 190.725, which specifies the supporting media channel. Next, create NMC interfaces over the MC interface, specifying a centre frequency and width, inducing Connection Termination Points (CTPs). Finally, establish the unidirectional connection between CTPs, from the interface NMC-CTP-DEG1-TTP-RX-190.7 to NMC-CTP-DEG4-TTP-TX-190.7. Once the cross-connections have been provisioned at all the nodes along the path and the transceivers configured with the appropriate transmission parameters, the service is active.

Experimental scenario

To test the KPIs, the system has been tested with two reference topologies as shown in Figure 21, modelling an NSFNet (14 nodes) and a METRO-HAUL European network (10 nodes). In order to scale to O(100) of devices (it has been tested with 80 max), the most critical part is the initial convergence in which the ONOS controller queries all the devices to get the ports, capabilities, etc. Given the non-optimized current implementation of ONOS dynamic configuration stores, this can take several minutes. That said, this is a process that happens typically at start up, and it is not deemed critical.

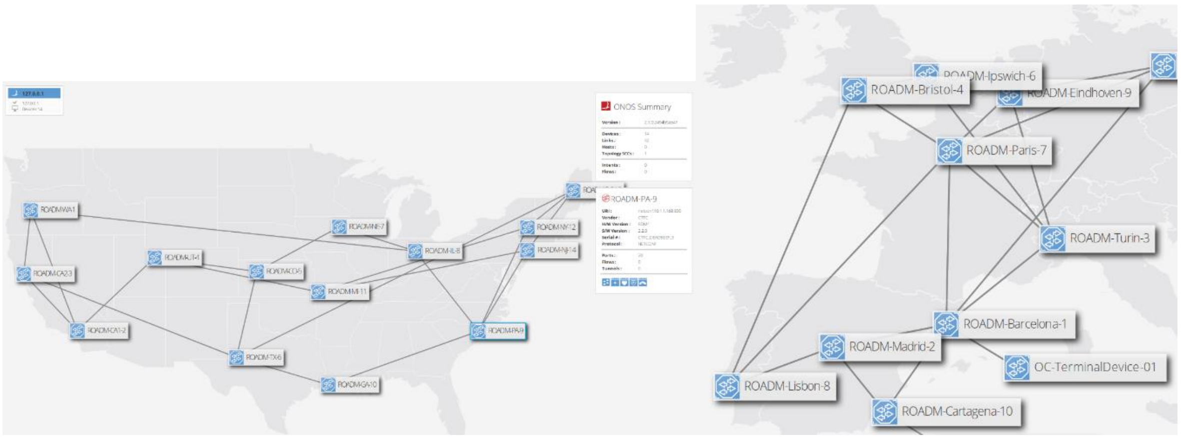
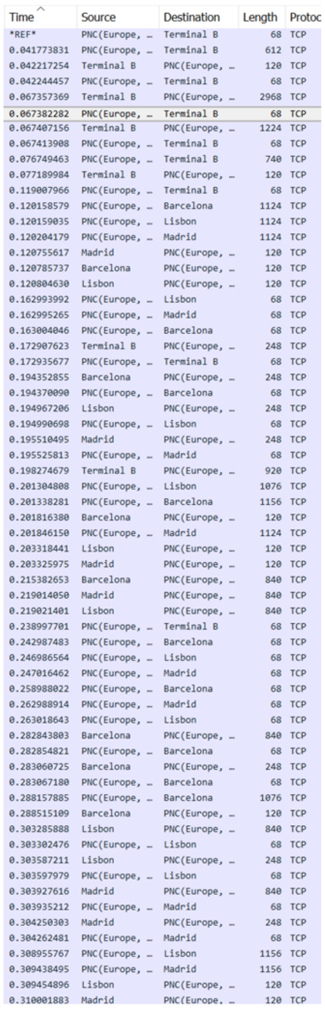
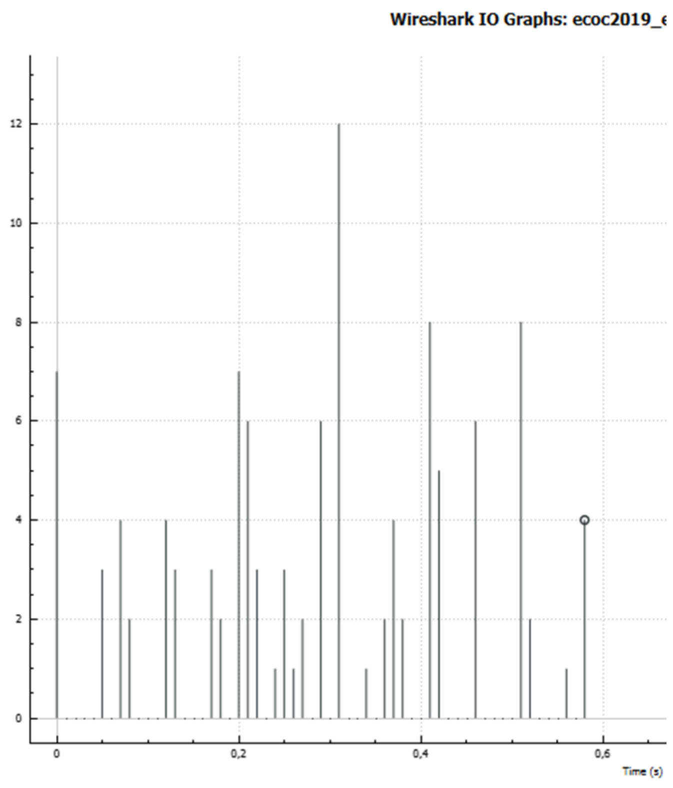


Figure 21. ONOS SDN Optical Controller reference topologies.

An end-to-end request for a network media channel is requested starting when the TAPI connectivity request is received. In the worst case, it is O(800ms), including the parallel provisioning. A sample service in the Europe (Dom B) abstracted node is mapped to 3 cross-connections in nodes Lisbon, Madrid, and Barcelona, as shown in Figure 22. Within the domain, the provisioning is O(500ms). Note that there is an overhead, given the XML encoding, the SSH transport for NETCONF, and the need to send multiple messages (creation of MC, NMC interfaces, and connections, OMS and OTS interfaces at each degree port are pre-created). As shown in the figure, there are O(100) IP packets exchanged between controller and agents, but the overall throughput is relatively low, making latency the critical aspect. Setup delays are computed without taking into account hardware configuration delays with real ROADMs, which can be of the order of seconds/minutes. Setup delay due only to the control plane has little impact on the performance.



Time	Source	Destination	Length	Protoc
REF	PNC(Europe, …	Terminal B	68	TCP
0.041773831	PNC(Europe, …	Terminal B	612	TCP
0.042217254	Terminal B	PNC(Europe, …	120	TCP
0.042244457	PNC(Europe, …	Terminal B	68	TCP
0.067357369	Terminal B	PNC(Europe, …	2968	TCP
0.067382282	PNC(Europe, …	Terminal B	68	TCP
0.067407156	Terminal B	PNC(Europe, …	1224	TCP
0.067413908	PNC(Europe, …	Terminal B	68	TCP
0.076749463	PNC(Europe, …	Terminal B	740	TCP
0.077189984	Terminal B	PNC(Europe, …	120	TCP
0.119007966	PNC(Europe, …	Terminal B	68	TCP
0.120158579	PNC(Europe, …	Barcelona	1124	TCP
0.120159035	PNC(Europe, …	Lisbon	1124	TCP
0.120204179	PNC(Europe, …	Madrid	1124	TCP
0.120755617	Madrid	PNC(Europe, …	120	TCP
0.120785737	Barcelona	PNC(Europe, …	120	TCP
0.120804630	Lisbon	PNC(Europe, …	120	TCP
0.162993992	PNC(Europe, …	Lisbon	68	TCP
0.162995265	PNC(Europe, …	Madrid	68	TCP
0.163004046	PNC(Europe, …	Barcelona	68	TCP
0.172907623	Terminal B	PNC(Europe, …	248	TCP
0.172935677	PNC(Europe, …	Terminal B	68	TCP
0.194352855	Barcelona	PNC(Europe, …	248	TCP
0.194370090	PNC(Europe, …	Barcelona	68	TCP
0.194967206	Lisbon	PNC(Europe, …	248	TCP
0.194990698	PNC(Europe, …	Lisbon	68	TCP
0.195510495	Madrid	PNC(Europe, …	248	TCP
0.195525813	PNC(Europe, …	Madrid	68	TCP
0.198274679	Terminal B	PNC(Europe, …	920	TCP
0.201304808	PNC(Europe, …	Lisbon	1076	TCP
0.201338281	PNC(Europe, …	Barcelona	1156	TCP
0.201816380	Barcelona	PNC(Europe, …	120	TCP
0.201846150	PNC(Europe, …	Madrid	1124	TCP
0.203318441	Lisbon	PNC(Europe, …	120	TCP
0.203325975	Madrid	PNC(Europe, …	120	TCP
0.215382653	Barcelona	PNC(Europe, …	840	TCP
0.219014050	Madrid	PNC(Europe, …	840	TCP
0.219021401	Lisbon	PNC(Europe, …	840	TCP
0.238997701	PNC(Europe, …	Terminal B	68	TCP
0.242987483	PNC(Europe, …	Barcelona	68	TCP
0.246986564	PNC(Europe, …	Lisbon	68	TCP
0.247016462	PNC(Europe, …	Madrid	68	TCP
0.258988022	PNC(Europe, …	Barcelona	68	TCP
0.262988914	PNC(Europe, …	Madrid	68	TCP
0.263018643	PNC(Europe, …	Lisbon	68	TCP
0.282843803	Barcelona	PNC(Europe, …	840	TCP
0.282854821	PNC(Europe, …	Barcelona	68	TCP
0.283060725	Barcelona	PNC(Europe, …	248	TCP
0.283067180	PNC(Europe, …	Barcelona	68	TCP
0.288157885	PNC(Europe, …	Barcelona	1076	TCP
0.288515109	Barcelona	PNC(Europe, …	120	TCP
0.303285888	Lisbon	PNC(Europe, …	840	TCP
0.303302476	PNC(Europe, …	Lisbon	68	TCP
0.303587211	Lisbon	PNC(Europe, …	248	TCP
0.303597979	PNC(Europe, …	Lisbon	68	TCP
0.303927616	Madrid	PNC(Europe, …	840	TCP
0.303935212	PNC(Europe, …	Madrid	68	TCP
0.304250303	Madrid	PNC(Europe, …	248	TCP
0.304262481	PNC(Europe, …	Madrid	68	TCP
0.308955767	PNC(Europe, …	Lisbon	1156	TCP
0.309438495	PNC(Europe, …	Madrid	1156	TCP
0.309454896	Lisbon	PNC(Europe, …	120	TCP
0.310001883	Madrid	PNC(Europe, …	120	TCP



Measurement	Captured
Packets	112
Time span, s	0.583
Average pps	192.1
Average packet size, B	394
Bytes	44084
Average bytes/s	75 k
Average bits/s	604 k

Figure 22. Example measurement of KPI for setup delay and throughput in a domain

3.2.7 Availability

The implemented controller has been contributed as Open Source to the upstream ONOS SDN Controller project. Next, we list key change sets that reflect the contributions. It is worth noting that submitting code of the demonstrator is still ongoing, following the rules, guidelines, and processes defined by the upstream project. It is also worth noting that any source code that is done for the specific demonstration of the project, but which is not integrated in the main project (e.g., being too specific, too research-oriented for a production environment, etc.) will still be available in the METRO-HAUL internal Repository, located at https://gitlab.com/METRO-HAUL/optical-controller/onos-controller. For the time being, contributions have been merged to the ONOS Main release (as within the METRO-HAUL participation in the ODTN project) https://github.com/opennetworkinglab/onos.

Let us illustrate the contributions process by listing some key changesets (merged to the upstream project), such as ONOS-7451 ODTN TAPI connectivity service https://gerrit.onosproject.org/#/c/17306/, ONOS-7828 ODTN OpenConfig FlowRule and LambdaQuery https://gerrit.onosproject.org/#/c/20382/, Added commons_jxpath as bundle of osgi_feature. https://gerrit.onosproject.org/#/c/20353/, TAPI 2.x rpc service registrator https://gerrit.onosproject.org/#/c/17400/, Lumentum ROADM-20 Whitebox, NETCONF driver supporting ROADM app https://gerrit.onosproject.org/#/c/17102/, Generic OpenConfig TerminalDevice driverhttps://gerrit.onosproject.org/#/c/18566/, Update sample files to TAPI >= 2.0.2, https://gerrit.onosproject.org/#/c/17854/ etc.

Two main implementations that are still under review are the OpenROADM device drivers and minor changes to other components. This is expected to be closed by Q3 2019.

3.2.8 Interfaces

The interfaces implemented in this component are mainly TAPI, OpenROADM drivers for optical matrixes, and OpenConfig for transceivers. They are described in detail in Section 4 of this document.

3.3 OLS controller

3.3.1 Description

Multi-layer orchestration requires knowledge of the topology of both the Optical and IP layers. Considering one of the METRO-HAUL scenarios, the Optical layer is managed by a technology-specific OLS controller that exposes the optical links and their features to the Orchestrator.

To install a service across a multi-layer network, the configuration of each device directly involved in the operations needs to be performed. ONOS is mostly device-based (i.e., OpenFlow oriented with “per device” configuration) and does not fit perfectly in common network environments, where technological domains are controlled by vendor-specific solutions such as proprietary controllers or Network Management Systems (NMS). Typically, such controllers provide a “service” scope provisioning interface to provision a complete path, instead of single configurations for individual devices. The reasoning is that a domain controller has a much more detailed understanding of the domain and finds better solutions than a general-purpose controller.

An example of such proprietary solutions is the Optical Line System (OLS) controller used in the project to configure ADVA’s optical infrastructure. The OLS controller is a (logically) centralized controller developed commercially to provide an abstraction for the underlying infrastructure and to facilitate control of the optical infrastructure via standardized northbound interfaces, like RESTCONF/NETCONF.

3.3.2 Current Transport API Specifications (work in progress)

The Transport API (TAPI) covers the following set of control plane functions and services:

1. The **topology service** handles the retrieval of information about topologies. There are different granularities defined for information retrieval, starting from the whole topology and going down to individual node-edge-point details.
2. The **connectivity service** manages connectivity services between service-endpoints. The service interface can be used with or without prior knowledge of the topology to create, retrieve, update, and delete (CRUD) connectivity services. Additionally, information about service-endpoints, connections, and their endpoints can be accessed.
3. The **path computation service** computes and optimizes point-to-point paths.

4. The **virtual network service** allows the user to define, retrieve and delete virtual network services, i.e., virtual network topologies. The virtual topologies correspond to reserved resources that can be controlled by the client.

5. The **notification service** is another piece of the model. The exposed notification types are currently limited to created and deleted objects, changed values, and changed states. Based on the types, notification subscriptions can be created, modified, deleted, suspended and resumed. In addition, notification records are retrievable. Alarms and performance monitoring are scheduled for the next release.

3.3.3 Status

Functions that are currently supported are:

1) **Discovery and provisioning of fully aggregated domains**, where client ports of transponders are exposed as Service Interface Points, and services can be created between transponders / muxponders inside the OLS domain.
2) **OLS configurations**: In this configuration, the OLS controller exposes two topologies, with one presenting the Open Line System to establish pure wavelength services (alien waves) and the other topology presenting the Transponder domain. Service establishment in the OLS domain supports the configuration of an alien wave tunnel and in the transponder domain activates the client and network side of the transponders using Ethernet interfaces on the client side and OTUx interfaces on the network side. Extensions proposed to the TAPI connectivity service with OTSI extensions were used to configure the wavelength used in both the alien wave case as well as the channel used at the network side of the transponders.

3.3.4 Availability

The Optical Line System (OLS) controller is the ADVA Network Manager System (NMS) and it is not publicly available.

3.3.5 Interfaces

The URL endpoints supported in the TAPI OLS controller implementation are described below (IP and port have to be changed accordingly);

- Base URL for all calls

```
http://{controller ip:port}/tapi/RESTCONF
```

- Full context

String CONTEXT = "config/context";

- Service Interface Point Details

```
interface Common {
  String SEPS = "config/context/service-interface-point";
  String SEPS_BY_UUID = "config/context/service-interface-point/{uuid}";
}
```

- Topology Details:

```
  interface Topology {
    String TOPOLOGY = "config/context/topology";
    String TOPOLOGY_BY_UUID = "config/context/topology/{uuid}";
    String NODES = "config/context/topology/{uuid}/node";
    String NODE_BY_UUID = "config/context/topology/{uuid}/node/{node_uuid}";
    String ONEPS = "config/context/topology/{uuid}/node/{node_uuid}/owned-node-
edge-point";
    String ONEP_BY_UUID = "config/context/topology/{uuid}/node/{node_uuid}/owned-
node-edge-point/{owned_node_edge_point_uuid}";
    String LINKS = "config/context/topology/{uuid}/link";
    String LINK_BY_UUID = "config/context/topology/{uuid}/link/{link_uuid}";
  }
```

- Connectivity service details:

```
  interface Connectivity {
    String CONNECTIVITY_SERVICES = "config/context/connectivity-service";
    String CONNECTIVITY_SERVICE_BY_UUID = "config/context/connectivity-
service/{uuid}";
    String CONNECTIONS = "config/context/connection";
    String CONNECTION_BY_UUID = "config/context/connection/{uuid}";
  }
```

3.4 SDN for Passive Optical Networks

3.4.1 Description

The SDN PON Controller is responsible for the control of the PON elements of the METRO-HAUL architecture. The NBI for the SDN PON controller is based on TAPI and realises the communication between the SDN PON Controller and the Parent Controller. The adopted SDN architecture is schematically depicted in Figure 23.

3.4.2 Sub-components

The main building blocks of the architecture are the Management Agent Software Framework (ConfD), the PON Configuration Agent (PCA), the PON Network Flow Agent (PNFA), and ONOS.

The ONOS controller platform is the central management system/orchestrator of the entire network. The ConfD is the main management and control entity and is responsible for retaining the configuration status of the PON. The abstraction of the PON as an L2 switch is based on YANG modelling and in particular on the legacy-switch model, but incorporating the QoS queue models of Broadband Forum. This integration allows to (re)configure the PON using the NETCONF protocol, with the configuration status stored into the Core engine Database (CDB). When configuration requests are arriving from the application layer via the North Bound Interface (NBI), the ConfD Core Engine Agent retrieves the data stored in the CDB and updates the configuration, while informing the other entities in real time of configuration status changes through a CDB API. The PCA, communicating with ConfD via the CDB API, is responsible for realizing the requested configuration changes to the PON elements: when the configuration is modified, the CDB engine informs the PCA of the changes and the PCA generates the appropriate CLI commands and executes them directly on the corresponding PON elements. The PNFA implements the flow management of the infrastructure. PNFA uses the OpenFlow protocol a) to receive flow requests from the OpenFlow controller of ONOS and to forward them to the MCA; and b) to inform ONOS' OpenFlow-controller of the activation of a new switch (PON) and to establish the initial flows. The MCA is responsible for translating these requests to the appropriate element mapping (e.g., VLAN to queue) and to execute the appropriate commands.

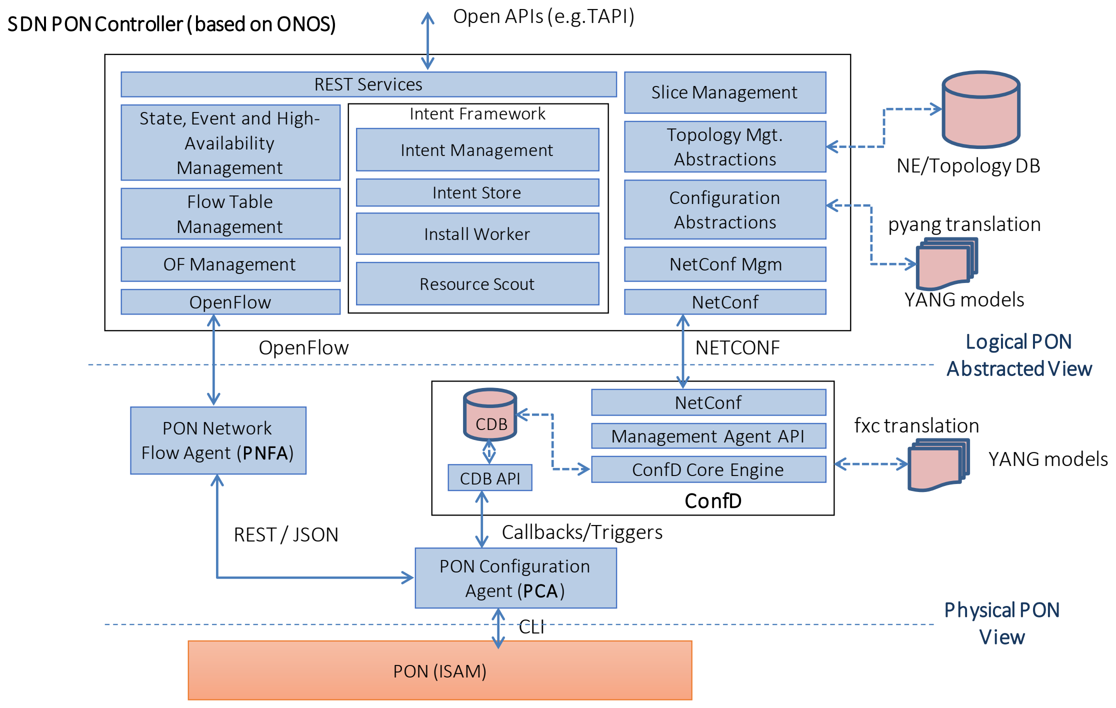


Figure 23. SDN Control of PON network

Under the METRO-HAUL abstraction framework, the SDN PON Controller handles PONs as legacy switches so a unified management system can be used for all switching entities in the network. Finally, in addition to physical PONs, logical PONs are exposed to the SDN PON Controller by ConfD, so, through its slice management entity, it can further support sliceability though the NBI.

