5GXCrosshaul
the integrated fronthaul/backhaul

H2020 5G-Crosshaul project
Grant No. 671598

XFE/XCI design at year 1, specification of southbound and northbound interface

Abstract

This document presents a detailed description of the main element of the proposed control plane within the 5G-Crosshaul architecture, the 5G-Crosshaul Control Infrastructure (XCI), and introduces the 5G-Crosshaul Forwarding Element (XFE), described already in [1]. This document covers design, implementation, deployment considerations of its different elements, as well as the interaction of the XCI with the other architectural planes (data plane and application plane) of the Software-Defined Networking concept via the proposed Southbound Interface (SBI) and Northbound Interface (NBI) respectively. This document concludes with the presentation of the XCI validation methodology.

Document Properties

Document Number:	D3.1
Document Title:	**XFE/XCI design at year 1, specification of southbound northbound interface**
Document Responsible:	Thomas Deiß (NOK-N)
Document Editor:	José Núñez-Martínez (CTTC)
Target Dissemination Level:	Public
Status of the Document:	Final
Version:	1.0

Production Properties:

Reviewers:	Xi Li (NEC), Domenico Siracusa (CNET), Sean Chang (ITRI).

Document History:

Revision	Date	Issued by	Description
0.0	15-09-16	José Núñez-Martínez	First version of the document with all the contributions
0.1	11-10-16	José Núñez-Martínez	Review finished.
1.0	18-10-16	José Núñez-Martínez	Final editing.

Disclaimer:

This document has been produced in the context of the 5G-Crosshaul Project. The research leading to these results has received funding from the European Community's H2020 Programme under grant agreement Nº H2020-671598.

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

For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.

Table of Content

List of Contributors

Partner No.	Partner Short Name	Contributor’s name
P01	UC3M	Nuria Molner, Antonio de la Oliva
P02	NEC	Andres Garcia Saavedra, Xi Li
P05	ATOS	Jorge Rivas Sánchez, Jose Enrique González Blázquez
P06	NOK-N	Thomas Deiß
P07	IDCC	Luca Cominardi
P13	NXW	Giada Landi, Francesca Moscatelli, Marco Capitani
P17	CTTC	Josep Mangues, Manuel Requena José Núñez, Iñaki Pascual, Jorge Baranda, Josep Mangues, Ramon Casellas, Ricard Vilalta, Raül Muñoz, Ricardo Martínez.
P18	CREATE-NET	Leonardo Goratti, Domenico Siracusa, Raihana Ferdous
P19	POLITO	Claudio Casetti

List of Tables

List of Figures

List of Acronyms

Acronym	Description
ABNO	Application Based Network Operations
AF	Adaptation Function
API	Application Programming Interface
BBU	Base-Band Unit
BER	Bit Error Rate
BH	Backhaul
BS	Base Station
CDN	Content Delivery Network
CDNMA	Content Delivery Network Management Application
COP	Common Orchestration Protocol
COTS	Commercial Off-the-Shelf
CPRI	Common Public Radio Interface
CPU	Central Processing Unit
C-RAN	Cloud Radio Access Network
CRUD	Create, Read, Update, Delete
DB	Data Base
DWDM	Dense Wavelength Division Multiplexing
E2E	End to End
EMMA	Energy-Management and Monitoring Application
ETSI	European Telecommunications Standards Institute
FH	Fronthaul
GMPLS	Generalized Multi-Protocol Label Switching
ID	Identifier
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
LAN	Local Area Network
LLDP	Link Layer Discovery Protocol
LMS	Local Management Service
MAC	Media Access Control
MANO	Management and Orchestration
MMA	Mobility Management Application
MPLS	Multiprotocol Label Switching
MTA	Multi-Tenancy Application
MVNO	Mobile Virtual Network Operator
NBI	NorthBound Interface
NFV	Network Functions Virtualization

NFVI	Network Function Virtualization Infrastructure
NFV-O	Network Function Virtualization Orchestrator
NS	Network Service
NSD	Network Service Descriptor
ODL	OpenDayLight
ONF	Open Networking Foundation
ONOS	Open Network Operating System
OTN	Optical Transport Network
OTT	Over-The-Top
PBB	Provide Backbone Bridging
PBSS	Personal Basic Service Set
PCE	Path Computation Element
PCP	PBSS Code Point
PNF	Physical Network Function
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
REST	Representational State Transfer
ROADM	Reconfigurable Optical Add and Drop Multiplexer
RRH	Remote Radio Head
RRU	Remote Radio Unit
SBI	SouthBound Interface
SDK	Software Development Kit
SDN	Software-Defined Networking
SFC	Service Function Chaining
SLA	Service Level Agreement
SMF	Single-Mode Fibre
SNMP	Simple Network Management Protocol
SRLG	Shared Risk Link Group
SUT	System Under Test
TCAM	Ternary Content Addressable Memory
TDM	Time Division Multiplexing
TE	Traffic Engineering
UE	User Equipment
UML	Unified Modeling Language
URL	Uniform Resource Locator
VDU	Virtual Deployment Unit
vCDN	Virtual Content Delivery Network
vEPC	Virtual Evolved Packet Core
VIM	Virtual Infrastructure Manager