3.4.3 Status

At the time of writing, the building blocks of the architecture are mostly complete. In detail:

- The PON Configuration Agent is implemented.
- The PON Network Flow Agent is implemented.
- The basic building blocks of the SDN PON Controller are implemented, including a NETCONF client based on Tail-f Java NETCONF Client (JNC) and an OpenFlow client based on OpenDaylight.

We are currently working on the:

- Definition of the interface/APIs (e.g., TAPI) towards the parent controller (Orchestrator).

- Implementation of the interface/APIs and integration with the parent controller (Orchestrator).

Description and validations

E. Kosmatos, C. Matrakidis, A. Stavdas, S. Horlitz, Th. Pfeiffer, A. Lord, "A Dynamic Transportation Platform for Metropolitan Networks Exploiting PON Technology and a Novel Control-Plane", ECOC2018, TuD3, 23-27 Sept. 2018, Rome, Italy

E. Kosmatos, C. Matrakidis, A. Stavdas, T. Orfanoudakis, " An SDN Architecture for PON Networks Enabling Unified Management using Abstractions", ECOC2018, We2.64, 23-27 Sept. 2018, Rome, Italy

3.4.4 Related testbed and platform

For experimental validation, we implemented the PCA and PNFA components and we integrated all the already available components (like ConfD) in order to realise the architecture depicted in Figure 23. Regarding the PON equipment, a GPON system (ISAM 7330) provided from NOKIA was integrated into the testbed.

3.4.5 Functional Tests

A set of functional tests were executed in order to ensure that the implemented functions are working properly and their outcomes were in line with the project objectives.

A well-defined set of tests were executed in order to validate the performance of the platform under a set of project scenarios and against a set of predefined KPIs including control and data plane latency.

3.4.6 KPIs

KPI Description	Control plane latency
Context	Measure and validate the following control plane latency components: • Latency in PCA for PON reconfiguration • Latency in PCA for PON reconfiguration causing DATA interruption (ms) • Total latency in PCA • Total NETCONF configuration cycle (request-update-response) (ms)
Target	Target: < 10 s
Assessment	The testbed architecture is described in the sections below. It includes the software architecture depicted in Figure 23, while the PON equipment, a GPON system (ISAM 7330), was provided by NOKIA.
Relationship to project KPIs	Related to MH1, MH2, MH3

KPI Description	Data plane interruption time
Context	Measure and validate the data interruption time using different traffic types (e.g., UDP and TCP) and different service types (e.g., video streaming, FTP

	download/upload)
Target	Target: < 20 s
Assessment	The testbed architecture is described in the following sections. It includes the software architecture depicted in Figure 23, while the PON equipment, a GPON system (ISAM 7330), was provided by NOKIA.
Relationship to project KPIs	Related to MH1, MH2, MH3

3.4.7 Availability

The developed software components are not publicly available.

3.4.8 Interfaces

At the time of writing, the interface (TAPI) towards the parent controller (Orchestrator) is under definition and implementation.

3.5 Service orchestrator

In the METRO-HAUL architecture, as mentioned in Section 2.2, the WIM is responsible for interconnecting the VNFs as part of an NS, where the VNFs are deployed in multiple PoPs (AMENs or MCENs) in different geographical locations. The WIM provisions the network using underlying SDN controllers, which in the case of METRO-HAUL, are separate for packet and optical domains. For the METRO-HAUL project, OSM is used as the reference ETSI-based NFV Management and Orchestration (MANO) system. OSM is an open-source MANO implementation under the umbrella of ETSI, in charge of orchestrating the VNF placement and life cycle in one or more data centres, potentially connected by a transport network. OSM, as part of Release 5, has been extended to connect to WIMs and provision connectivity between VNFs hosted at multiple VIMs registered with OSM. Figure 24 shows the setup where an NS consisting of 3 VNFs is deployed by OSM. The placement for VNFs 1 and 2 has been specified on PoP-A, and for VNF 3 on PoP-B. The virtual link connecting the VNF 2 and VNF 3 spans the WAN link interconnecting the PoP. The network for this link is provisioned by the WIM under instruction from OSM, using SDN controllers.

OpenStack is used as the VIM in the METRO-HAUL project. It is configured to provision Layer 2 provider networks attached to the VMs in the compute nodes. When OSM instantiates an NS as shown in Figure 24 with the appropriate VNF placement option, it also instructs OpenStack in PoP-A to create a provider network to attach to VNF-2 which spans out of the compute node at PoP-A. OpenStack assigns a VLAN id “A” to this provider network. Similarly, a provider network is created at PoP-B with VLAN id “B”. Once the VNFs and the networks are created, the VIM sends the VLAN information to the OSM. OSM detects that the virtual link connecting VNFs 2 and 3 spans multiple datacentres. Once OSM receives the VLAN information, it uses a pre-defined WIM port mapping to obtain the WAN endpoints connected to each datacentre. These endpoints along with VLAN information about the PoPs is sent to the WIM which uses this data to provision a network between the endpoints. The endpoints may be a physical endpoint for (for example) a combination of port number with OpenFlow DPID; or may be logically defined, such as a Transport API (TAPI) service interface point (SIP).

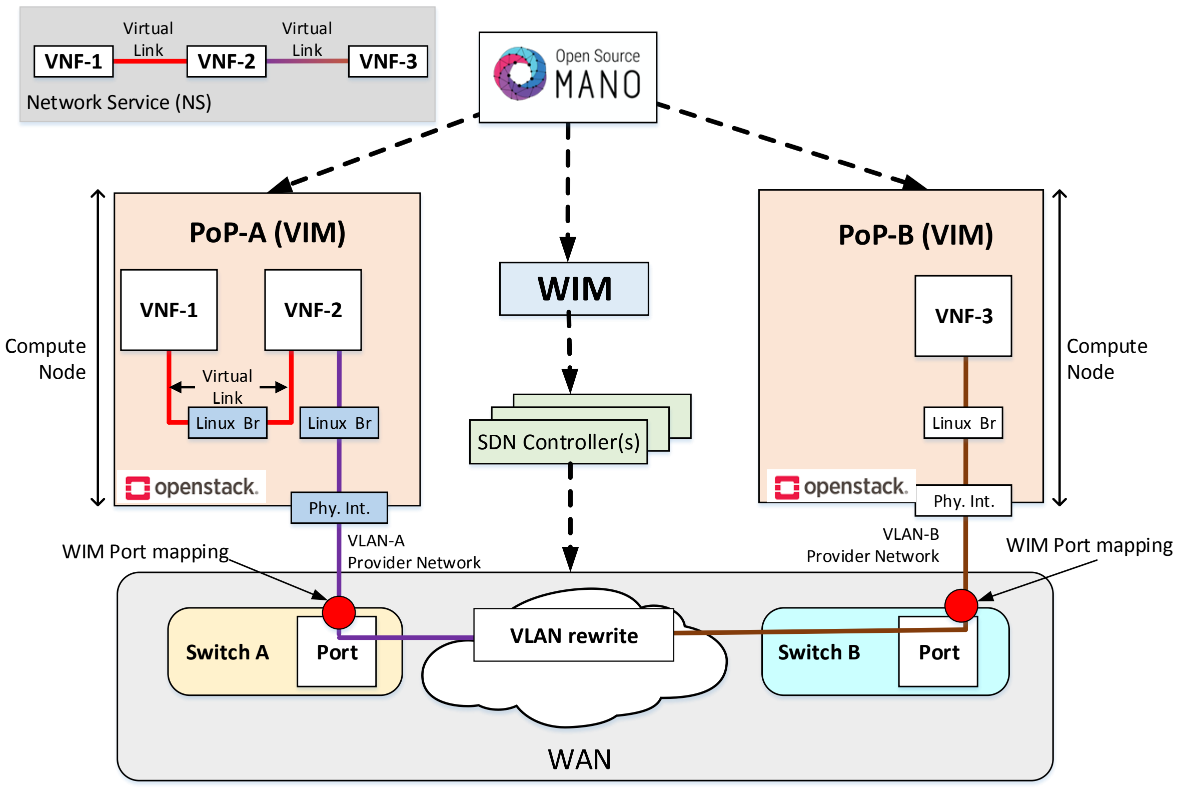


Figure 24: Integrated SDN and ETSI NFV/MANO system

The following sections provide details of the work carried out to enable OSM to provision connectivity between multiple PoPs using the WIM.

3.5.1 OSM extensions for WIM

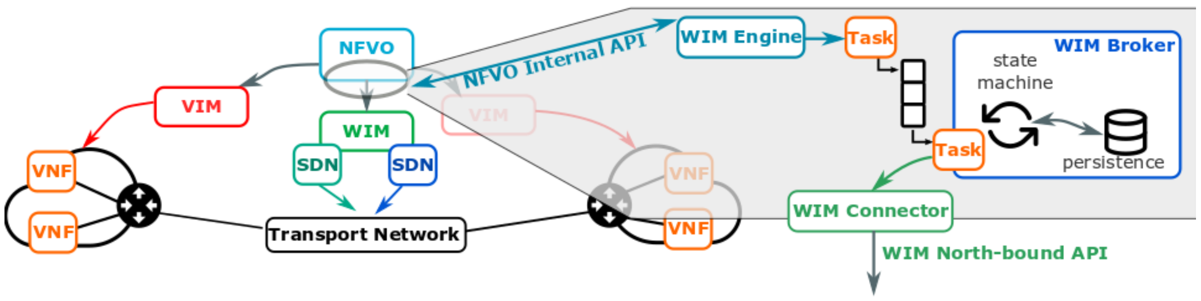


Figure 25: WIM Extension for OSM

In this section, the detailed internal working of the OSM extension for WIM are explained. We introduce sub-components inside the NFVO, as indicated in Figure 25. This architecture is based on the execution of asynchronous tasks to avoid blocking behaviour and to improve the overall network service provisioning time. The WIM Engine has three responsibilities: firstly, to provide information to the NFVO about the available WIMs and the characteristics of the connectivity they can sustain; secondly, to check the feasibility of the presented placement solution; and finally, to specify and schedule a series of tasks that encode the instructions for establishing WAN links. While each task works as a standalone processing unit, the WIM Broker coordinates their threaded execution in a task queue by implementing a state machine upon activation. By tapping into a shared communication mechanism internal to the NFVO (implemented via either a common database or event stream), tasks can verify preconditions and fire actions to the external WIMs accordingly. As an example, special information, such as VNF IP and VLAN segmentation id, might need to be published by a VIM to the NFVO after its workload is processed. The VLAN information is part of the endpoint information that OSM sends to the WIM. To wait for this information, the tasks can be suspended, and rescheduling them is the responsibility of the WIM Broker. Conversely, when all the preconditions are met, a task invokes commands on a WIM Connector.

3.5.2 WIM connectors

The WIM connector for a particular WIM is based on an abstract WIM connector class. Depending on the type of WIM, the abstract WIM connector class can be extended to accommodate the WIM-specific North-bound APIs (NBIs). When OSM is deploying an NS, it detects if a link connecting multiple VNFs spans multiple VIMs. This invokes the OSM to access the relevant WIM to provision the network between the VIMs, where the WAN network interconnecting the VIMs is managed by this WIM. The WIM requires the endpoints as explained earlier, and they include the port mapping and the VLANs. In the following, we describe two WIM connectors.

3.5.2.1 Transport API based WIM connector

The choice of TAPI as the interface between the OSM and the WIM is motivated by multiple factors. TAPI, as defined by the ONF, has fulfilled its goal of having a common standardized North Bound Interface (NBI) across multiple network controllers, to be used, e.g., by a network orchestrator or applications. It builds on core models that are technology agnostic (aligned with the ONF Core Information Model, which defines a common object model for all types of Software Defined Networks, including components like network resources or service constructs) while allowing technology-specific extensions for relevant layers. TAPI follows a model-driven development: models are defined in UML and automatically translated into YANG which, in turn, may be automatically validated, and stubs can be automatically generated for common programming languages. From the point of view of maturity and adoption, it has been demonstrated in multiple proof-of-concepts, interop events and is supported by many vendors [OIF-ONF]. It has been adopted by several initiatives and SDOs, such as MEF OpenCS Optical Transport or, for the latest v2.1 release, the ODTN Project [ODTN] as the NBI for a controller of an optical disaggregated transport network. Release 2.0 added the ability to take into account node constraints, protection services and consistent OAM and monitoring, and the latest 2.1 release includes new models for the photonic layer, with support for flexi-grid network media channels, including model for the OCH, OTSi, OTSiA, OTSiG, OMS, OTS, and Media channels as per ITU-T G.872 (2017) version 4.

A preliminary implementation of TAPI based WIM connector with MANO for NFV orchestration over a Multi-PoP infrastructure has been done. For demonstrating its applicability, the same network service composed by 3 VNFs (Figure 26a) was deployed over an optical infrastructure. Figure 26b shows the experimental setup, which consists of two OpenStack datacentres interconnected by a combination of packet switches and optical cross connects (OXCs) which are SDN-enabled and managed by a combination of SDN controllers. On top of the SDN controller, we employ a TAPI proxy based on a TAPI 2.0 reference implementation [TAPI]. The TAPI proxy exposes two service endpoints (svc_ep_1 and svc_ep_2), at both edges of the interconnection network between the datacentres, to OSM. The information about these service endpoints is added to OSM before network service deployment as part of a port mapping process, making them available for usage.

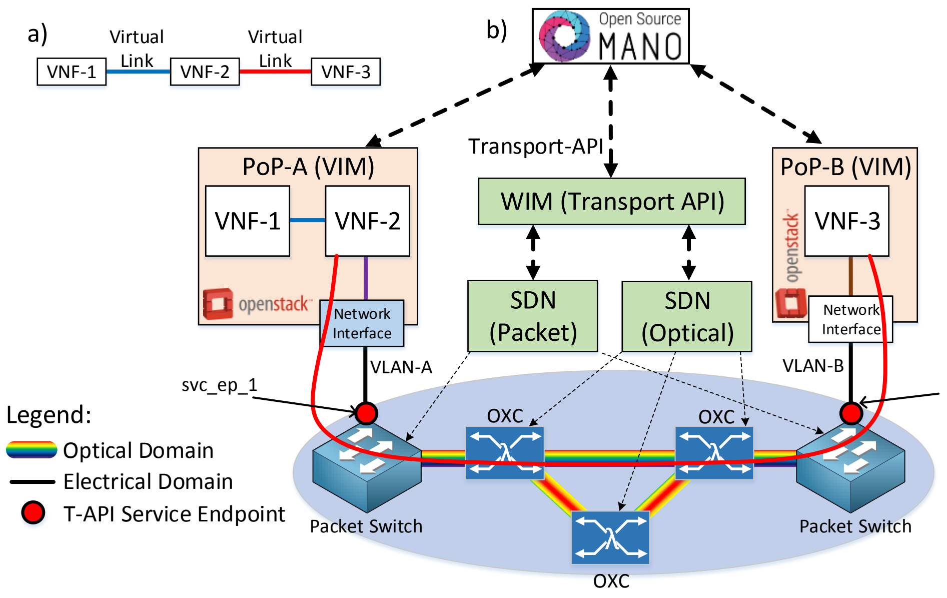


Figure 26: a) Network Service b) Testbed for TAPI with OSM

During the NS deployment phase, OSM detects that the virtual link connecting VNFs 2 and 3 spans multiple datacentres. Once OSM receives the VLAN information, it uses the pre-defined port mapping to obtain the TAPI service endpoints connected to each datacentre. The TAPI service endpoint (svc_ep) and the VLAN for each datacentre is sent to the TAPI proxy to provision a network between the datacentres. The TAPI proxy translates this information into low-level device specific flow installation using the SDN controller to interconnect the two VLANs at the either datacentre. More details about the message sent to TAPI based WIM from OSM are mentioned in Section 4.4.

3.5.2.2 L2 service model connector

3.5.2.2.1 Description

The L2 service model connector is the component responsible for the connection between the OSM and the Parent Controller to handle connectivity between METRO-HAUL Network endpoints to support connectivity between VNFs running in different VIMs. This component implements RFC 8466 “YANG Data Model for Layer 2 Virtual Private Network (L2VPN) Service Delivery” [RFC8466] defined by the IETF to describe an L2VPN service from the point of view of the consumer of the service. The connection to the parent controller is performed via RESTCONF [RFC8040] interface with the data encoded in JSON following RFC 8466 data model. So, this plugin implements all the methods defined for the WIM plugins by creating the necessary RESTCONF calls.

The Service Delivery Model of RFC 8466 models the L2 connection needed in the METRO-HAUL network. An **AMEN/MCEM Node** is seen as two separate entities. One of them is associated to the Compute infrastructure (compute nodes and internal switching), which hosts the VNFs, and the other one associated to the packet and optical network. From the modelling perspective, every compute infrastructure in an AMEN/MCEM model is considered a “site”, according to the terminology in RFC 8466. The L2 link between the compute infrastructure and the network node is considered a “bearer”. Every bearer can be used entirely for a single connection or can be split into multiple VLANs to be used for multiple connections. An example of an AMEN/MCEM Node and a detail of the bearer is shown in Figure 27.

3.5.2.2.2 Sub-components

The only component of the Plugin is the python class **wimconn_ietfl2vpn.py,** which implements all the functions to communicate between the OSM and the Parent Controller in order to create and manage the L2VPNs of the system. All these functions are explained in detail in section 4.4.2.

3.5.2.2.3 Status

The L2SM connector is functional and it performs all of the functions described in Section 4.4.2. to create an L2VPN Service on the Parent controller.

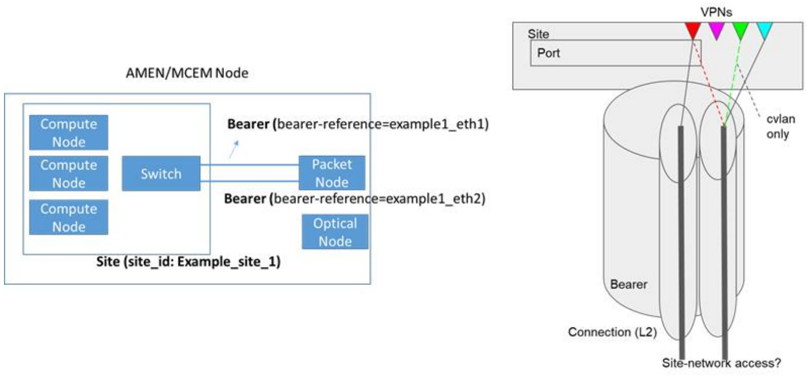


Figure 27: Example of an AMEN/MCEM Node

3.5.2.2.4 Tests

The Plugin has been tested on a local Swagger server which made the functions of the Parent Controller. All the results of the tests have been positive, all developed functions performed the desired objectives.

3.5.2.2.5 Availability

The plugin is available online at:
https://osm.etsi.org/gerrit/#/c/7434/1/osm_ro/wim/wimconn_ietfl2vpn.py

3.5.2.2.6 Interfaces

The interface of the plugin implements a RESTCONF+JSON interface with the RFC 8466 data model. The following operations are used by the plugin:

- Create connectivity service
- Delete connectivity service
- Delete all connectivity services
- Edit connectivity service
- Get connectivity service’s status
- Get all connectivity services

3.6 System for network virtualization and slicing

In this section, the system for network virtualization and slicing is presented. The system consists of an integrated environment for NS deployment across multiple datacentres/PoPs using a single OSM instance; where an NS comprises of multiple interconnected VNFs.

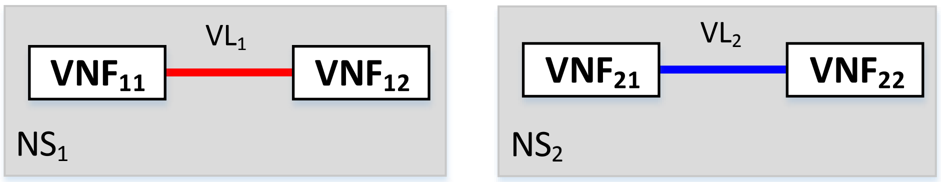


Figure 28: Reference NSs to explain system for network virtualization and slicing

To explain this system, assume there are two VIMs: VIM_A and VIM_B. A WAN interconnects the two VIMs. Reference Network Services for this example are NS_1 and NS_2 as shown in Figure 28, where VNF_{11} and VNF_{21} are to be deployed in VIM_A, and VNF_{12} and VNF_{22} are to be deployed in VIM_B. The VIMs are based on OpenStack version Pike. OSM natively supports integration with OpenStack based VIMs. An ETSI based Network Service Descriptor (NSD), which is based on the NSs as shown in Figure 28, is required to be defined on OSM. At the time of deploying an NS, the location, i.e., the VIM where a VNF of an NS must be deployed, is required to be specified. The same procedure is followed as explained in Section 3.6, where the OpenStack based VIMs deploy the VNFs; as shown in Figure 29. Furthermore, as explained in Section 3.6, when an NS is deployed using OSM which spans multiple PoPs/VIMs, OSM uses an external WIM to interconnect the PoPs. OSM provides the endpoints where the VIMs interface with the WAN network along with the VLANs used at each VIM for a Virtual Link (VL) interconnecting the VNFs.

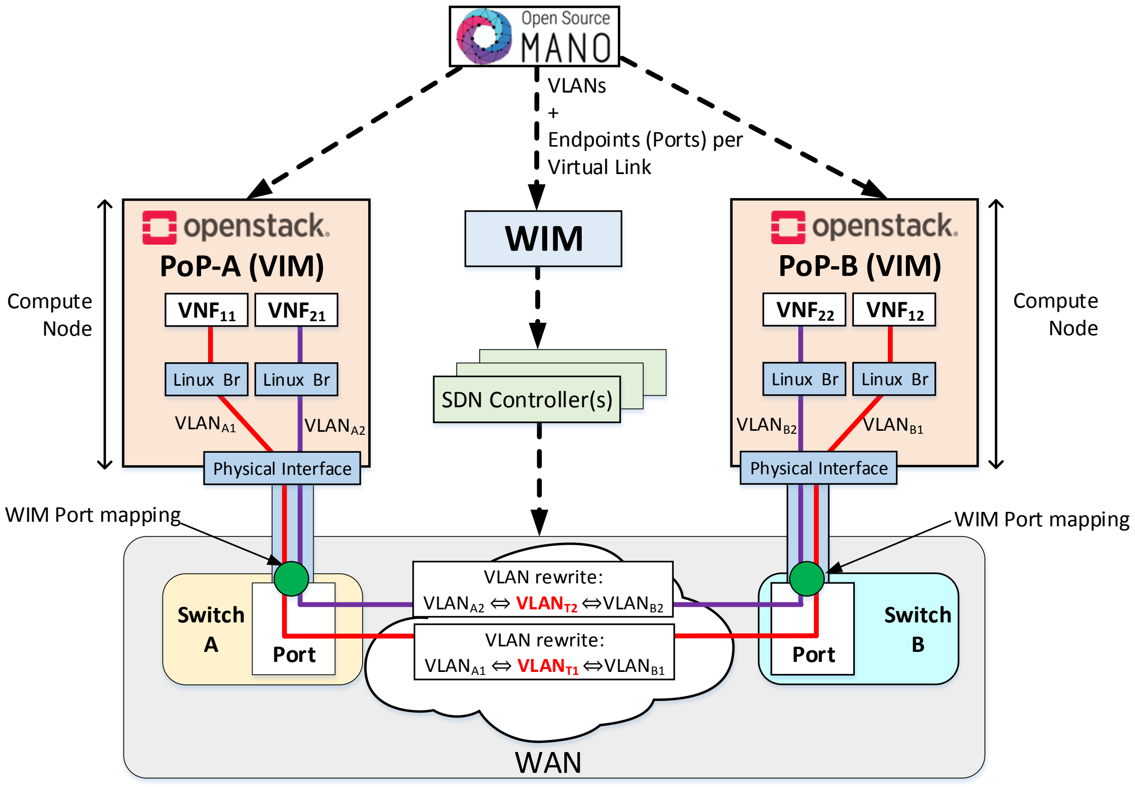


Figure 29: Network slicing in WAN

Assume that the VLAN assignment for the provider networks made by OpenStack for the VNFs is as shown in Figure 29. The VLAN assignment per VL is received by OSM from OpenStack instances once the provider networks are deployed and connected to the VNFs. OSM then proceeds to request the WIM to deploy the network per VL basis by sending the endpoints (physical or logical) along with the VLANs. The WIM interconnects the provider networks at both OpenStack instances using the WAN. An intermediate transport VLAN is used for e.g., $VLAN_{T1}$ for NS_1 is used to interconnect the $VLAN_{A1}$ with $VLAN_{B1}$ by performing a VLAN rewrite on the edge WAN switches connected to the VIMs as shown in Figure 29. A similar process is done for NS_2 with a separate transport $VLAN_{T2}$ to interconnect its respective VNFs. Since separate VLANs are utilized at each VIM and the WAN network, hence this fulfils the need for network slicing.

The WIM connector in OSM as part of Release 5 is equipped to send the VLAN and endpoint information to the WIM. The WIM, which is external to OSM, is responsible for setting up the transport VLANs per VL, to ensure proper slicing.

3.7 Monitoring and Data Analytics subsystem

3.7.1 Description and components

The detailed MDA subsystem architecture together with the related interfaces is shown in Figure 30. It consists of two sub-components:

- **MDA agents** are designed to collect measurements from one or more nodes in the disaggregated data plane. While each node controller usually controls one single node and exposes one single interface toward the SDN controller, MDA agents are designed to be in charge of monitoring and telemetry of a set of nodes. MDA agents consist of two building blocks, the *local configuration* module and the *local KDD* module. The local configuration module is in charge of receiving configuration, and the exposes *SBIm* interface.

  A number of node adapters (one per node) are used to implement the specific protocols exposed by every node controller through the *MONx* interface for monitoring data collection. Since different protocols for monitoring and telemetry might be considered for the *MONx* interface, node agents include bespoke node adapters implementing specific protocols and function mapping for the underlying node controller.

  Regarding the local KDD module, its scope is to apply data analytics to the measurements received from the nodes. Output of the data analytics procedure is forwarded to the MDA controller to implement network-wide control loops; two types of messages are supported: *i*) IPFIX-based monitoring messages including processed monitoring samples (i.e., values are averaged over the selected monitoring period, e.g., 15 minutes) using the *NBIm* interface; and *ii*) through asynchronous notifications using the *SBIm* interface Regarding telemetry, measurements are locally processed by specific KDD processes in the KDD module to reduce data exchange with the MDA controller. Note that telemetry measurements might be taken on a sub-second basis, so analysis is performed locally in the MDA agents and results can be conveyed to the MDA controller for decision making.

- The **MDA controller** collates monitoring data records from all the nodes in the network and stores them in an internal big-data repository. Such monitoring data is available for external systems through the IO4 interface. Data are analysed and machine learning algorithms are applied, which can trigger notifications suggesting actions to the SDN controller. The MDA controller includes the operational databases (e.g., topology and connections) retrieved from the SDN controller using the M-COM interface, which entails having a complete view of the network. With operational databases synchronized, the MDA controller is able to correlate them e.g., with the route of an LSP for failure localization purposes.

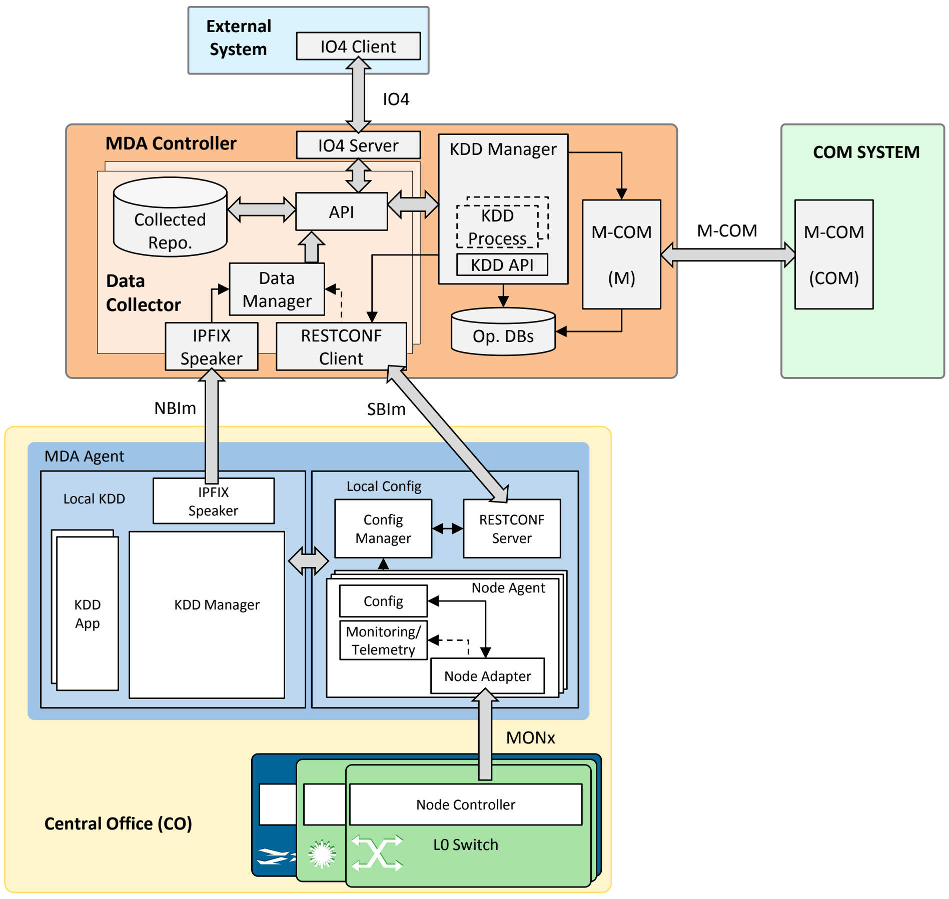


Figure 30. METRO-HAUL MDA architecture

3.7.2 Status

The MDA subsystem development phase has been successfully completed.

3.7.3 Related testbed and platform

The experiments have been carried out in the UPC's SYNERGY testbed. The scenario consists of an MDA controller and the ONOS SDN controller in the control and management plane and a number of network locations each with an MDA agent in charge of one local L2 switch, TP node, and L0 switch. MDA agents, as well as node controller emulators, have been developed in Python, where node controller emulators expose a NETCONF NBI.

3.7.4 Functional Tests

The tests carried out to validate the MDA subsystem consisted of implementing the basic workflows defined in Section 2.3.2: 1) *LSP set-up and monitoring / telemetry activation;* and 2) *degradation detection and network reconfiguration*.

3.7.5 KPIs

The identified KPIs are related to MH5. Fault/degradation detection time KPI, defined in D2.4.

KPI Description	No real-time degradation notification and node configuration tuning
Context	After the detection of a degradation by the MDA controller and the localization of the failure, it sends a recommendation to the SDN controller suggesting configuration changes. In this KPI, we assume that the scope of the changes is restricted to simple configuration tuning in a node.
Target	1 minute from the time an MDA agent sends a notification with the detection of a gradual degradation until the new configuration is received in the remote node.
Assessment	Workflow 2: time measured from messages 2 to 4, where recommended action was set to node configuration tuning.
Relationship to project KPIs	This KPI relates to the gradual degradation no real-time detection scenario described in KPI MH5 in D2.4.

KPI Description	No real-time degradation notification and network reconfiguration
Context	After the detection of a degradation by the MDA controller and the localization of the failure, the MDA sends a recommendation to the SDN controller suggesting configuration changes. In this KPI, the change consists of a network reconfiguration. For the sake of simplicity, we assume that such reconfiguration is re-routing one single optical connection avoiding the localized failure.
Target	10 minutes from the time an MDA agent sends a notification with the detection of a gradual degradation until the connection re-routing is completed.
Assessment	Workflow 2: time measured from messages 2 to 4, where recommended action was set to connection re-routing.
Relationship to project KPIs	This KPI relates to the gradual degradation no real-time detection scenario described in KPI MH5 in D2.4.

3.7.6 Availability

The MDA subsystem will be available at the METRO-HAUL GitLab Repository.

3.7.7 Interfaces

Interaction between the monitoring service and the rest of METRO-HAUL entities is enabled through the following interfaces (Section 4):

- *IO4*. Interface associated to external configuration of observation points (OPs) and data retrieval from other METRO-HAUL services.
- *M-COM*. Interface used to retrieve of operational databases and notify data-driven events from/to COM modules.
- *SBIm*. Interface between MDA agents and the MDA controller for network discovery and OP (de-)activation.
- *NBIm*. Interface between the MDA controller and MDA agents for data from network devices exportation.

- *MONp/MONo*. Interface for data collection from network devices.

3.8 Placement, planning and reconfiguration subsystem

3.8.1 Description

The Placement, Planning, and Reconfiguration Subsystem (namely, Network Planner) aims to optimize the network resources from two different perspectives: Off-line network design algorithms are mainly devoted to capacity planning both for green-field scenarios and partially deployed (i.e. brown field) deployments. Once network infrastructure is in production stages and operational, on-line resource allocation takes into account flows generated by end-user-oriented services that have different requirements in terms of bandwidth, delay and QoS. In this regard, the Network Planner enables the optimization of the resource allocation in the optical metro network to effectively provision VNFs in specific computing nodes considering heterogeneous requirements.

3.8.2 Sub-components

The Network Planner architecture, introduced in Section 3.6 of Deliverable 4.1 [D4.1] and extended in Section 2.4 of this document, is divided into front-end and back-end subcomponents as depicts Figure 31.

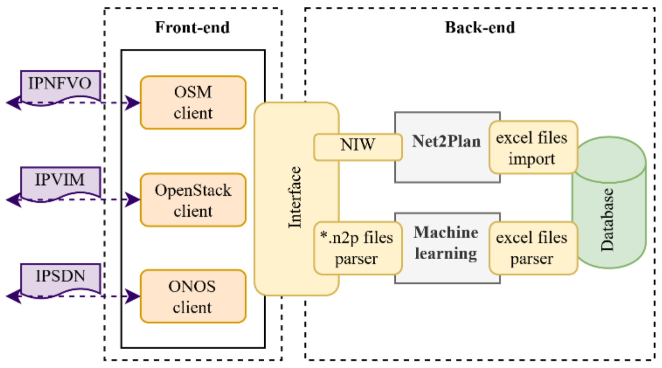


Figure 31: Placement, planning and reconfiguration system.

3.8.2.1 Front end

The Front-end contains the necessary clients to enable the interfaces that allow the exchange of information necessary for the planning tool (the Back-end) to carry out its operations, Figure 12 where the Network Planner interacts with the METRO-HAUL Control, Orchestration and Management (COM) system. The COM, responsible for the dynamic provisioning of services, interacts with the Network Planner via the IPNFVO, IPVIM, and IPSDN interfaces that provide the capability to query the Service Platform NFVO, the VIMs, and the WIM in order to plan the network properly and to provide network resource allocation and capacity planning solutions. Additional details are provided below. In line with the COM description in Section 2.4, the Network Planner works across-layers interfacing with SDN controllers at the (Network) Control Layer and WIM/VIM/NFVO elements at the MANO Layer. For instance, the Network Planner can assist the SDN controllers on path computation while providing indications to the VIM/OSM elements for the placement of VNFs.

3.8.2.2 Back-end

The back end consumes the information gathered from the Front-end to execute its algorithms with the eventual support/assistance of the ML. The Back-end block has been conceived as a very flexible environment that allows easy plug-in of a variety of computation algorithms implemented in heterogeneous software environments, provided that they are compatible with the overall architecture.

The front and Back-end modules exchange information by means of native Net2Plan *.n2p files, which are XML-based representations of network status. The Back-end module includes: a) a ML-based RWA algorithm, b) a ML-based VNF-placement algorithm, c) a ML-based algorithm integrating VNF placement and RWA. The Back-end module also includes a database to store any information consumed and produced by the algorithms (see right side of Figure 31). This permits the use of historical datasets that are particularly useful to train the ML algorithms.

3.8.3 Status

Regarding the status of the Front-end implementation tasks, the clients to use the IPSDN, IPNFVO, and IPVIM interfaces are currently implemented. Additionally, the usage of such clients is tested and presented in three demonstrations in International conferences [Mor18a], [Mor18b], and [Gar19a]. It is planned for the next stages to create a common framework in terms of a Java-Based Library which includes all the available clients to encapsulate all the Network Planner communication functionalities in the Front-end module. Moreover, the clients in this future library will be adapted to the NIW library to ease the usage in the Back-end module.

The implementation of the Network Planner interface with the Parent Controller/WIM is completed. The usage of the Rest/API related to such interface is presented in [Gar19b] and submitted in [Mor19]. The Network Planner-vertical interface is expected to be tested in next stages.

Regarding the implementation status of the Back-end module, the NFV over IP over WDM (NIW) library chosen to use in the Back-end module is available in the Net2plan planning tool since version 0.6.0. The algorithm to solve the network service /service chain allocation algorithm is ready and tested in [Mor18b] and [Gar19a]. The standalone VNF placement and the Network Resource and Wavelength Allocation (RWA) algorithms are currently under development. However, preliminary versions based on machine learning approaches are analysed in [Tro19].

3.8.4 Related testbed and platform

The platforms used to test the functionality of the placement, planning and reconfiguration subsystem are divided in two main testbeds.

3.8.4.1 Portable testbed

As shows Figure 32, the portable testbed is composed of a personal laptop, three high-performance mini-PCs and two regular auto-configured switches. In particular, the Net2Plan planning tool and the ETSI NFV Orchestrator OSM (in a virtual machine) run in the personal laptop. The two mini-PCs emulate AMEN/MCEN nodes in a metropolitan network with a VIM OpenStack instance in each of them. One mini-PC runs the ONOS SDN controller and a Mininet instance emulated metropolitan transport network. This portable platform was used in demonstrations [Mor18a], [Mor18b], and

[Gar19a]. This platform was used to test the NPNFVO, NPVIM, and NPSDN interfaces and a heuristic approach for solving the network service /service chain allocation problem.

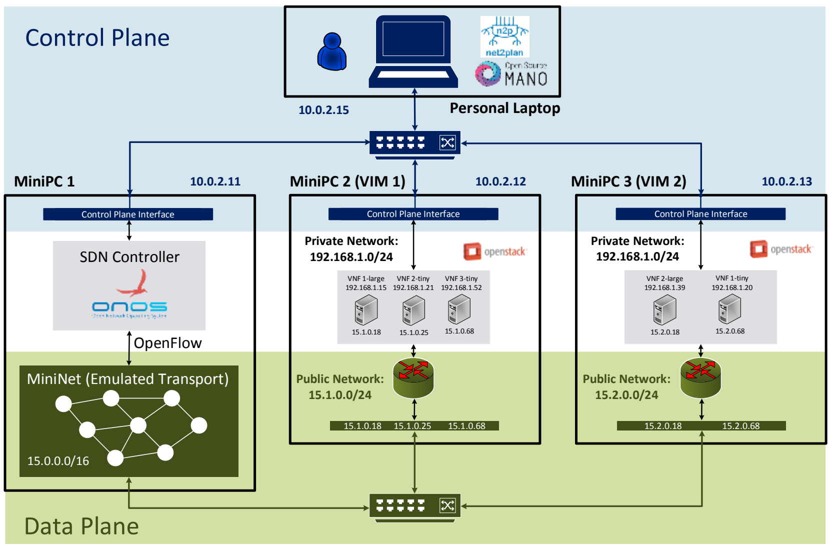


Figure 32: Portable testbed configuration.

3.8.4.2 Fixed Bristol-Cartagena testbed

The experimental setup to implement an SDN-NFV WAN architecture has been performed in two testbed islands, the two Net2plan instances (a GUI to emulate the vertical role and Tomcat-based server to provide Optimization-as-a-Service) were located in a laboratory in Cartagena (Spain) whilst the VIMs, the ETSI OSM, WIM, and SDN controllers and a multi-layer topology were placed in the High Performance Networks group facilities at the University of Bristol (UK). Both testbeds were connected by a private VPN to provide control plane visibility. The multi-layer infrastructure consists of two OpenStack datacentres interconnected by a combination of packet switches and OXCs, which are SDN-enabled and managed by a combination of SDN controllers as described in the functional schema exposed in Figure 33. This testbed was performed in [Mor19] to test the WIM/Parent Controller-Network Planner and Vertical-Network Planner Interfaces.

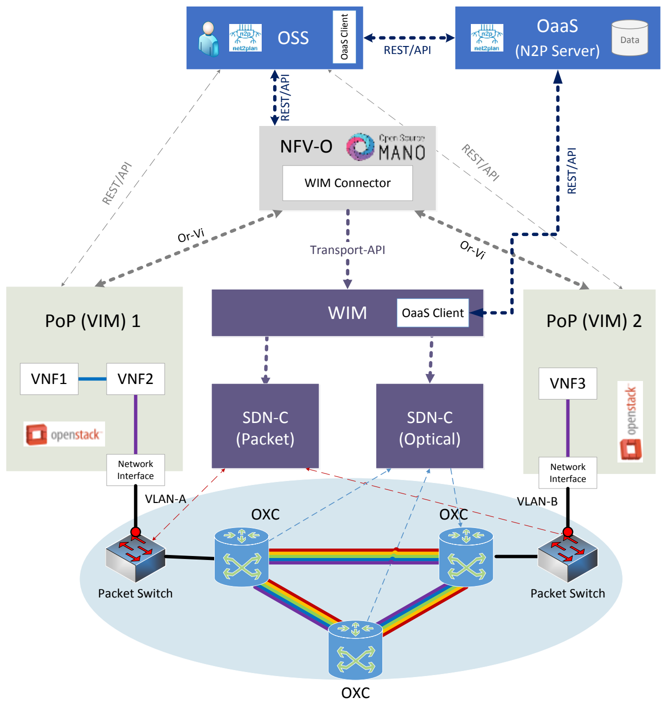


Figure 33: Fixed Bristol-Cartagena testbed configuration.

3.8.5 Functional Tests

As mentioned in the previous subsection, the portable testbed and the fixed Bristol-Cartagena testbed are used to test the following functionalities:

- Performance of the interfaces related to the network planner tasks: IPSDN, IPVIM, IPNFVO, Vertical-Network Planner, and Network Planner-Parent Controller/VIM. The Front-End clients and the network Optimization-as-a-Service Server and client are tested.
- Performance of the Network Planner’s Back-End algorithms: network service/service chain allocation, VNF placement algorithm, and Network Resource and Wavelength Allocation (RWA).

3.8.6 KPIs

The particular KPIs defined for the purposes of the placement, planning and reconfiguration subsystem is summarised in the next tables.

KPI Description	NP1: Execution time of each Back-end algorithms.
Context	The execution time of the algorithms depends on the code implementation and the size of the problem to be solved in terms of number of nodes of the topology, number of links, number of VNFs of the network service, VIMs occupation and network load.
Target	To be defined (< 20 ms?, It is depends on the size of the problem…)
Assessment	This is tested by measuring the execution time of the algorithm within the open-source Net2plan framework as part of the portable and the fixed Bristol-Cartagena testbeds.
Relationship to project KPIs	This KPI is related to the global METRO-HAUL KPI MH3. Optimizing the execution time of the algorithms are essential to minimize the set-up time of the network

	service slice.

KPI Description	NP2: Interfaces communication delay
Context	Augmenting the number of call procedures in the communication tasks used in the corresponding interfaces can result in increasing the end-to-end time in functional blocks communications.
Target	To be defined (< 5 ms? it depends on the communication channels…)
Assessment	This is tested by measuring the end-to-end time for all the network planner-related interfaces in the portable and the fixed Bristol-Cartagena testbeds.
Relationship to project KPIs	This KPI is related to the global METRO-HAUL KPI MH3. The end-to-end communication time between the functional blocks and the network planner is an essential part of the total set-up time of network service slices.

3.8.7 Availability

The source code of both network planner Front-end and Back-end modules will be available at the METRO-HAUL GitLab Repository.

3.8.8 Interfaces

The interfaces used or expected to be used by the placement, planning, and reconfiguration subsystem are:

- Interface Vertical Network Planner
- Interface Network Planner – WIM/Parent Controller
- Interface Network Planner – NFV Orchestrator, IPFVFVO
- Interface Network Planner – SDN Optical Controller, IPSDN
- Interface Network Planner – VIM, IPVIM

Further information about the interfaces is exposed in Section 4.

3.9 Service and traffic monitoring system

3.9.1 Description

This element is in charge of monitoring the system at the packet layer. It is composed of a set of network traffic probes watching which service is being provided by the underlying layers. The measurements obtained are made available to the monitoring and data analytics system, which can also request active measurements. These elements are needed to design a monitoring system with big-data analytics framework supporting cognition.

3.9.2 Sub-components

- Active Network Probe at 100 Gb/s.
- Passive Traffic Probe at 100 Gb/s.

3.9.3 Status

The development of an active probe at 100 Gb/s in this work package has followed the milestones below:

- M12: Initial designs;
- M18: First developments released;
- M21: Rest API to configure the probe.

Ongoing work:

- M24: Integration tests.
- M30: Migration from development board to production board.

3.9.4 Related testbed and platform

The probe is being validated in the Naudit 100G testbed and will be further validated in the Berlin Demo testbed, measuring the delays in the network. The integration with Berlin demo is expected for M30.

Joint work with UPC and Telefónica is currently being performed to test the functionality in a more integrated scenario, with a demo proposal submitted to ECOC 29019.

3.9.5 Functional Tests

1. The active probe receives measurement requests.
2. The active probe performs active measurements based on the request parameters. Measurements are done according to the defined KPIs.
3. The active probe returns the obtained measurements.
4. The obtained measurements are sensible with respect to the configured network path.

The following capabilities have been tested:

- Sending packet trains from the probe at 100 Gb/s.
- Receiving packet trains at the probe at 100 Gb/s.
- Measuring such packet trains (length, interarrival, sequence, errors, etc.) at line rate speed.

3.9.6 KPIs

KPI Description	Network path capacity
Context	It is useful to know what the achievable network path capacity is, and if an optical path has been established according to user-level specifications with respect to this KPI.
Target	Capacity in bits per second can be calculated, based on the number of bytes transmitted in a sequence of frames, and the time it took to receive them. The clock has a frequency of 300 MHz, so the measured time has a precision in the order of units of ns. It is expected that 100 Gb/s throughput can be measured.
Assessment	The active probe sends a packet sequence to another active probe at the other end. The packets are sent back to back at the source, and the interarrival time is measured at the destination, providing an estimation of the network path capacity.
Relationship to project KPIs	This KPI can help to measure the MH6 (Capacity of METRO-HAUL infrastructure) KPI and reduce the MH5 (Fault/degradation detection time).

KPI Description	Network Delay
Context	It is useful to know what the latency of the network is, and if an optical path has been established according to user-level specifications with respect to this KPI.

Target	Delay in ns can be measured. The clock has a frequency of 300 MHz, so the measured time has a precision in the order of units of ns.
Assessment	Packets in the sequence are timestamped, so it is possible to measure the RTT. OWD could also be measured with GPS receivers at the probes (not implemented right now).
Relationship to project KPIs	This KPI can help to measure the MH6 (Capacity of METRO-HAUL infrastructure) KPI and reduce the MH5 (Fault/degradation detection time).

KPI Description	Network Jitter
Context	It is useful to know the delay variation of the network, and if an optical path has been established according to user-level specifications with respect to this KPI.
Target	Delay in ns can be measured. The clock has a frequency of 300 MHz, so the measured time has a precision in the order of units of ns.
Assessment	Packets in the sequence are timestamped, so apart from delay, it is also possible to know the delay variation, using any of the existing jitter definitions.
Relationship to project KPIs	This KPI can help to measure the MH6 (Capacity of METRO-HAUL infrastructure) KPI and reduce the MH5 (Fault/degradation detection time).

KPI Description	Network Packet or Frame Loss
Context	it is useful to know what the PER of the network is, and if an optical path has been established according to user-level specifications with respect to this KPI.
Target	Packet sequences of up to 2^{32}-1 packets can be transmitted, so packet losses in very reliable links can be identified. Note, however that this measurement could take several minutes. Different BERT patterns can be used in the test to check also the BER and if the FEQ is working properly.
Assessment	Based on the number of packets in the sequence, we can calculate the fraction of packets that are not received from those that are transmitted at the sender. Accuracy will depend on the number of packets that are sent. For instance, to measure a 1% packet loss it will be necessary to send 1000 packets. The following BERT types have been implemented in the payload of the packet, so it can be checked if the FEQ is working properly: • PRBS (Pseudo Random Binary Sequence): binary sequence that is difficult to predict and exhibits statistical behaviour similar to a truly random sequence. • All zeros: A pattern composed of zeros only. • All ones: A pattern composed of ones only. This pattern causes the repeater to consume the maximum amount of power. • [1:7]: Also referred to as "1 in 8" has only a single one in an eight-bit repeating sequence. • [1:1]: A pattern composed of alternating ones and zeroes. • [2:8]: Pattern contains a maximum of four consecutive zeros and only two ones. • [3:24]: Pattern contains the longest string of 15 consecutive zeros with only three ones.

Relationship to project KPIs	This KPI can help to measure the MH6 (Capacity of METRO-HAUL infrastructure) KPI and reduce the MH5 (Fault/degradation detection time).

3.9.7 Availability

The hardware development is not publicly available at this moment, but an FPGA bitstream can be made available to other partners on request to be used for other project tasks.

The software development associated to the hardware can be released at the project Gitlab if necessary.

3.9.8 Interfaces

A REST interface has been implemented in order to be interoperable with the MDA controller.

3.9.9 Implementation details

This section dives deep into the hardware active probe. To start with, a Virtex Ultrascale+ FPGA device is being used. In particular, the VCU118 development board, which includes two 100 GbE interfaces, 8 GB of DDR4 memory, and a XCVU9P-L2FLGA2104E FPGA. The QSFP28+ physical cages are mapped directly to the FPGA. An integrated 100G Ethernet is in charge of connecting the FPGA side with the physical network. Moreover, a segmented LBUS interface is provided. An adaptor was inserted to convert from LBUS to AXI4-Stream, which is the de facto standard for streaming applications in the FPGA arena, and vice versa. The AXI4-Stream interface has 512-bit width of data and is clocked at 322 MHz (3.1 ns) to reach the needed throughput, even with the smallest frames.

The packet train technique [Ru16] has been implemented in the FPGA at 100 Gb/s, the development is split into two independent designs which could reside in the same FPGA. On one hand, the transmitting side, a synthetic packet generator has been developed. On the other hand, the receiving side is in charge of filtering the packets, analysing them and generating a summary. In what follows, each one of these elements is detailed.

3.9.9.1 Synthetic Packet Generator

This piece of hardware is written in Verilog and implemented as a Finite State Machine (FSM) in such a way that the behaviour is completely deterministic. This module is in charge of generating UDP packets that will carry useful information for the measurement. Therefore, users can configure some of the fields in the packet such as source and destination IP addresses, source and destination ports, packet size, and Bit Error Rate Test (BERT) type. Moreover, each UDP packet carries a payload with a timestamp, identifier, and burst number, that are iperf compatible. As well as that, users can configure the number of packets in the burst, the content of the payload based on the BERT type, and the inter packet gap. All those configurations are done through an AXI4-Lite interface from a program running in the computer hosting the FPGA card. Every burst could have different parameters. This module generates packets at the maximum throughput, which is extremely close to the theoretical value in 100G Ethernet links.

The UDP payload content is structured as follows (see Figure 34):

- Local Timestamp (8 bytes)
- Extended Identifier (4 bytes)

- Total amount of packets configured by the user (4 bytes)
- Burst ID (2 bytes)
- 4 free bytes reserved for future use.
- BERT payload.

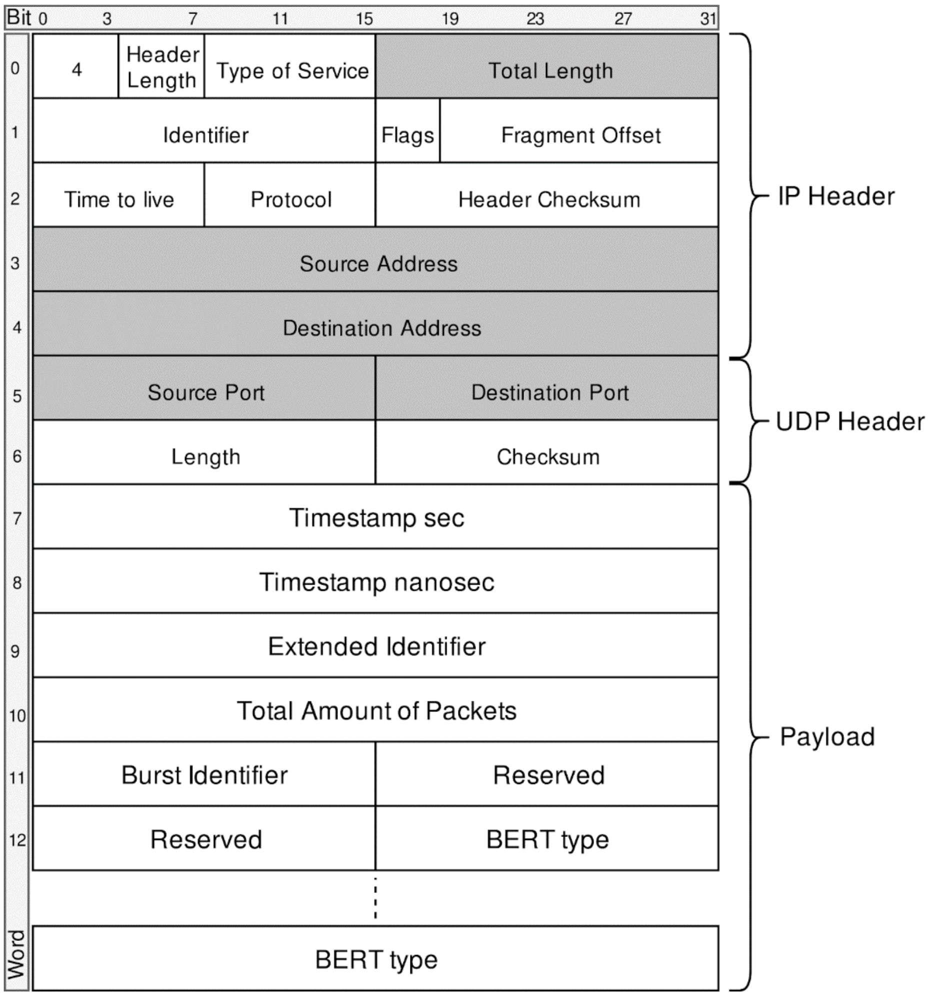


Figure 34. Packet content

3.9.9.2 Packet Filter + IPFIX

In the receiver side, packets are timestamped at the moment they reach the FPGA side. The packet handler module is in charge of filtering packets by type. For instance, ARP and UDP packets are taken into account in this implementation. After that, UDP packets are inspected and those fields that are configurable are verified in order to check if the packet matches with the user configuration. If so, the useful information is extracted to compute the packet train parameters. This information is fed to the statistics generator, which collects it. The results of the burst can be exported as a summary using a message following the IPFIX format. Every burst generates at least one IPFIX message. However, depending on packet loss, more than one summary could be generated. There are three ways of generating the summary: a) at the end of a burst; b) discontinuous incremental identifier; and c) timeout, when after a certain amount of time no packet is received, the summary is generated regardless of the number of received packets.

The format of the IPFIX template is presented in Table 1:

Table 1. IPFIX fields used to export active measurements

TYPE	NAME	IANA'S ELEMENT ID
unsigned32	Src_IP	8
unsigned32	Dst_IP	12
unsigned16	Src_port	7
unsigned16	Dst_port	11
unsigned64	PacketDeltaCount	2
unsigned64	PacketTotalCount	86
unsigned64	OctetDeltaCount	1
unsigned16	EthLength	242
unsigned32	Jitter	387
dateTimeNanoseconds	Start_Timestamp	156
dateTimeNanoseconds	End_Timestamp	157
unsigned64	Rtt	Custom
unsigned16	Burst_ID	Custom
unsigned16	Group_ID	Custom

3.10 De-fragmentation app

As described in Section 2.5.2 the de-fragmentation app is composed of two main components: the de-fragmentation tool and a set of REST APIs that extends the features of the Optical SDN controller.

In turn, the de-fragmentation tool architecture is based on three main components: the de-fragmentation server (Figure 35-A) coordinating the defragmentation procedures, a web page (Figure 35-B) used to collect the inputs from the users, and the defragmentation engine (including all the executable code and a set of API interfaces).

The overall software architecture of the defragmentation app follows the scheme shown in Figure 35 and described hereafter:

- Defragmentation engine (Figure 35-A) implements the de-fragmentation routine coded in C++. This routine takes as input the current network state and provides as output a list of actions (i.e., reconfiguration of established lightpaths) to reduce the defragmentation. The routine interfaces with the de-fragmentation webserver (Figure 35-B) through a .jar executable that shows the Java API.
- The De-fragmentation Server (Figure 35-B) reads the details about currently established lightpaths (i.e., including utilized paths and OChs, flow 2 in Figure 35) through the ONOS core REST APIs the underlay network topology (i.e., list of devices and links, flow 1 in Figure 35), and through the extended ONOS optical-rest application . It then provides the network description to the defragmentation engine (flow 4 in Figure 35).
- Once that the defragmentation routine is executed within the engine, any result (either the new spectrum allocation if the process is successful, or an error message in case of impossibility of reconfiguration) is returned to the server through Java APIs provided by the engine itself (flow 5 in Figure 35).
- The user can analyse the status of the network after defragmentation and check the new optical connection set-up through the Defragmentation Web Page (Figure 35-C). This Web

Page allows the user to set some reconfiguration conditions, such as the time frequency between two defragmentation operations, or the maximum acceptable network fragmentation threshold. As an example, the user can define a maximum fragmentation value that can be tolerated in the network, if this value is surpassed a defragmentation reconfiguration is automatically run. These settings are provided by the user and taken as input from the defragmentation server through flow 3 in Figure 35. The network fragmentation threshold is then passed from the server to the Defragmentation engine so that a defragmentation operation starts only when the given threshold is surpassed. The information flow 7 in Figure 35 allows to graphically show the defragmentation outputs on the Defragmentation Web Page (e.g., the new link fragmentation values, the new spectrum allocation of optical connections, and so on).

- The Optical SDN controller is involved in the procedure in two phases. First, it is used to retrieve network and lightpaths information (i.e., flows 1 and 2) through GET commands. Then it is used to deploy the required re-configurations on the network through POST and DELETE commands (i.e., flow 6 in Figure 35).

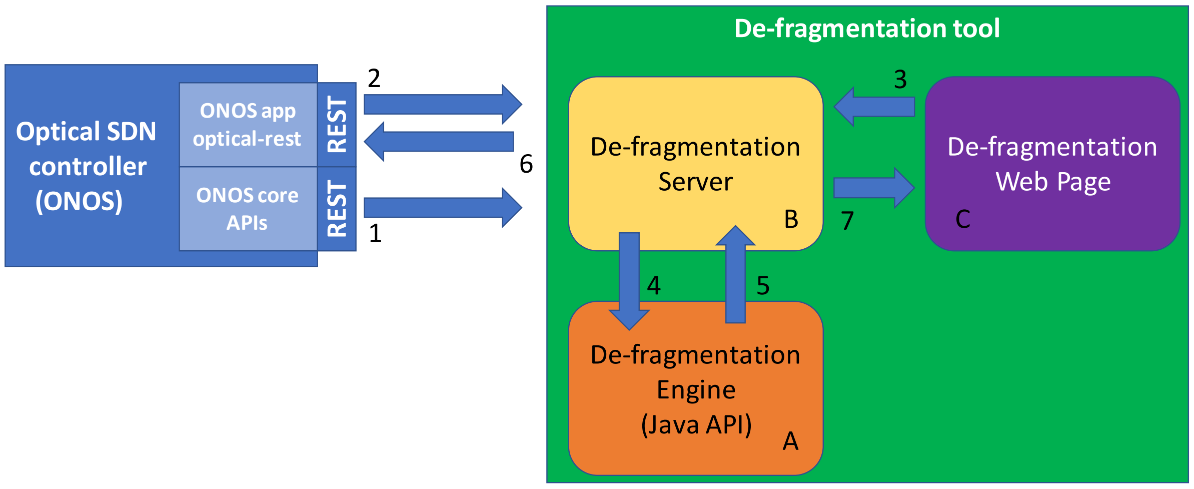


Figure 35. Overall architecture and data flows for the de-fragmentation procedure

3.11 SDN application for proactive soft-failure detection

3.11.1 Description

The fault detection component is extended with laser degradation failure mode detection. Laser degradation analysis is a crucial process for the enhancement of laser reliability. We implemented a data-driven fault detection approach based on Long Short-Term Memory (LSTM) to detect the different laser degradation modes based on synthetic historical failure data, attaining classification accuracy of ~95%.

3.11.2 Sub-components

- Database and Parser (SQL DB on ML server)
- Learning component
- LSTM based ML engine

3.11.3 Status

We targeted different laser failure modes: namely gradual, rapid, and sudden degradation as well as for normal laser operation. The output features, including optical power P, the threshold current I_0 and temperature T, are extracted from real laser datasheets specifications, whereas the underlying coefficients to create these features are generated using normal distributions shown in Figure 36.

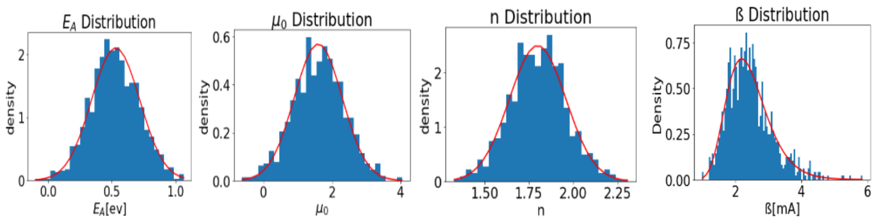


Figure 36. Coefficients to create output features

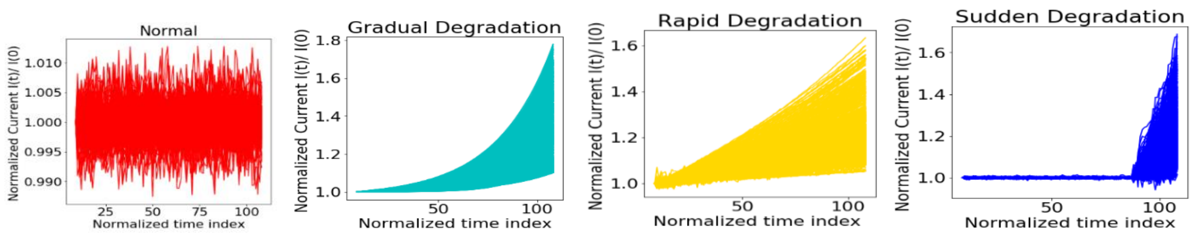


Figure 37 shows the different laser failure modes patterns after pre-processing, where the x-axis represents different time indexes. Normal is category 0, gradual is category 1, rapid is category 2, and sudden is category 3.

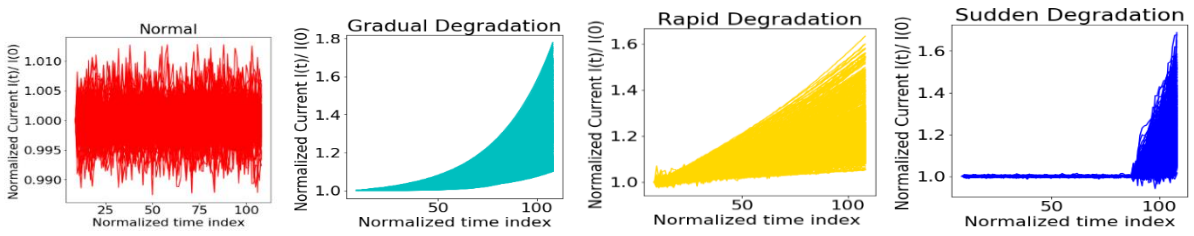


Figure 37. Laser failure modes patterns

Our preliminary analysis suggests that the LSTM-based model was the best model in terms of all the evaluation metrics as expected because LSTM is able to learn long term dependency in a temporal pattern. Multi-class logistic regression and "random forest" performed as second and third best models, whereas K-nearest neighbours had the worst accuracy.

Roadmap:

Core prototype design and development using commercial hardware has already been completed. We will continue adding fault management functionalities.

3.11.4 Related testbed and platform

The application is developed on ADVA premises using the in-house lab and compute infrastructure.

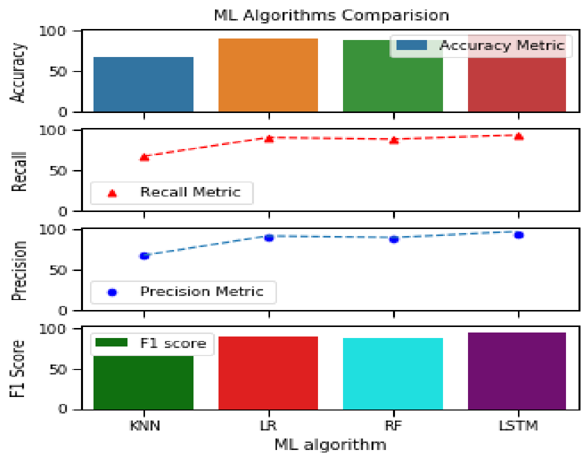


Figure 38. Laser failure modes patterns

3.11.5 Functional Tests

How to measure: This stage is carried out in ML model development stage (offline). Figure 39 reports the confusion matrix (1.00 represents 100% accurate classification), where it can be shown that class 2 is sometimes misclassified as class 1, and class 3 is misclassified as class 1, whereas other classes are correctly identified.

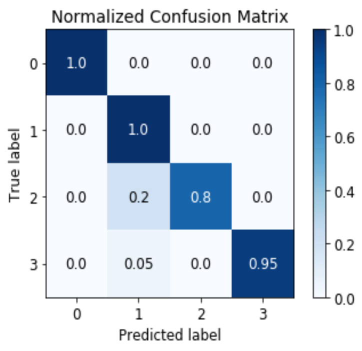


Figure 39. LSTM Confusion matrix with normalized classification percentages.

Note the data is based on 6,000 unique laser behaviour responses, of which 20% are used for model evaluation and KPI generation, a standard ML approach. For the interested reader, further KPIs may be derived from the figure presented in the *Status* section, and include Precision, Recall, and F1 score.

3.11.6 KPIs

KPI Description	Correct classification rates of failures
Context	Reliability is an important aspect of the developed application
Target	The classification accuracy is ~95%. Note that the gradual and rapid degradations may sometimes be misclassified (20%), whereas gradual and sudden may also be misclassified by ~5% due to partial pattern match. This is expected behaviour due to overall performance generalization across failure modes.
Assessment	K. Abdelli, D. Rafique, S. Pachnicke, "Machine Learning for Laser Failure Mode Detection," ICTON (2019)

Relationship to project KPIs	The measured KPI relates to MH5 (Fault/degradation detection time) KPI, and details fault degradation detection rate.

3.11.7 Availability

The SDN Application for proactive soft-failure detection is made available to use cases (RESTAPI).

4 Interfaces

This section is devoted to describing every interface defined between the components described in the previous section. The interfaces are described in terms of the functions that they support and how such functions have been implemented.

4.1 Operator Network Planner

This interface oversees the Vertical (Operator) and Network Planner procedures. This interface is used by the operator to inform the Network Planner about the requirements of the network service to be deployed. In the Network Planner side, the information coming from the operator related to the network service is received is used to create and complete an ETSI-based Network Service Descriptor (NSD) ready to be deployed in the NFV Orchestrator (OSM). This interface is added to this document after choosing the less-invasive option exposed in the section 4.1.6 of the previous deliverable [D4.1].

To this purpose, an Optimization-as-a-Service (OaaS)-based server is implemented in a Tomcat server which permits the deployment and execution of JAVA-based applications (.war) that includes the Back-End algorithms. To request service chain optimal instantiation, an OaaS-based client must be used in the Vertical side. Further explanation of the Network OaaS server and interface definition is provided in [Gar19b].

Function	**Implemented interface**
Authentication	This functionality is implemented as follows: • The vertical OaaS client requests a token to establish a session. This function is accomplished by requesting an authentication POST call.
Network service management	This functionality is implemented as follows: • The vertical OaaS client requests for the execution of an algorithm to solve the VNF placement problem. In this execution POST call, a JSON file is included with the Network service requirements, such as, origin and destination nodes, latency or bandwidth. The server receives the JSON, executes the corresponding algorithm, and stores the results in a database. • The network Planner prepares an NSD ready to be in instantiated in the NFV Orchestrator with the VNFs placement locations. Additionally, if necessary, the optimal path in the transport network associated to the network service instantiation is also stored with a unique NSI ID in the database to be requested by the Parent Controller/WIM.

4.2 Net2plan to Parent Controller/WIM

In a similar way to the interface defined in section 4.1, this interface uses the concept of network Optimization as a Service. This interface will be used in the cases where the Parent Controller/WIM wants to request the Network Planner about an optimal path in the transport network associated to a Network Service Instantiation. In this case, an OaaS client must be deployed in the Parent Controller/WIM module.

Function	**Implemented interface**
Authentication	This functionality is implemented as follows: • The Parent Controller/WIM OaaS client requests a token to establish a session. This function is accomplished by requesting an authentication POST call.
Network Service Instantiation (NSI) path request	This functionality is implemented as follows: • The vertical OaaS client queries the OaaS module for the previously calculated transport path stored in the database for a certain NSI. • The Network Planner OaaS server retrieves the path solution and sends it to the vertical client in a JSON file including the origin and destination nodes and the ordered set of links to be traverse for the service chain.

4.3 Network Planner IPSDN, IPNFVO, and IPVIM

The Front-end includes a series of client modules to exchange information with the software components of the COM architecture and consume it in the Network Planner framework. This information involves the network topology in terms of network nodes, links, and traffic engineering attributes, such as occupied optical frequency slots or additive metrics, as well as the status of the different hypervisors in computed nodes of the infrastructure.

In particular, the exchange of information is performed exploiting three interfaces:

4.3.1 Interface Planner – NFV-O (IPNFVO)

IPNFVO aims to provision network services composed as an ordered sequence of VNFs through network service descriptors (NSD), which are the information models that represent network services in Open Source MANO (OSM) terminology. The IPNFVO interface uses the OSM native REST/API and a Java-based OSM client is developed within the Front-end module in the network planner to enable the connectivity between the network planner and the NFV orchestrator.

Function	**Implemented interface**
Service chain characteristics	The network service slice features defined in the NSD are: • Start/end points • Type and number of VNFs • Virtual networks in VIMs to allocate the virtual machines

	of the VNFs • Type of Interconnections (forwarding rules between VNFs of the same Service Chains) • Specific requirements of the virtual link (data rate, tolerated latency) • Cost to instantiate a new VNF • Disjoint placement of active/standby VNFs on physically separated machines • VNF Hardware accelerations requirement (hardware acceleration mechanisms)
NVF control	The OSM REST/API will be used or adapted for that purpose. The NFV-O provides information about virtual resources to the planning tool. Providing IT resources information coming from the VIMs through this interface is under investigation. The control of the VNF VMs is handled by management IPs, assigned from OpenStack floating IP pools or fixed IP addresses within certain virtual networks in the VIMs. This option is configured in the NSD.
NFV Instantiation Features	The results of the execution of the optimal algorithm will be sent by the network planner's Front-end module to the NFV-O. The placement of the VNFs to be instantiated in the VIMs are determined by the results of the network planner algorithms and they are notice to OSM as additional attributes in the query for instantiation of the NS.

4.3.2 Interface Planner – SDN controllers (IPSDN)

IPSDN aims to gather information from the transport infrastructure and provides transport network resource allocation in line with the requirements of the network services. Recent demonstrations relevant for IPNFVO and IPSDN reported the Planning Tool Net2Plan assisting an OSM instance in the optimal allocation and instantiation of network services (NSs) in a simulated transport network [Mor18a]. Subsequently, [Mor18b] demonstrated an ONOS controller emulating the packet-layer network and considered the end-to-end latency requirements to perform flow allocation jointly with the NS allocation and VNF instantiation via OSM.

In order to permit communication between the SDN optical controller and the network planner, a specific client coded in Java has been developed in the Front-end module of the network planner based on the native ONOS REST/API.

Function	**Implemented interface**
Topology Information	The implementation of the IPSDN permits the network planner to retrieve the following information regarding the topology necessary for the execution of the Back-end algorithms: • Forwarding nodes and NFV-capable Nodes

	• Transmission delays of physical nodes (e.g., L2/L1 switches) • Physical links in the optical domain and packet domain • Bandwidth resources • Propagation delay
Monitoring data management	The implementation based on the native ONOS REST/API allows retrieving information about monitoring data and management of the network: • Nodes connectivity status • Links load status • Failures alert

4.3.3 Interface Planner - VIM (IPVIM)

IPVIM interacts with the IT resource manager to enable multiple functionalities inherent in the NFV technology. In particular, a recent demonstration reported an open-source Net2Plan extension for interfacing multiple OpenStack instances, which enables multi-tenant slicing, IT resource visualization, and VM migration [Gar19].

The implementation of the Front-end module contains a Java-coded client to enable the IPVIM between the network planner and the VIMs. This OpenStack client uses the by-default functionalities of the native OpenStack client exploited by the open source library OpenStack4J.

Function	**Implemented interface**
Compute/storage/networking resource requirements	The IPVIM exploits the OpenStack REST/API to obtain the telemetry service (Gnocchi) the available list of resources that generate metrics and measures. • VNF resources requirements (CPU cores, RAM, storage) • Scaling constraints for the VMs • VIM instances real-time status
VIM control	Regarding the VIM control, the current implementation of the Front-end OpenStack Client permits: • Multitenant IT slicing with control of the IT infrastructure by different carriers and according to a set of quotas. • Intra-cluster VM migration which permits multi-node OpenStack cluster instantiation. Additionally, the IT resource manager to assess the correct instantiation of the VMs.

4.4 OSM to Parent controller

The interface from OSM to parent controller is the Or-Wi interface. Here, the parent controller refers to the WIM. In Section 3.6.2, the functionality of the WIM connector for OSM was discussed. In this section, the detailed implementation of the WIM connector is presented: where the WIM connector is used to contact the relevant WIM to interconnect PoPs hosting VNFs. Currently, the WIM connector implementation is based on the ELINE. Some of the functions of the abstract WIM connector class in OSM are described in Table 2.

Table 2: WIM Connector class functions

Function	**Description**
Init	Initializes the WIM class which includes adding the relevant WIM connectivity information and access credentials.
create_connectivity_service	Requests the WIM to establish WAN connectivity between the specified WAN endpoints. The WIM sends a UUID along with some additional information about the connection.
get_connectivity_service_status	Monitor the status of the connectivity service established using the UUID of the connection
edit_connectivity_service_status	Change an existing connectivity service using its UUID.
delete_connectivity_service	Requests WIM to terminate a WAN connectivity using its UUID.
get_all_active_connectivity_services	Gets all active connections provisioned by a WIM.

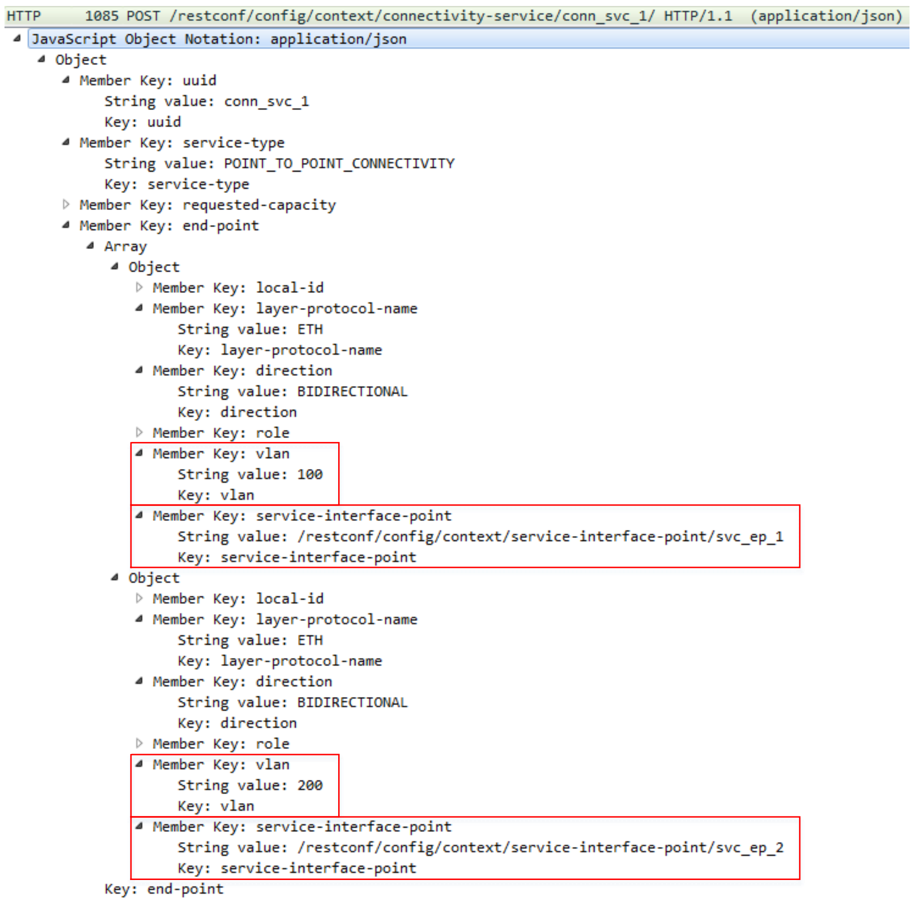
Based on these functions, TAPI and L2SM based two WIM connectors are proposed for OSM.

4.4.1 TAPI based WIM connector

As shown in Section 3.5.2.1, a proof of concept TAPI based WIM connector has been implemented which has the capability of only creating a WAN interconnection between two TAPI service interface points (SIPs). With the reference scenario of Figure 26, the port mapping is added to OSM which is the mapping of the VIM to a TAPI SIP. Once the NS is instantiated, OSM waits for the VLAN information at either VIM. This VLAN information along with the TAPI SIPs is sent to the TAPI based WIM connector. The TAPI *connectivity_service* API is accessed from the function is called from the *create_connectivity_service* function in the WIM connector class. The message that is sent as part of the API is shown in Figure 40. The endpoints in the message are augmented with the value of the VLAN corresponding to the VIMs. The connectivity service is requested based on Ethernet based SIPs.

4.4.2 OSM to Parent Controller via L2SM

The L2SM WIM plugin allows connecting the OSM with the Parent Controller to handle the lifecycle of a layer 2 VPN service, request creation, modification and deletion, by utilizing the service delivery model defined in RFC 8466. The service delivery model provides the YANG model. The RESTCONF paradigm with JSON encoding will be used for the communication OSM-Parent Parent Controller. In this section, the different operations are detailed.



```
HTTP      1085 POST /restconf/config/context/connectivity-service/conn_svc_1/ HTTP/1.1  (application/json)
◢ JavaScript Object Notation: application/json
  ◢ Object
    ◢ Member Key: uuid
        String value: conn_svc_1
        Key: uuid
    ◢ Member Key: service-type
        String value: POINT_TO_POINT_CONNECTIVITY
        Key: service-type
    ▷ Member Key: requested-capacity
    ◢ Member Key: end-point
      ◢ Array
        ◢ Object
          ▷ Member Key: local-id
          ◢ Member Key: layer-protocol-name
              String value: ETH
              Key: layer-protocol-name
          ◢ Member Key: direction
              String value: BIDIRECTIONAL
              Key: direction
          ▷ Member Key: role
          ◢ Member Key: vlan
              String value: 100
              Key: vlan
          ◢ Member Key: service-interface-point
              String value: /restconf/config/context/service-interface-point/svc_ep_1
              Key: service-interface-point
        ◢ Object
          ▷ Member Key: local-id
          ◢ Member Key: layer-protocol-name
              String value: ETH
              Key: layer-protocol-name
          ◢ Member Key: direction
              String value: BIDIRECTIONAL
              Key: direction
          ▷ Member Key: role
          ◢ Member Key: vlan
              String value: 200
              Key: vlan
          ◢ Member Key: service-interface-point
              String value: /restconf/config/context/service-interface-point/svc_ep_2
              Key: service-interface-point
        Key: end-point
```

Figure 40: TAPI create connection message from OSM to TAPI based WIM

4.4.2.1 Request Connectivity service creation

Using the **create_connectivity_service** method, OSM can request the creation a service between two given connection points. For each connection point, the plugin needs to know in advance the site and the bearer reference, that is which exact link will be used. Using this method, OSM can select if the connection will use encapsulation (VLAN) or not. The workflow for creating the service is represented in Figure 41.

The creation of a L2 VPN service in the Parent controller consists in three calls from OSM. The first one (OSM_PC_L2SM_1) is to request the creation of the VPN Service using the POST method. The WIM plugin will generate the UUID for each new L2 VPN service that is being requested. The next operations (OSM_PC_L2SM_2) are the creation of a site_network_acces over existing sites (one operation per end point). In the creation of the site_network_access, all information of the connection (e.g. VLAN) is provided. If the operation is successful, the VLAN will be configured for the first site and it will be attached to the VPN. If this operation is done without problems, then the same operation will be made for the second site. The next tables show how these petitions work.

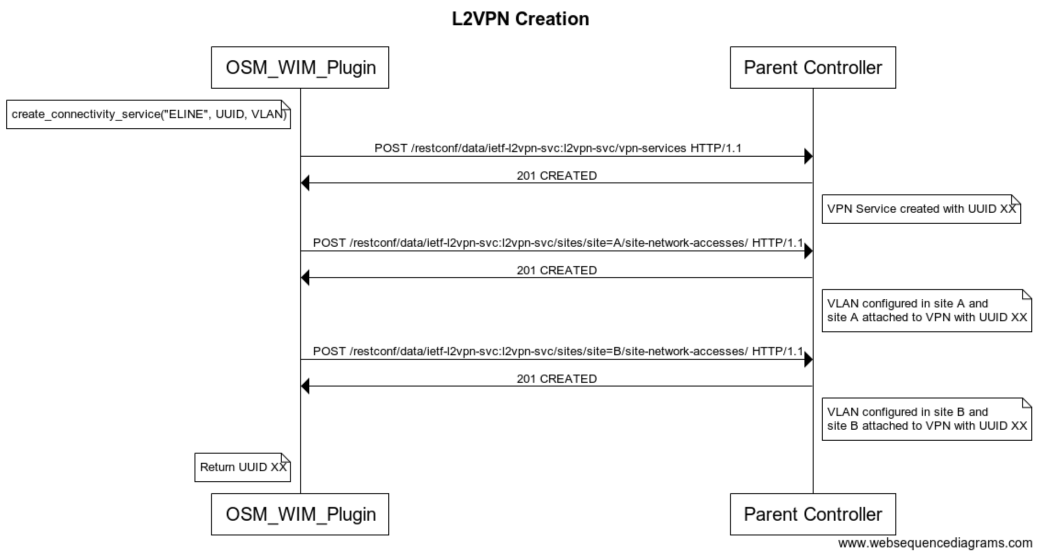

L2VPN Creation
OSM_WIM_Plugin
Parent Controller
create_connectivity_service("ELINE", UUID, VLAN)
POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services HTTP/1.1
201 CREATED
VPN Service created with UUID XX
POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=A/site-network-accesses/ HTTP/1.1
201 CREATED
VLAN configured in site A and site A attached to VPN with UUID XX
POST /restconf/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=B/site-network-accesses/ HTTP/1.1
201 CREATED
VLAN configured in site B and site B attached to VPN with UUID XX
Return UUID XX
OSM_WIM_Plugin
Parent Controller
www.websequencediagrams.com

Figure 41: L2VPN creation paradigm

Table 3. OSM_PC_L2SM_1

Description	Creation of L2 VPN Service in parent controller
URL	/RESTCONF/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services
Method	POST
Inputs	Information of the vpn-services in the JSON object: • **svc-topo**: Used to identify the type of VPN service topology that is required. • **vpn-svc-type:** defines the service type for provider-provisioned L2VPNs. • **customer-name:** Used to identify the customer • **vpn-id:** VPN UUID.
Outputs	The next HTTP codes can be received from the Parent Controller: • **201:** OK – The service has been created. • **408:** Timeout. • **409:** The service already exists. (uuid exists)

Example of the JSON object	`{`
	`'vpn-service': [{`
	`'svc-topo': 'any-to-any',`
	`'vpn-svc-type': 'vpws',`
	`'customer-name': 'osm',`
	` 'vpn-id': '9ce50dea-5eeb-412f-ab0f-fcb5c76ec06d'`
	`}`
	`}`

Table 4. OSM_PC_L2SM_2

Description	VLAN configuration and VPN attachment for a site
URL	/RESTCONF/data/ietf-l2vpn-svc:l2vpn-svc/sites/site=A/site-network-accesses/
Method	POST

Inputs	Information of the site-network-accesses in the JSON object: • **vpn-id:** VPN UUID. • **site-role:** defines the role of the site in a particular VPN topology. • **network-access-id:** ID generated for the new network -access of the VPN The next parameters (connection) are only filled if the service is required to be with encapsulation: • **encapsulation-type:** allows the user to select between Ethernet encapsulation (port-based) or Ethernet VLAN encapsulation (VLAN-based). • **cvlan-id:** contains the ID of the VLAN.
Outputs	The next HTTP codes can be received from the Parent Controller: • **201:** OK – The service has been created. • **408:** Timeout. • **409:** The service already exists.
Example of the JSON object (Service with encapsulation)	`{ 'site-network-access': [{ 'vpn-attachment': { 'vpn-id': '9bef07aa-017c-4bd1-966d-a1dcda0318a5', 'site-role': 'any-to-any-role' }, 'network-access-id': 'a9a62e57-e7a8-48b4-bbbf-d8776fa490d7', 'connection': { 'encapsulation-type': 'dot1q-vlan-tagged', 'tagged-interface': { 'dot1q-vlan-tagged': { 'cvlan-id': 22 } } } }] }`

4.4.2.2 Requests to delete a service from OSM to Parent Controller

The WIM plugin also has the capability of deleting an already created service (**delete_connectivity_service method)** using its UUID. The method uses the OSM_PC_L2SM_3 operation, which is based on an HTTP DELETE call:

Table 5. OSM_PC_L2SM_3

Description	Delete a Service in Parent Controller
URL	/RESTCONF/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services/vpn-service={}/
Method	DELETE
Inputs	**Service UUID**: to identify the service the user wants to delete.
Outputs	The next HTTP codes can be received from the Parent Controller: • **204:** NONE – The service has been deleted. • **408:** Timeout.

4.4.2.3 Delete all services from OSM to Parent Controller

There is the possibility to delete all the services created from OSM (**clear_all_connectivity_services function**) with only a call to the Parent Controller from OSM**.** The WIM Connector communicates with the parent controller using the OSM_PC_L2SM_4 operation detailed bellow:

Table 6. OSM_PC_L2SM_4

Description	Delete all connectivity services
URL	/RESTCONF/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services
Method	DELETE
Inputs	<NONE>
Outputs	The next HTTP codes can be received from the Parent Controller: • **204:** NONE – All services has been deleted. • **408:** Timeout.

4.4.2.4 Get the status of a service from OSM to Parent Controller

The WIM Connector function **get_connectivity_service_status** provides knowledge of the status of an existing service, with an operation providing the UUID of the desired service. The function retrieves the information from the parent controller using the OSM_PC_L2SM_5 operation detailed bellow:

Table 7. OSM_PC_L2SM_5

Description	Get status of a service
URL	/RESTCONF/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services/vpn-service={}/
Method	GET
Inputs	**Service UUID**: to identify the desired service.
Outputs	The next HTTP codes can be received from the Parent Controller: • **200:** The service exists. • **408:** Timeout.

4.4.2.5 Get all services from OSM to SDTN

A function is also available in the WIM Connector for getting all the existing services: **get_all_active_connectivity_services.** With just the OSM_PC_L2SM_6 operation, the L2SM WIM Plugin can get all the created services.

Table 8. OSM_PC_L2SM_6

Description	Get all existing services
URL	/RESTCONF/data/ietf-l2vpn-svc:l2vpn-svc/vpn-services
Method	GET
Inputs	<NONE>
Outputs	The next HTTP codes can be received from the Parent Controller: • **201:** OK – Recovers all the existing services. • **408:** Timeout.

4.4.2.6 Edit a service from OSM to Parent Controller

The WIM Connector function **edit_connectivity_service** allows to modify all the parameters of an existing service. For this aim, two calls are made to modify both endpoints with an HTTP PUT method. A JSON object with all the new parameters also should be provided.

Using this method is also possible to transform a non-encapsulated L2VPN into a new VLAN or vice-versa. The next table gives the information about a PUT call to the Parent Controller, but two should be made in the same way.

Table 9. OSM_PC_L2SM_7

Description	Creation of L2 VPN Service in parent controller
URL	/RESTCONF/data/ietf-l2vpn-svc:l2vpn-svc/sites/site={}/site-network-accesses/
Method	PUT
Inputs	Function parameters: • **Service_uuid:** UUID of the desired service to modify. • **Conn_info:** contains information about the endpoints to modify. Information of the vpn-services in the JSON object: • **vpn-id:** VPN UUID. • **site-role:** defines the role of the site in a particular VPN topology. • **network-access-id:** ID generated for the new network -access of the VPN The next parameters (connection) are only filled in if the service is required to be with encapsulation: • **encapsulation-type:** allows the user to select between Ethernet encapsulation (port-based) or Ethernet VLAN encapsulation (VLAN-based). • **cvlan-id:** contains the ID of the VLAN. Information of the end points (conn_info): • **site-network-access-id:** Network_access_id of the desired endpoint connected to the given VPN service.
Outputs	The next HTTP codes can be received from the Parent Controller: • **201:** OK – The service has been created. • **408:** Timeout. • **400:** The service does not exist.

Example of the JSON object	`{`
	`'site-network-access': [{`
	`  'vpn-attachment': {`
	`    'vpn-id': '9445cdff-cb56-40bc-8f4b-a73603d21283',`
	`    'site-role': 'any-to-any-role'},`
	`    'network-access-id': '7dbd671b-e00f-42ff-a573-0415adc9f81e',`

	`'connection': {}`
	`}]`
	`}`

4.5 Parent controller to optical SDN

The need to orchestrate multiple technologies is a key METRO-HAUL requirement. Most SDN Controllers offer proprietary interfaces (or, at best, open yet no standardized interfaces) to applications (or, more generically, high level controllers or other functional layers referred to as Network Orchestrators), and such SDN controllers are arranged following an approach commonly referred to as “vendor domains or islands”. This heterogeneity, due to having different controller interfaces in a multi-domain context, forces the use of “plugins” and it is difficult and expensive to extend (with the so-called umbrella management systems, used by the operators to deploy services spanning multiple domains). As a driving motivation and clear problem statement, there is a need for a standard interface, with common models, to act as a controller NBI.

The Transport API (TAPI) [TAPI] published by the ONF meets the main requirements to be a protocol and interface used between an orchestrator and multiple domain controllers. A TAPI based interface offers multiple services; let us just mention the main key ones, the topology and connectivity. The services are modelled in the YANG modelling language.

Common Context. The TAPI context is the shared information between a TAPI client (user) and the TAPI server (SDN controller). The model defines a TAPI domain as being able to provide services between Service Interface Points (or SIPs) mainly characterized by their universally unique identifiers (UUIDs). A basic operation for a client is to “retrieve” the context in order to obtain the list of SIPs, so connectivity services are requested between two (or more) exported SIPs.

Topology context and models. If a given TAPI server supports a topology model, it augments the TAPI shared context with a (list of) topologies. Each topology is composed of a list of nodes, which, in turn, has Node Edge Points (NEPs). Links connect two NEPs. The model is flexible enough to support recursive topologies and different levels of abstraction. The level of detail exported is configurable by policy. A client is thus able to obtain an (abstracted) view of the topology and map TAPI SIPs to external NEPs.

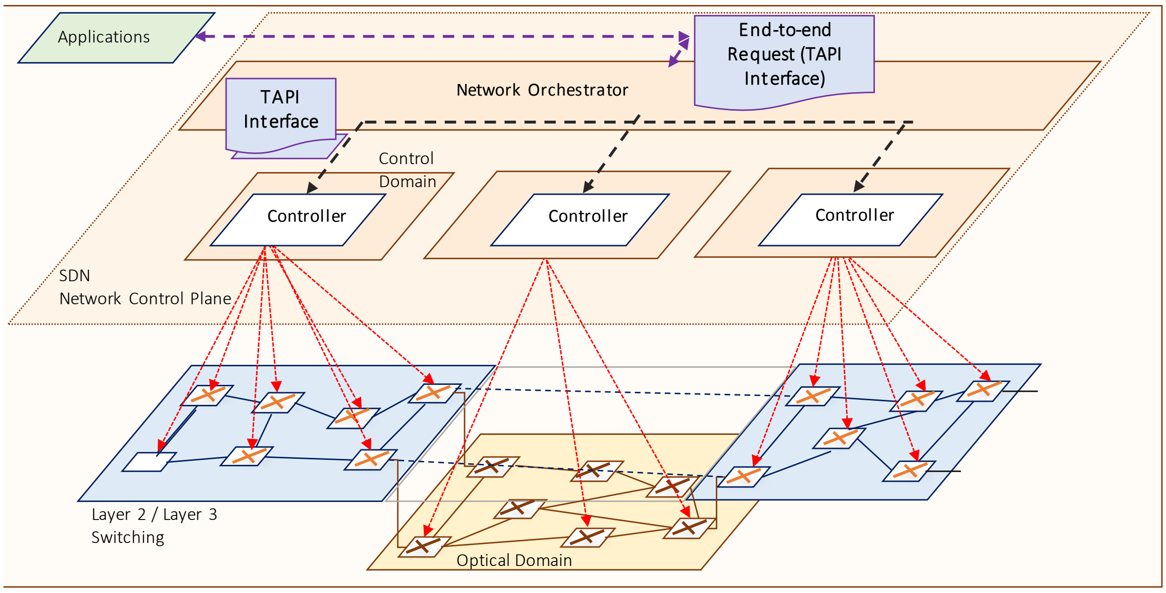
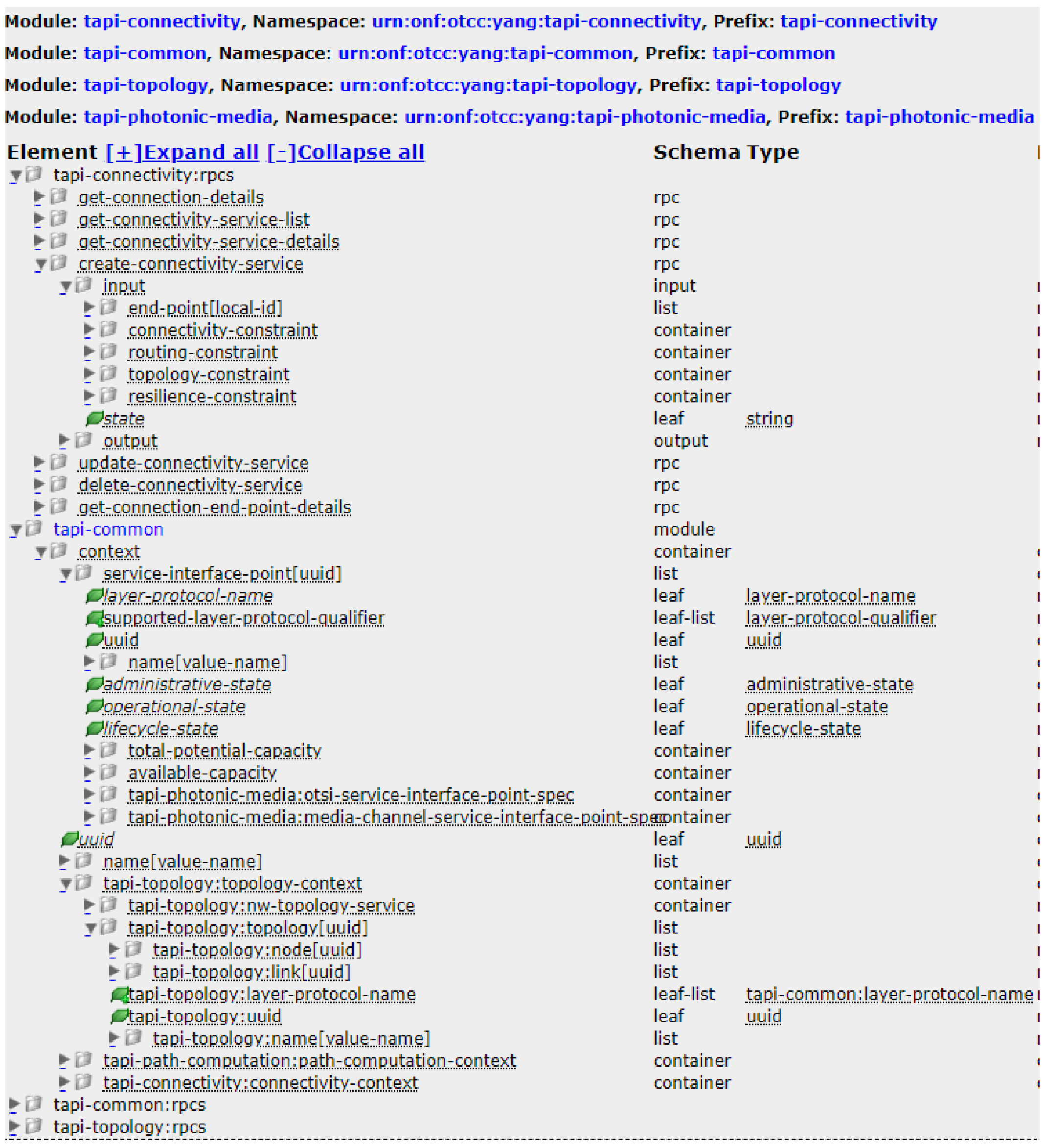


Figure 42. TAPI use case on the orchestration of multiple network domains.

Connectivity context and models. Finally, the third model augments the shared context in order to support Connectivity Services. The instantiation of a connectivity service relies on the instantiation of several connections (e.g., one end-to-end connection and connections internal to each TAPI node). For this, Connection End Points (CEPs) are instantiated over NEPs (and contain information about the connections), and connections involve two or more CEPs.

The METRO-HAUL implementation is of TAPI 2.1, which includes extensions for the photonic media layer. As shown in Figure 43, the basic functions of context and topology retrieval as well as the service provisioning and release have been implemented.



Module: tapi-connectivity, **Namespace:** urn:onf:otcc:yang:tapi-connectivity, **Prefix:** tapi-connectivity
Module: tapi-common, **Namespace:** urn:onf:otcc:yang:tapi-common, **Prefix:** tapi-common
Module: tapi-topology, **Namespace:** urn:onf:otcc:yang:tapi-topology, **Prefix:** tapi-topology
Module: tapi-photonic-media, **Namespace:** urn:onf:otcc:yang:tapi-photonic-media, **Prefix:** tapi-photonic-media

```
Element [+]Expand all [-]Collapse all                           Schema Type
tapi-connectivity:rpcs
  get-connection-details                                        rpc
  get-connectivity-service-list                                 rpc
  get-connectivity-service-details                              rpc
  create-connectivity-service                                   rpc
    input                                                       input
      end-point[local-id]                                       list
      connectivity-constraint                                   container
      routing-constraint                                        container
      topology-constraint                                       container
      resilience-constraint                                     container
      state                                                     leaf       string
    output                                                      output
  update-connectivity-service                                   rpc
  delete-connectivity-service                                   rpc
  get-connection-end-point-details                              rpc
tapi-common                                                     module
  context                                                       container
    service-interface-point[uuid]                               list
      layer-protocol-name                                       leaf       layer-protocol-name
      supported-layer-protocol-qualifier                        leaf-list  layer-protocol-qualifier
      uuid                                                      leaf       uuid
      name[value-name]                                          list
      administrative-state                                      leaf       administrative-state
      operational-state                                         leaf       operational-state
      lifecycle-state                                           leaf       lifecycle-state
      total-potential-capacity                                  container
      available-capacity                                        container
      tapi-photonic-media:otsi-service-interface-point-spec     container
      tapi-photonic-media:media-channel-service-interface-point-spec container
    uuid                                                        leaf       uuid
    name[value-name]                                            list
    tapi-topology:topology-context                              container
      tapi-topology:nw-topology-service                         container
      tapi-topology:topology[uuid]                              list
        tapi-topology:node[uuid]                                list
        tapi-topology:link[uuid]                                list
        tapi-topology:layer-protocol-name                       leaf-list  tapi-common:layer-protocol-name
        tapi-topology:uuid                                      leaf       uuid
        tapi-topology:name[value-name]                          list
    tapi-path-computation:path-computation-context              container
    tapi-connectivity:connectivity-context                      container
tapi-common:rpcs
tapi-topology:rpcs
```

Figure 43. TAPI Selected YANG models (tree view)

Function	**Implemented interface**
TAPI Context for Service Interface Points	• The client or parent controller can perform a GET on the context resource, which has been augmented with topology and connectivity information. • For example, for an SDN controller at IP 10.1.1.146, the parent needs to perform a GET resource • `http://10.1.1.146:8181/onos/RESTCONF/data/tapi-common:context`

Example	The JSON encoded reply encodes the context, including the Service Interface Points, which can be either transceiver client side (DSR) or line side (PHOTONIC media). An excerpt of the context is as follows:
	
	`"service-interface-point" : [`
	`        {`
	`           "uuid" : "c72bbb8a-2fa1-49a7-b248-b7b09375e5d4",`

```
            "name" : [
               {
                  "value" : "NETCONF:10.1.1.174:830/101",
                  "value-name" : "onos-cp"
               }
            ],
            "layer-protocol-name" : "PHOTONIC_MEDIA"
         },
         {
            "name" : [
               {
                  "value-name" : "onos-cp",
                  "value" : "NETCONF:10.1.1.174:830/3"
               }
            ],
            "uuid" : "80bd3d6b-1c75-4cd4-9fdc-93c842bbb2fa",
            "layer-protocol-name" : "DSR"
         },
```

Function	**Implemented interface**
TAPI Context with topology	• The TAPI context has been augmented with information regarding the TAPI topology, following the models • Similar to the previous call, we get the list of topologies, links, nodes, node edge points, etc. • `http://10.1.1.146:8181/onos/RESTCONF/data/tapi-common:context`
Example	The JSON encoded reply encodes the topologies, as follows (only a small excerpt is shown):

```
{
   "tapi-common:context" : {
      "tapi-connectivity:connectivity-context" : {},
      "tapi-topology:topology-context" : {
         "topology" : [
            {
               "uuid" : "e3e88aa0-646e-40c2-a06f-174b47062623",
               "node" : [
                  {
                     "uuid" : "4929bb2d-f82b-45a2-99dd-bbc4adf2acc0",
                     "name" : [
                        {
                           "value-name" : "device-id",
                           "value" : "NETCONF:10.1.1.167:830"
                        }
                     ],
                     "owned-node-edge-point" : [
                        {
                           "uuid" : "80ba66c9-b952-457c-9991-93502d4d8271",
                           "name" : [
                              {
                                 "value-name" : "odtn-port-type"
                              },
                              {
                                 "value-name" : "odtn-connection-id"
                              },
```

	`{`
	`"value" : "NETCONF:10.1.1.167:830/204",`
	`"value-name" : "onos-cp"`
	`}`
	`],`
	`"tapi-connectivity:cep-list" : {`

Function	**Implemented interface**
TAPI Connectivity Service Creation and deletion	• The TAPI context has been augmented with information regarding the TAPI connections. The Remote Procedure Call method has been selected. • Create connectivity `http://10.1.1.146:8181/onos/RESTCONF/operations/tapi-connectivity:create-connectivity-service`

Example	Sample exchange between parent and ONOS controller. The TAPI client requests is typically (for the connection request between line ports):
	
	`{`
	`  "tapi-connectivity:input" : {`
	`    "end-point" : [`
	`      {`
	`        "role" : "UNKNOWN",`
	`        "protection-role" : "WORK",`
	`        "layer-protocol-qualifier" :`
	`          "tapi-photonic-media:PHOTONIC_LAYER_QUALIFIER_NMC",`
	`        "local-id" : "c72bbb8a-2fa1-49a7-b248-b7b09375e5d4",`
	`        "service-interface-point" : {`
	`          "service-interface-point-uuid" :`
	`            "c72bbb8a-2fa1-49a7-b248-b7b09375e5d4"`
	`        },`
	`        "layer-protocol-name" : "PHOTONIC_MEDIA"`
	`      },`
	`      {`
	`        "layer-protocol-name" : "PHOTONIC_MEDIA",`
	`        "service-interface-point" : {`
	`          "service-interface-point-uuid" :`
	`            "783d8d9f-b0ed-429b-8380-8c119fd1efd5"`
	`        },`
	`        "local-id" : "783d8d9f-b0ed-429b-8380-8c119fd1efd5",`
	`        "layer-protocol-qualifier" :`
	`          "tapi-photonic-media:PHOTONIC_LAYER_QUALIFIER_NMC",`
	`        "protection-role" : "WORK",`
	`        "role" : "UNKNOWN"`
	`      }`
	`    ]`
	`  }`
	`}`
	
	If the service is established successfully, ONOS returns:
	
	`{`
	`  "tapi-connectivity:output" : {`
	`    "service" : {`
	`      "connection" : [`

```
            {
               "connection-uuid" : "1bdd326a-2837-42b2-882d-5fa01e3b589d"
            }
         ],
         "uuid" : "e68ed95b-042b-403c-b30b-060520e82c39",
         "end-point" : [
            {
               "service-interface-point" : {
                  "service-interface-point-uuid" :
                    "c72bbb8a-2fa1-49a7-b248-b7b09375e5d4"
               },
               "local-id" : "76f10c2f-cac3-4339-b86a-734ee3f01811"
            },
            {
               "local-id" : "722966f3-fc14-449d-84c5-78f2523bd3e8",
               "service-interface-point" : {
                  "service-interface-point-uuid" :
                     "783d8d9f-b0ed-429b-8380-8c119fd1efd5"
               }
            }
         ]
      }
   }
}
```

4.6 MCOM

The M-COM interface is devoted to synchronizing information from operational databases from the COM module to the monitoring platform. This information can be correlated with monitoring data and enables issuing of notifications to the COM module about different events, even providing recommended actions. The M-COM interface can also be used to manage telemetry functionalities from monitorable network/IT devices; in that regard, the M-COM interface offers methods to create/modify/delete OPs referring to telemetry streams in network/IT devices thus, configuring them to export telemetry data to the appropriate MDA platform components.

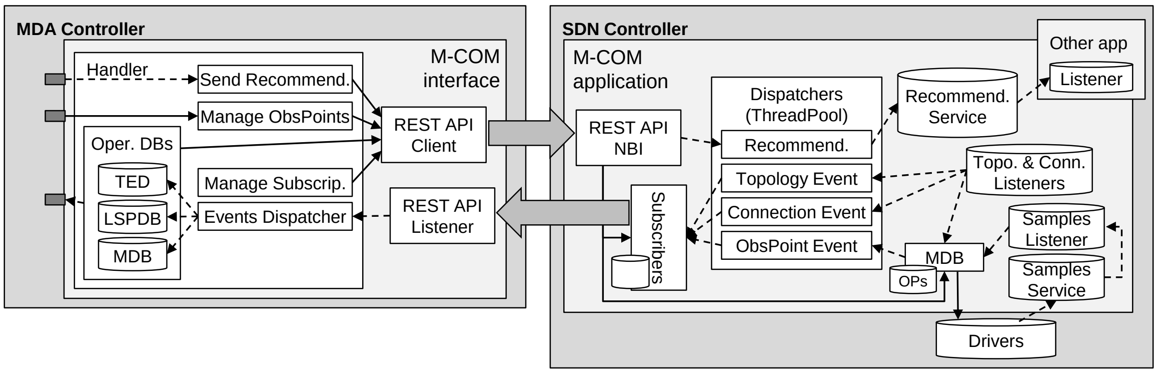


Figure 44. Components of the M-COM interface

Figure 44 illustrates the components belonging to the M-COM interface at both, the MDA controller and SDN controller sides. At the COM module side, the M-COM interface is implemented as a pluggable application and consists of: *i*) the *REST API NBI* block publishes M-COM management commands on the COM North Bound Interface (NBI) to receive control commands, *ii*) the *Subscribers* block keeps track of the subscribers that requested to receive database updates from the COM module and distributes events to them, *iii*) the *Dispatchers* block is implemented as a thread pool and integrates the logic to process M-COM – related events, *iv*) the *Recommendation Service* provides a means for other applications in the SDN controller to subscribe to messages received from the MDA controller, *v*) the *Topology & Connection Listeners* are in charge of capturing events from the COM operational databases and triggering the execution of appropriate tasks in the *Dispatcher* block, *vi*) the *MDB* block keeps track of the OPs requested by the MDA controller and configures the COM *Drivers* to enable telemetry and send the collected samples directly to the MDA controller, *vii*) the *Samples Service* enables *Drivers* to push their telemetry data to be received by those *Sample Listeners* connected to the service, and *viii*) the Samples Listener collects the telemetry samples issued by the COM *Drivers* and, if configured, forwards them to the subscribers of the MDB database.

At the MDA controller side, the M-COM interface consists of: *i*) a *REST API Client* that serves as a gateway to issue RPC commands to the COM module, *ii*) a *REST API Listener* that receives asynchronous messages from the COM module, such as database change events, and *iii*) a *Handler* block that integrates the logic of the M-COM interface. The latter, in turn, consists of four sub-blocks: *i*) the *Operational Databases* stores a clone of the operational databases in the COM module and is updated by database change events from the *REST API Listener*, *ii*) the *Manage Subscriptions* block deals with subscriptions handling to receive database changes from the COM, *iii*) the *Manage ObsPoints* block provides an interface to (de)configure OPs at the COM from the MDA controller, and *iv*) the *Send Recommendation* block enables the MDA controller to issue messages to feed applications in the COM module.

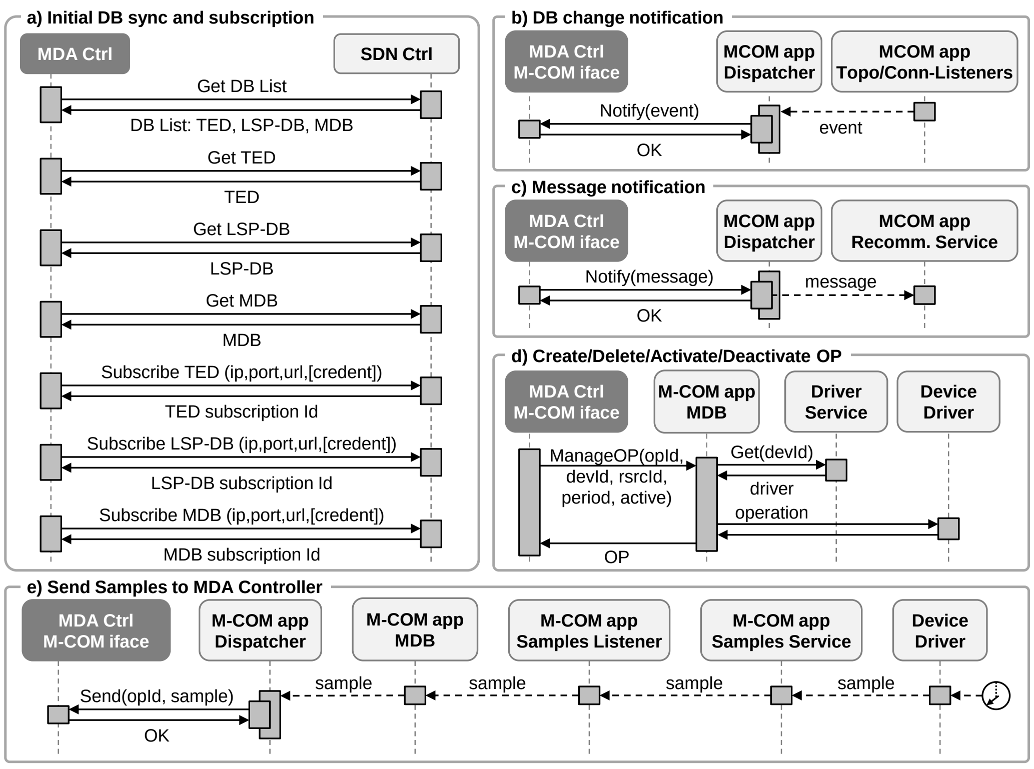


Figure 45. Operations supported by M-COM interface

Figure 45 illustrates the workflows and messages defined for the M-COM interface. Table 10 describes the implemented interface.

Table 10. M-COM Implemented interface

Function	**Implemented interface**
Get list of databases	Returns the list of *Database* identifiers for those operational databases managed by a COM module (i.e., TED and LSP-DB for T-SDN controller). OUTPUT: List of *Database* instances and details for each of them, including: • *Database-Identifier*: String • *Database-Name*: String • *Resource-Type* list: Enumeration value
Get database	Returns contents of the operational database instance identified by the provided inputs. INPUT: Database-Identifier (mandatory): String OUTPUT: List of Database instances and details for each of them, including: • *Database-Identifier*: String • *Database-Name*: String • *Resource* list: List of objects or error if some field is invalid.
Subscribe to database changes	Subscribes the MDA controller to asynchronous notifications of database changes to avoid periodic polling processes. The requests should carry the endpoint (*IP-address*, *Port*, and *URL*) as well as optional credentials (*Username* and *Password*) that the M-COM application will use to issue the database change notifications. INPUT: • *Database-Identifier* (mandatory): String • *IP-address (mandatory): String* • *Port (mandatory): Integer* • *URL (mandatory): String* • *Username (optional): String* • *Password (optional): String* OUTPUT: Acknowledgement with the subscription identifier, or error if some field is invalid.
Notify database change	Notifies about changes in a resource within database. NOTIFICATION: • *Database-Identifier*: String • *Time:* Integer encoding the timestamp of the change • *Type:* Enumerated value encoding the type of change • *Resource:* List of pointers identifying the resource location • *Attributes:* List of resource attributes
Notify message	Notifies about messages from the MDA controller to the COM NOTIFICATION: • *Time:* Integer encoding the timestamp of the change • *Event-Type*: Enumeration value • *Message*: Object

Manage observation points	Create/get/update/remove OPs in the COM module and trigger appropriate operations in the related network/IT devices through COM device drivers. INPUT: • *OP-Identifier* (mandatory): Integer • *Device-Identifier* (mandatory)*: String* • *Resource-Identifier* (mandatory)*: String* • *Period* (mandatory)*: Float* • *Active* (mandatory)*: Boolean* • *Template-Identifier* (optional): Integer • *Driver-Parameters* (optional): map with key-value pairs OUTPUT: Acknowledgement with the OP attributes, or error if some field is invalid or device does not support monitoring configuration
Send Samples	When this functionality is available at a COM's device driver, the driver periodically collects telemetry samples from the device and issues them to the MDA controller through the M-COM interface. In case this functionality is not available in the driver, the device needs to be configured to forward the samples to some component in the MDA platform, e.g., an MDA agent or the MDA controller. NOTIFICATION: • *Database-Identifier* (mandatory): String (fixed to MDB) • *Time* (mandatory)*:* Integer encoding the timestamp of the sample • *Template-Identifier* (mandatory)*:* Integer • *Resource* (mandatory)*:* list of pointers identifying the resource location • *OP-Identifier* (mandatory): Integer • *Values* (mandatory): Object containing the sample values

4.7 IO4

The IO4 interface is designed to enable an external system, such as an OSS/BSS, to access the collected data repository at the MDA controller to retrieve raw data from network/IT devices, i.e., for monitoring, billing, or analysing different scenarios.

Figure 46 illustrates the components belonging to the IO4 interface. From the MDA controller side, the interface consists of a *REST API server* that receives commands from the external system and triggers the execution of tasks in a *Requests Dispatcher*. The *Requests Dispatcher* accesses to the *MDB* to retrieve OPs and validate the requests, and to the *Samples Repository* to retrieve the samples according to the request parameters, i.e., the time frame and the OP identifier.

In a generalist way, an external system willing to use the IO4 interface only needs to implement a simple *REST API Client* and a *Monitoring Handler* block responsible for managing the monitoring entities in the MDA controller. That *Monitoring Handler* needs to implement two databases: an *MDB* to store the attributes retrieved for each OP requested to the MDA controller, and a *Samples* database to store the samples belonging to these OPs. Note that when an OP is requested, the attributes should carry OP's sampling period; the *Monitoring Handler* at the external system should schedule sample requests with that specific periodicity to prevent flooding the MDA controller with unneeded/redundant requests.

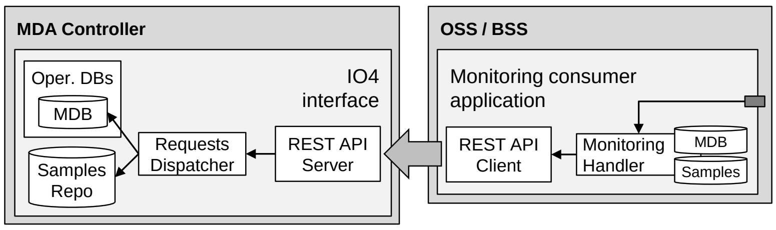


Figure 46. Components of the IO4 interface

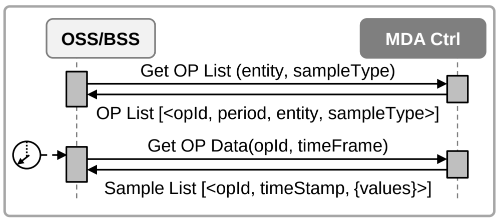

OSS/BSS
MDA Ctrl
Get OP List (entity, sampleType)
OP List [<opId, period, entity, sampleType>]
Get OP Data(opId, timeFrame)
Sample List [<opId, timeStamp, {values}>]

Figure 47. Operations of the IO4 interface

Figure 47 illustrates the workflows and messages defined for the IO4 interface while Table 1 describes the implemented interface.

Table 11. IO-4 Implemented interface

Function	**Implemented interface**
Get list of OPs	Returns the list of OPs that fulfil a set of optional input criteria, i.e., entity name and sample type. INPUT: • *Entity-Name* list (optional): String to filter returned OPs by entity owning the OPs. • *Observation-Point-Type* list (optional): Enumeration value to filter returned OPs by type. OUTPUT: List of *Observation-Point* instances and details for each of them, including: • *Observation-Point-Identifier*: String • *Period*: Float • *Entity*: String with the device/resource identifier of the OP • *Sample-Type*: Enumeration value
Retrieve OP data	Returns the list of available samples in the repository for a given OP. INPUT: • *Observation-Point-Identifier* (mandatory): String • *Start-Time* (optional): Datetime • *End-Time* (optional): Datetime ○ If set Start-Time and/or End-Time, data will be returned only for the specified time window OUTPUT: Available data in the repository, or error if invalid parameters are provided.

4.8 MDA controller - MDA agent (SBIm/NBIm)

Two interfaces are defined between the MDA controller and MDA agents. While SBIm is designed for issuing monitoring-related configuration messages from the MDA controller to the MDA agent, NBIm deals with asynchronous retrieval of aggregated samples from the MDA agent to the MDA controller by means of the IPFIX protocol.

4.8.1 SBIm

The SBIm interface is defined for issuing monitoring-related configuration messages to the MDA agents from the MDA controller. The interface messages are defined using YANG and transport of messages is implemented by means of a RESTCONF API Client and Server.

Figure 48 presents the structure of the implemented YANG data model that is used between the MDA controller and MDA agents. The model defines two differentiated subtrees with the objective to separate configuration from monitoring and telemetry responsibilities. The *configuration* subtree includes every programmable or monitorable component in the node, whilst the *monitoring* subtree includes monitoring capabilities and OPs and it is specifically designed to facilitate autodiscovery of network device components by MDA agents.

A key element in the model is the *component*, representing any configurable or monitorable element in the node and is locally identified by its *component-id*. Nodes are assumed to feed a data store compliant with this model during bootstrapping, whereas dissemination towards MDA agents and SDN controller can be made per-update notification or by polling. Although components under the configuration and monitoring subtrees are related, we relax such conditions to allow obtaining only the monitoring subtree during the auto discovery process. In fact, the configuration subtree is not stored in the MDA agent to avoid synchronization issues; whenever configuration of some component is required by a KDD application, the local configuration module retrieves it directly from the node controller.

Monitoring/telemetry in monitorable components can be activated by creating and enabling OPs and deactivated by disabling and/or deleting OPs. Monitorable components include a list of tuples with supported monitoring template identifiers (*template-id*) and the container, e.g., an interface, where measurements will be done (*container-component-id*). Monitoring templates represent different measurement methods, so each OP is associated to one single monitoring template and interface tuple. We use the *template-id* field to identify the monitoring method used for the measurements, as well as their data structure. Specifically, we define the following identifiers: *i*) *template-id* 310 identifies monitoring of BER and optical power in the transponders; *ii*) *template-id* 330 identifies optical spectrum monitoring performed by OSAs, which send sequences of tuples <frequency, optical power>.

The *monitoring* subtree also includes every existing OP (enabled, i.e., performing measurements, or disabled). Specifically, the ability to enable and disable existing OPs supports allocation of monitoring devices in active monitoring schemes. Nodes are expected to allocate monitoring resources by translating tuples <*component-id*, *template-id*, *container-component-id*> into device-specific commands. Observation domain/point identifiers are internal identifiers used by KDD processes inside MDA agents to isolate different datasets, and they are included in monitoring messages. Multiple OPs can be defined for a single component, each reporting monitored data formatted as per the selected supported template.

Figure 48 presents an example for a lightpath (L0-LSP): configuration parameters include, but are not limited to, modulation format, bit/baud-rate, and the allocated frequency slot specified by *central-frequency* and *slot-width* parameters. In the *monitoring* subtree, *components* include two relevant attributes: *i*) *monitorable-element* (“what”), which is used for correlation purposes between the operational databases in the SDN controller and the MDA controller; and *ii*) *monitorable-capabilities* containing a set of pairs with the supported *template-id* (“how”), representing the monitoring method that such component supports in the specific container component, identified by its *container-component-id* in the network node (“where”).

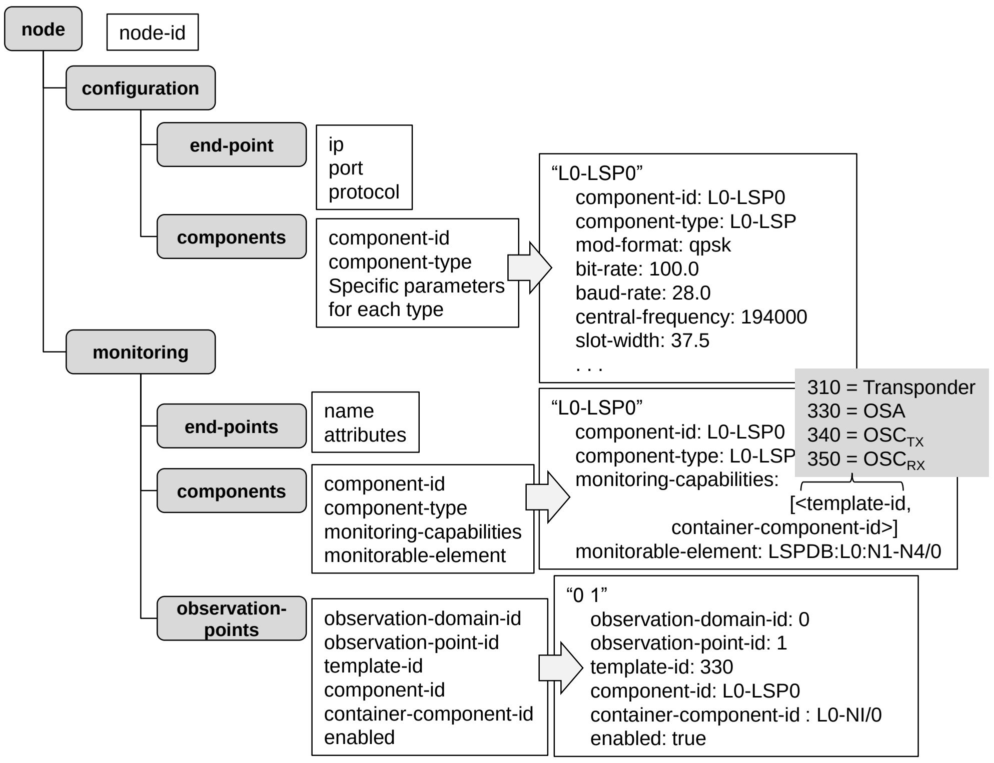


Figure 48. YANG data model (and example values) for SBIm interface

For illustrative purposes all of the templates have been included in Figure 48, but only those supported by the component in the network node will be advertised. Finally, an OP is currently active; specifically, L0-LSP0 is being monitored in interface L0-NI/0 using an OSA.

4.8.2 NBIm

NBIm deals with asynchronous retrieval of aggregated samples from the MDA agent to the MDA controller by means of the IPFIX protocol [RFC7011] extended with [RFC6313] to support encoding complex data structures; in particular, *BasicList* and *SubTemplateList* are required to enable exporting samples containing lists of scalar fields and lists of records, respectively, such as L0 Optical Spectrum Analyzer samples that are encoded as a list of tuples with the measured optical power in dBm for each frequency in the spectrum. In IPFIX, each data record must fulfil an IPFIX template that defines the meaning of the binary data carried in data records. Table 12 to Table 14 details the custom IPFIX fields, templates and sub-templates that have been defined for NBIm.

NBIm uses several custom IPFIX fields when a suitable IANA-registered/IETF-standardised one is not available. These fields are identified by a *field identifier* and a Private Enterprise Number (PEN). Custom fields defined specifically for NBIm, are identified using UPC's PEN (25785). Another PEN that we use for existing fields defined by other enterprises is VMWARE PEN (6876). Table 12 describes the list of custom IPFIX fields defined and used in the templates defined in Table 13 and Table 14.

Table 12. Custom IPFIX fields used in NBIm interface

Field Id – Name (PEN Id – Name)	**Encoding**	**Description**
893 – tunnelSourceIPv4Address (6876 – VMWARE)	unsigned32	see Open vSwitch Manual [OVS]
894 – tunnelDestinationIPv4Address (6876 – VMWARE)	unsigned32	see Open vSwitch Manual [OVS-manual-ref]
1002 – layer2BitDeltaCount (25785 – UPC)	unsigned64	Number of L2 octets (headers + payload) since the previous report (if any) in incoming packets for this flow at the Observation Point/Group.
1003 – ber (25785 – UPC)	float64	Measured Bit Error Rate since the previous report (if any) in incoming packets for this flow at the Observation Point/Group.
1006 – direction (25785 – UPC)	unsigned8	Direction of the flow: ingress (0x00) / egress (0x01), bidirectional (0x02).
1007 – observationGroupId (25785 – UPC)	unsigned64	An identifier of an Observation Group that is unique per network slice. Typically used for limiting the scope of other Information Elements.
1008 – rxPowerDecibelMilliwatts (25785 – UPC)	float64	Received optical power in dBm.
1009 – txPowerDecibelMilliwatts (25785 – UPC)	float64	Transmitted optical power in dBm.
1014 – frequencyGigaHertz (25785 – UPC)	float64	Frequency in GHz.
1015 – stlPaginationIndex (25785 – UPC)	unsigned16	Index of the page of the STL that follows these pagination fields.
1016 – stlPaginationTotal (25785 – UPC)	unsigned16	Total number of the pages for the STL that follows these pagination fields.

Table 13 summarizes the IPFIX templates used at the NBIm interface. For each field, a comment was added specifying if the field is a custom field defined in Table 12 (Custom) or a *SubTemplateList* (STL, *sub-template-id*). No comment is included for IANA-registered/IETF-standardised fields. For STLs, the *sub-template-id* is a reference to a sub-template defined in Table 14.

Note that each IPFIX message requires a mandatory header containing the *observation domain identifier* that is used as a network slice identifier and the timestamp in seconds since UNIX epoch for the message. These fields have been deliberately avoided in the following template definitions to keep them as comprehensive as possible.

Since a *SubTemplateList* could contain huge number of records, the concept of pagination of an STL has been defined. Each STL not fitting in a single IPFIX message can be split in several separate STL records by the IPFIX Exporter process to be sent in different IPFIX messages by adding the fields *stlPaginationIndex* and *stlPaginationTotal* to the *subTemplateList* field. Upon reception of the IPFIX messages, the IPFIX collector process (or another process taking the role of processing these IPFIX messages) should reconstruct the STL pages into a single STL.

Table 13. IPFIX Templates used in NBIm interface

Template Id / Name (Kind)	**Field (Comment, if any)**
300 L0 Link Power in dBm (Collector)	observationPointId
	rxPowerDecibelMilliwatts (Custom)
301 L0 Link Power in dBm (Exporter Sliced)	originalObservationDomainId
	observationGroupId (Custom)
	rxPowerDecibelMilliwatts (Custom)
302 L0 Link Power in dBm (Exporter Not Sliced)	observationGroupId (Custom)
	rxPowerDecibelMilliwatts (Custom)
310 L0 Transponder BER, Tx and Rx Power in dBm (Collector)	observationPointId
	tunnelSourceIPv4Address (Custom)
	tunnelDestinationIPv4Address (Custom)
	direction (Custom)
	ber (Custom)
	rxPowerDecibelMilliwatts (Custom)
	txPowerDecibelMilliwatts (Custom)
311 L0 Transponder BER, Tx and Rx Power in dBm (Exporter Sliced)	originalObservationDomainId
	observationGroupId (Custom)
	direction (Custom)
	ber (Custom)
	rxPowerDecibelMilliwatts (Custom)
	txPowerDecibelMilliwatts (Custom)

Template Id / Name (Kind)	Field (Comment, if any)
	flowDurationMilliseconds
312 L0 Transponder BER, Tx and Rx Power in dBm (Exporter Not Sliced)	observationGroupId (Custom)
	direction (Custom)
	ber (Custom)
	rxPowerDecibelMilliwatts (Custom)
	txPowerDecibelMilliwatts (Custom)
	flowDurationMilliseconds
330 L0 OSA Optical Spectrum in dBm (Collector)	observationPointId
	stlPaginationIndex (Custom)
	stlPaginationTotal (Custom)
	subTemplateList (STL, 10000)
	paddingOctets
331 L0 OSA Optical Spectrum in dBm (Exporter Sliced)	originalObservationDomainId
	observationGroupId (Custom)
	stlPaginationIndex (Custom)
	stlPaginationTotal (Custom)
	subTemplateList (STL, 10000)
	paddingOctets
332 L0 OSA Optical Spectrum in dBm (Exporter Not Sliced)	observationGroupId (Custom)
	stlPaginationIndex (Custom)
	stlPaginationTotal (Custom)
	subTemplateList (STL, 10000)
	paddingOctets
256 L2 Traffic OpenVSwitch Ethernet Flows w/o L3 w/o L4 (Collector)	observationPointId
	flowDirection
	sourceMacAddress
	destinationMacAddress
	ethernetType
	ethernetHeaderLength
	flowStartDeltaMicroseconds
	flowEndDeltaMicroseconds

Template Id / Name (Kind)	Field (Comment, if any)
	packetDeltaCount
	layer2OctetDeltaCount
	flowEndReason
264 L2 Traffic OpenVSwitch Ethernet Flows w/IPv4 w/TCP-UDP-SCTP (Collector)	observationPointId
	flowDirection
	sourceMacAddress
	destinationMacAddress
	ethernetType
	ethernetHeaderLength
	ipVersion
	ipTTL
	protocolIdentifier
	ipDiffServCodePoint
	ipPrecedence
	ipClassOfService
	sourceIPv4Address
	destinationIPv4Address
	sourceTransportPort
	destinationTransportPort
	flowStartDeltaMicroseconds
	flowEndDeltaMicroseconds
	packetDeltaCount
	layer2OctetDeltaCount
	flowEndReason
	octetDeltaCount
	octetDeltaSumOfSquares
	minimumIpTotalLength
	maximumIpTotalLength
266 L2 Traffic OpenVSwitch Ethernet Flows w/IPv4 w/ICMP	observationPointId
	flowDirection
	sourceMacAddress

Template Id / Name (Kind)	Field (Comment, if any)
(Collector)	destinationMacAddress
	ethernetType
	ethernetHeaderLength
	ipVersion
	ipTTL
	protocolIdentifier
	ipDiffServCodePoint
	ipPrecedence
	ipClassOfService
	sourceIPv4Address
	destinationIPv4Address
	icmpTypeIPv4
	icmpCodeIPv4
	flowStartDeltaMicroseconds
	flowEndDeltaMicroseconds
	packetDeltaCount
	layer2OctetDeltaCount
	flowEndReason
	octetDeltaCount
	octetDeltaSumOfSquares
	minimumIpTotalLength
	maximumIpTotalLength
321 L2 Traffic Ethernet Flows (Exporter Sliced)	originalObservationDomainId
	observationGroupId
	packetDeltaCount
	layer2BitDeltaCount
	flowDurationMilliseconds
322 L2 Traffic Ethernet Flows (Exporter Not Sliced)	observationGroupId
	packetDeltaCount
	layer2BitDeltaCount
	flowDurationMilliseconds

Table 14 summarizes the IPFIX sub templates used in NBIm interface. For each field, a comment was added specifying if the field is a custom field defined in Table 12 (Custom).

Table 14. IPFIX SubTemplates used in NBIm interface

SubTemplate Id / Name	Field (Comment, if any)
10000 L0 OSA Optical Spectrum tuple in dBm	frequencyGigaHertz (Custom)
	rxPowerDecibelMilliwatts (Custom)

4.9 Network Devices to MDA agent MONp/MONo

In METRO-HAUL, two different approaches for performing and collecting measurements from the devices have been considered: *monitoring* and *telemetry*. While monitoring is focused on performing measurements at regular intervals, e.g., every minute, telemetry is intended for the continuous measurement that is sent as a data stream, and no regular intervals are needed, i.e., measurements are sent as soon they are available or at a rate much shorter than for the monitoring approach, e.g., one per second. For both, monitoring and telemetry, every measurement might convey values for a set of parameters that are collected simultaneously and have a complete meaning. Therefore, measurements are defined as data records that follow a given data structure defined as a template, i.e., different templates are defined for different measurements to be collected.

Two different interfaces have been defined for network device monitoring: monitoring of packet devices (MONp) and monitoring for optical devices (MONo). Table 15 presents the template (*templateId*: 256) defined for MPLS-TP LSPs under MONp interface, where number of packets and number of bytes are measured.

Table 15 Template 256 for MPLS-TP LSPs

Field Name	Description
packetDeltaCount	Number of packets since the previous report.
layer2OctetDeltaCount	Number of L2 octets since the previous report.

For the optical later, two different templates have been defined under the MONo interface so far for BER and optical power measured at the optical transponders (*templateId*: 310) and for the spectrum acquired by Optical Spectrum Analysers (*templateId*: 330). Table 16 presents *templateId* 310, where BER and optical power are metered. Finally, Table 17 defines *templateId* 330 for optical spectrum monitoring.

Table 16 Template 310 for Lightpaths at the transponders

Field Name	Description
Ber	Bit error rate

rxPowerDecibelMilliwatts	Received optical power (dBm)
txPowerDecibelMilliwatts	Transmitted optical power (dBm)

Table 17 Template 330 for optical spectrum

Field Name	Description
opticalSpectrum	Ordered vector of tuples <frequencyGigaHertz, powerDecibelMilliwatts>

Although the previous templates are defined to be protocol independent, other fields are needed to be included when monitoring is used, since individual messages are sent for every measurement. Specifically, the Observation Point (OP) id and the direction in which the measurement has been performed are commonly needed, as defined in Table 18.

Table 18 Additional fields for monitoring messages

Field Name	Description
observationPointId	Id of the observation point
direction	Direction of the connection / link

4.10 Optical SDN to OpenROADM and OpenConfig

This section provides details on the ONOS drivers implemented to support optical devices modelled with the OpenROADM and the OpenConfig standard. Both drivers implement a NETCONF southbound interface that interacts with NETCONF server agents residing on the data plane devices. Specifically, in line with the work of the ODTN group, the OpenROADM driver is devoted to communicating with ROADM devices while the OpenConfig driver is devoted to communicating with transponder devices.

Key implementation files for the OpenROADM driver are available at the link below. It has been tested with ONOS 1.12 in a collaborative work among TIM, CTTC, and CNIT, and was published at ECOC 2018. The plan is to contribute them to the ODTN group in the next weeks.

https://gitlab.com/METRO-HAUL/optical-controller/tim-controller/tree/onos-1.12/drivers/metrohaul

Key implementation files for the OpenConfig driver have been already contributed to the ODTN group and are currently in a revision phase. Code is currently available at the link below. It has been tested with ONOS 2.1 in a collaborative work among CNIT and PoliMi, and was published at OFC 2019.

https://gerrit.onosproject.org/#/c/21550/

The OpenConfig driver has been tested in conjunction with the OpenConfig agent developed at CNIT. Besides virtual hardware, that agent (based on ConfD tool, please refer to D3.2 for additional details) current supports Ericsson transponders; moreover, it is in test with transponders provided by other METRO-HAUL partners.

The OpenROADM driver has been tested in conjunction with the OpenROADM agents developed at TIM (based on the Net2peer tool) and at CTTC (based on the ConfD tool). Please refer to D3.2 for additional details of the agents. Besides virtual hardware, the TIM agent is currently supporting an experimental ROADM device deployed at TIM, and it is in test with other ROADM devices provided by other METRO-HAUL partners. In particular, an agent is in development in collaboration between CNIT and Ericsson for the support of the IRIS device matrix developed by Ericsson (please refer to D3.1 for additional details).

Function	**Implemented interface**
DeviceDescription Discovery	• Driver Implementation of the DeviceDescription discovery for both OpenROADM and OpenConfig YANG • Implemented methods discover device details interfacing with the device via NECONF calls, for both OpenROADM and OpenConfig YANG • Implemented method to discover device ports and features discoverPortDetails abstract methods, for both OpenROADM and OpenConfig YANG • Implemented method to support unidirectional ports and associate them with its counterpart partner-port, for OpenROADM YANG
Lambda Query	• Implemented methods discover tunability requirements for each of the OLS ports, in order to ensure an end-to-end path computation with optical wavelength continuity, for both OpenROADM and OpenConfig YANG
FlowRule Programmable	• Driver implementation of the FlowRuleProgrammable behaviour for both OpenROADM and OpenConfig YANG • Implemented methods: getFlowEntries, applyFlowRules, removeFlowRules, etc.

4.11 Optical SDN to OLS

For this interface, the implemented driver behaves as if the OLS was an ONOS device. In short, the driver implements a RestSB (A South bound interface that interacts with an OLS using a REST interface) that also relies on the TAPI models. Key implementation files are in the repository

https://github.com/opennetworkinglab/onos/tree/master/drivers/odtn-driver/src/main/java/org/onosproject/drivers/odtn/tapi

Note that this interface complements the one described in Section 4.4 regarding the use of TAPI, when applied recursively to control an OLS subsystem in such disaggregation model. Much like in similar drivers, 3 main behaviours are implemented: discovery (to discover a device capability, ports, etc), lambda query (to discover tunability capabilities), and flow rule programmable (to configure the connections and forwarding behaviour of the devices).

Function	**Implemented interface**
TapiDeviceDescription Discovery	• Driver implementation of the DeviceDescription discovery for ONF TransporTAPI (TAPI) v2.1 based open line systems (OLS) • Implemented methods discover device details interfacing with the device via REST calls • Implemented method to discover device ports and features discoverPortDetails abstract methods, retrieve TAPI Service Interface Points and inform the ONOS controller about the mappings
TapiDeviceLambda Query	• Driver implementation of the TAPI Device Lambda query discovery for ONF TransporTAPI (TAPI) v2.1 based open line systems (OLS) • Implemented methods discover tunability requirements for each of the OLS ports, in order to ensure an end-to-end path computation with optical wavelength continuity • Implemented method to discover device ports and features queryLambdas per port number, manage the mapping
TapiFlowRule Programmable	• Driver implementation of the FlowRuleProgrammable behaviour for ONF TransporTAPI (TAPI) v2.1 based open line systems (OLS) • Implemented methods discover device details interfacing with the device via REST calls • Implemented methods: getFlowEntries, applyFlowRules, removeFlowRules, etc.

4.12 De-fragmentation app

As described in Section 2.5.2 the de-fragmentation app is composed of two main components: the de-fragmentation tool, and a set of REST APIs that extends the features of the Optical SDN controller.

The de-fragmentation tool reads the network topology and the information about currently established lightpaths from the ONOS-based Optical SDN controller through the following commands:

Table 19 Defragmentation Server Main Functions – Interface toward ONOS controller

Main Functions	ONOS REST APIs commands
ImportNetwork ONOS Core REST APIs (Flow 1 in Figure 35)	This functions imports the network topology and Optical connections set-ups from ONOS through the following gets: • GET /topology/clusters • GET /topology/clusters/{id}/devices

	• GET /topology/clusters/{id}/links
UpdateNetworkStatus ONOS optical-rest app REST APIs (Flow 2 in Figure 35)	• GET /onos/optical/intents
ModifyNetworkStatus (Flow 6 in Figure 35)	• POST /onos/optical/intents • DELETE /onos/optical/intents

Table 20 Defragmentation engine Main Functions

Main Functions	API
Import Network (Flow 4 in Figure 35)	• addWdmNode() ○ addAddDropNode() • addOms() ○ addWdmLinkSpectrumGo()/Ret() ○ setOmsLabel(String Label) ○ setOmsCost(int cost) • addService(int idCount) ○ setLabel(String Label) ○ addSource(Int IdSource) ○ addDestination(Int IdDest) ○ addServicePath(ServicePath servicePath) ○ setServiceRate(Int rate) • addDefragmentation() ○ setActive(bool active) ○ setDefragmentationOption(Enum option)
Return Defragmentation (Flow 5 in Figure 35)	• getDefragmentationMessage() • getOpticalReconfiguration() • getFragmentationValue()

5 Conclusions

This deliverable D4.2 reports the final architecture of the METRO-HAUL COM system from their initial design in D4.1; note that the COM architecture is aligned with related initiatives, projects and standards developing organizations (SDO), like the 5GPPP, OpenConfig, OpenROADM, ODTN, and the IETF, thanks to the involvement of METRO-HAUL partners. The document also includes the status and assessment of the final implementation of METRO-HAUL functional components, including control plane validation of relevant performance indicators and the designed tests. The description of each functional component in the COM includes the list of subcomponents and their status. The functional tests carried out to validate and integrate the subcomponents are described together with the testbeds used. The definition of the specific KPIs in which the component is involved are defined and measured. Finally, the interfaces connecting functional components of the COM are defined in terms of functions with regard to requirements in D4.1 and new functions. Precisely, the alignment of the interfaces defined between components in the COM architecture with other projects and SDOs ensure inter-operability. Details of the implementation of every function are reported.

The components including in the final METRO-HAUL COM architecture show high innovative and ground-breaking features related to industry key factors, like end-to-end network services, optical disaggregation, monitoring, machine learning, and network planning, as proved by the high number of accepted top-ranked conferences and journal contributions.

In addition to the successful preliminary tests, several public demonstrations have been carried out focused on the single components and partial integration of a subset of components, see, e.g., [Ca19], [Es19], [Mo18], [Tro19b] [Mor18a], [Mor18b] and [Gar19a]. Note that the demonstrations have been carried out during top-ranked conferences in the field of optical communications and networking (like OFC and ECOC), as well as during EuCNC. Of particular interest are demonstrations of the integration between the NFVO and the network planner, between the NFVO and the parent controller, between the optical SDN controller and the MDA controller, and between the MDA controller and the traffic monitoring system. Such demonstrations probe the maturity level achieved by the components, some of which have been opened as METRO-HAUL contributions to other projects. In particular, several developments based on ONOS have been merged to the ONOS Main release, as well as released in the framework of the ODTN project.

Interestingly, the evaluated KPIs show excellent performance, and are in-line with the METRO-HAUL project KPIs. Note that the per-component KPIs defined and measures in this deliverable are the most meaningful to meet the project KPIs.

The main work now is devoted to the integration of the components for the METRO-HAUL Control Plane Demo that will be reported in deliverable D5.3. This deliverable is the demo of the control plane aspects of the METRO-HAUL project, and it will use the first releases of the developed software.

6 List of acronyms

ACTN	Abstraction and Control of Traffic-Engineered Networks
AMEN	Access-Metro Edge Node
API	Application Programming Interface
BER	Bit Error Rate
CO	Central Office
COM	Control, Orchestration and Management
CRUD	Create, Retrieve, Update and Delete
DWDM	Dense Wavelength Division Multiplexing
GPON	Gigabyte Passive Optical Network
KDD	Knowledge Discovery from Data
KPI	Key Performance Indicator
LSP	Label Switched Path
MANO	Management and Network Orchestration
MCEN	Metro Core Edge Node
MDA	Monitoring and data analytics
ML	Machine Learning
NBI	North-bound Interface
NFV	Network Functions Virtualization
NFVI	NFV Infrastructure
NFVO	NFV orchestrator
NIW	NFV over IP over WDM
NMC	Network Media Channel
NPSO	Placement and Scaling Optimizer
NRAO	Network Resource Allocation Optimizer
NS	Network Service
NSD	Network Service Descriptor
OaaS	Optimization as a Service
OLS	Open Line System
OLT	Optical Line Terminal
O-NE	Optical Network Elements
ONOS	Open Network Operating System
ONU	Optical Network Units
OP	Observation Points
OXC	Optical cross-connect
OVS	Open Virtual Switch
PCA	PON Configuration Agent
PNFA	PON Network Flow Agent
PON	Passive Optical Network
PoP	Point of presence
QoS	Quality of Service
RPC	Remote Procedure Calls
SBI	South-bound Interface
SDO	Standards Developing Organizations

SIP	Service interface point
SSH	Secure Shell
SWI	Single Wavelength Interface
TAPI	Transport API
VIM	Virtual Infrastructure Managers
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNF	Virtualized Network Function
VPN	Virtual Private Network
WIM	WAN Infrastructure Manager

7 References

[ACTN]	Ceccarelli, D., Lee, Y., editors, "Framework for Abstraction and Control of TE Networks (ACTN)", IETF RFC8453, august 2018.
[Ca19]	A. Campanella et al. "ODTN: Open Disaggregated Transport Network. Discovery and control of a disaggregated optical network through open source software and open APIs". OFC2019 Demo
[Cas18]	Casellas, R., Martinez, R., Vilalta, R., and Munoz, R., "Control, management, and orchestration of optical networks: Evolution, trends, and challenges," Journal of Lightwave Technology, vol. 36, no. 7, pp. 1390–1402, 2018
[Cas19]	Casellas, R., Giorgetti, A., Morro, R., et al "Enabling Network Slicing Across a Disaggregated Optical Transport Network", in Proceedings of the Optical Networking and Communication Conference & Exhibition (OFC), 3-7 March 2019, San Diego, CA (USA). 2019.
[D4.1]	METRO-HAUL Project Deliverable D4.1
[Es19]	P. R. Esmenats et al, "Autonomic NFV Network Services on Top of Disaggregated Optical Metro Networks" OFC2019 Demo
[Gar19a]	M. Garrich, M. Hernández-Bastida, C. San-Nicolás-Martínez, F. J. Moreno-Muro, P. Pavon-Marino, "The Net2Plan-OpenStack Project: IT Resource Manager for Metropolitan SDN/NFV Ecosystems", in Proc. of OFC19, San Diego, USA, Mar. 2019.
[Gar19b]	Miquel Garrich, César San Nicolás Martínez, Francisco-Javier Moreno Muro, María-Victoria Bueno Delgado and Pablo Pavón Mariño, "Network Optimization as a Service with Net2Plan", accepted in EuCNC19, Valencia, Spain, June 2019.
[Gio19]	Giorgetti, A., Casellas, R., Morro, R., et al., "ONOS-controlled Disaggregated Optical Networks", in Proceedings of the Optical Networking and Communication Conference & Exhibition (OFC), 3-7 March 2019, San Diego, CA (USA). 2019
[IANA]	IANA IPFIX Entities [Online]: https://www.iana.org/assignments/ipfix/ipfix.xhtml
[Mo18]	R. Morro et al. "Automated End to End Carrier Ethernet Provisioning over a Disaggregated WDM Metro Network with a Hierarchical SDN Control and Monitoring Platform" ECOC 2018.
[Mor18]	Morro, R., Lucrezia, F., Gomez, P., et al., "Automated End to End Carrier Ethernet Provisioning over a Disaggregated WDM Metro Network with a Hierarchical SDN Control and Monitoring Platform", in Proceedings of 44th European Conference on Optical Communication (ECOC 2018), 23-27 September 2018, Roma (Italy).
[Mor18a]	F.J. Moreno-Muro, C. San-Nicolas-Martinez, E. Martin-Seoane, M. Garrich, P. Pavon-Marino, O. Gonzalez de Dios, V. López, "Joint Optimal Service Chain Allocation, VNF instantiation and Metro Network Resource Management Demonstration", in Proc. of OFC18, San Diego, USA, Mar. 2018.
[Mor18b]	F.J. Moreno-Muro, C. San-Nicolás-Martínez, M. Garrich, P. Pavon-Marino,O. González de Dios, R. Lopez Da Silva, "Latency-aware Optimization of Service Chain Allocation with joint VNF instantiation and SDN metro network control", in Proc. of ECOC19, Rome, Italy, Sep. 2018.
[Mor19]	F.J. Moreno-Muro, M. Garrich, C. San-Nicolás-Martínez, M. Hernández-Bastida, P. Pavón-Mariño, A. Bravalheri, A.S. Muqaddas, N. Uniyal, R. Nejabati, D. Simeonidou, R. Casellas, O. González de Dios, "Joint VNF and Multi-Layer Resource Allocation with an Open-source Optimization-as-a-Service Integration", submitted to ECOC 19, Dublin, Ireland, Sep. 2019.

[NFV]	Network Functions Virtualisation (NFV) in ETSI. https://www.etsi.org/technologies/nfv/nfv
[ODTN]	Open Networking Foundation (ONF), The Open Disaggregated Transport Network (ODTN) project, https://www.opennetworking.org/odtn/
[OpenConfig]	OpenConfig project and data models http://openconfig.net and https://github.com/openconfig/public/tree/master/release/models
[OpenROADM]	The Open ROADM Multi-Source Agreement (MSA) http://www.openroadm.org Accessed: 2019-04-12
[OSM]	Open Source NFV Management and Orchestration (MANO), https://www.etsi.org/technologies/nfv/open-source-mano
[OVS]	Open vSwitch Manual - database schema (ovs-vswitchd.conf.db) [Online]: http://www.openvswitch.org/support/dist-docs/ovs-vswitchd.conf.db.5.txt
[RFC6313]	B. Claise, G. Dhandapani, P. Aitken, and S. Yates, “Export of Structured Data in IP Flow Information Export (IPFIX),” IETF RFC 6313, 2011.
[RFC7011]	B. Claise, B. Trammell, and P. Aitken, “Specification of the IP Flow Information Export (IPFIX) Protocol,” IETF RFC 7011, 2013.
[RFC8040]	A. Bierman, M. Bjorklund, and K. Watsen, “RESTCONF Protocol,” IETF RFC 8040, 2017
[RFC8466]	B. Wen, G. Fioccola, C. Xie, L. Jalil, “A YANG Data Model for Layer 2 Virtual Private Network (L2VPN) Service Delivery,” IETF RFC 8466, 2018.
[Ric18]	Riccardi, E., Gunning, P, González de Dios, et al., ‘An Operator view on the Introduction of White Boxes into Optical Networks’, Journal of Lightwave Technology, 2018, 36, (15), pp. 3062-3072
[Ru16]	M. Ruiz, J. Ramos, G. Sutter, et al. “Accurate and affordable packet-train testing systems for multi-Gb/s networks’, IEEE Communicatons Magazine, 2016, 54, (3), pp. 80-87, DOI:10.1109/MCOM.2016.7432152
[TAPI]	Open Networking Foundation (ONF), Transport API project https://wiki.opennetworking.org/display/OTCC/TAPI Accessed: 2019-04-12
[Tro19]	Sebastian Troia, David Eugui, Ignacio Martín, Ligia Maria Moreira Zorello, Guido Maier, José Alberto Hernández, Oscar González de Dios, Miquel Garrich, José Luis Romero-Gázquez, Francisco-Javier Moreno-Muro, Pablo Pavón Mariño, Ramon Casellas, “Machine Learning-assisted Planning and Provisioning for SDN/NFV-enabled Metropolitan Networks”, accepted in EuCNC19, Valencia, Spain, June 2019.
[Tro19b]	S. Troia et al “Dynamic Virtual Network Function Placement over a Software-Defined Optical Network” OFC2019 Demo

end of document