VIMaP	Virtual Infrastructure manager and Planner Application
VLAN	Virtual LAN
VM	Virtual Machine
VNF	Virtual Network Function
VNFC	Virtual Network Function Component
VNFD	Virtual Network Function Descriptor
VNFFG	Virtual Network Function Forwarding Graph
VNFM	Virtual Network Function Manager
VTN	Virtual Tenant Network
WP	Work Package
XAF	5G-Crosshaul Adaptation Function
XCF	5G-Crosshaul Common Frame
XCI	5G-Crosshaul Control Infrastructure
XCSE	5G-Crosshaul Circuit Switching Element
XFE	5G-Crosshaul Forwarding Element
XML	Extensible Markup Language
XPFE	5G-Crosshaul Packet Forwarding Element
XPU	5G-Crosshaul Processing Unit

1. Executive Summary

The 5G-Crosshaul project aims at developing a 5G integrated fronthaul and backhaul transport network solution, enabling a flexible interconnection of the radio access with the core network by software-defined reconfiguration of all network elements. In order to achieve this, there is a need for high capacity and low latency transmission techniques and novel unified data and control plane mechanisms.

This document provides a thorough description of the software-defined of the proposed 5G transport network solution: the 5G-Crosshaul Control Infrastructure (XCI). On top of the XCI the project is proposing the logic for managing the integrated fronthaul and backhaul infrastructure via an upper layer of external applications, which are described in D4.1 [2]. The XCI represents the Management and Orchestration (MANO) platform to operate all available types of resources: networking, computing, and storage. The architecture of the XCI is based on Software Defined Network (SDN) and Network Functional Virtualization (NFV) principles. To the upper layer, it provides a unified platform via a Northbound Interface (NBI) to control and monitor the underlying data plane by a common set of core services and primitives. The XCI interacts with the data plane via its Southbound Interface (SBI) to:

- control and manage the packet forwarding behavior of the 5G-Crosshaul forwarding elements (XFE) in the 5G-Crosshaul network;
- control and manage the physical configuration of the different link technologies; and
- control and manage the 5G-Crosshaul Processing Units (XPU) via NFV.

The XCI consists of a part dealing with NFV and specific SDN controllers for the different types of resources. The NFV part is aligned with ETSI NFV architecture. The SDN controllers handle the network elements in the data plane. 5G-Crosshaul extends the SDN controllers to handle the different link technologies, providing a common SDN controlled network substrate, which can be reconfigured based on the needs of the network tenants.

We expect the data plane to use different transmission technologies. Also, a part of the data plane may be a legacy system, not consisting of XFEs, whereas other parts may consist of XFEs. In both cases, the data plane consists of different technological domains. The different domains can be combined by SDN controllers in a peer or a hierarchical model. In the peer model, each SDN controller handles devices of multiple domains. In the hierarchical model, a parent controller combines the different domains into a single network and the dedicated child controllers handle a single technological domain. For the design of the data plane, WP3 has worked jointly with WP2, especially on the XFE and SBI design. A detailed description of the XFE and SBI has been

presented in D2.1 [1], covering both joint WP2 and WP3 as well as work done by WP3 only.

The SDN controllers provide different services to the NFV part of the XCI and upper layer applications, such as a topology and inventory service. Other services provide analytical or optimization algorithms. An example of an optimization algorithm is to setup paths through the network to use as few resources as possible, in turn allowing to reduce the power state of the unused resources. This deliverable presents several analytical models of the network used as a basis for optimization algorithms

The services of the controllers have to be accessible by the upper layers via its NBI. The NBI is REST based, following the paradigm used by many SDN and IT controllers. For each of the services, this document defines initial versions of the information model, the actual API, as well as workflows to use the API. If needed, these initial versions will be refined when implementing the services.

The components of the data and control plane will be integrated in the demonstrations of WP5. Each of these components has to be validated to ensure their maturity before the integration. We define a validation approach based on test cards, defining the scope of the individual tests and the test steps of each test. An overview of the tests is provided in the main part of this document, the detailed steps are defined in appendices.

2. Key achievements

The key achievements reported in this document are summarized in the following.

The **first** important **achievement** presented in this document is **a detailed description of** the central entity of the control plane architecture defined within the 5G-Crosshaul network, **the XCI**. This description focuses on its **different elements, interrelations and deployments considerations** within the 5G-Crosshaul network. This document identifies possible software platforms for XCI's implementation.

The **second** key **achievement** is **the initial definition of the NBI based on REST**, to allow interaction between the different modules of the XCI and with the different applications present in the application plane. This definition identifies the required NBI services present in the XCI and proposes a **specific API**, the more relevant information **data models** associated to this NBI service, and a **workflow** illustrating the use of this service by a generic 5G-Crosshaul application or by an internal module inside the XCI that, in turn, can expose an NBI.

Finally, the **methodology adopted for the validation and evaluation of the data plane and control plane components** under development in WP3 is presented. The scope of this validation activity is to assure the proper behaviour of the functions and algorithms that are integral part of the components constituting the whole XCI control-plane infrastructure and operating the heterogeneous resources in the data plane.

3. Introduction

This document provides a detailed description of the main elements of the proposed control plane within the 5G-Crosshaul architecture, see Figure 1, the 5G-Crosshaul Control Infrastructure (XCI). This description covers the design, implementation, and deployment considerations of its different elements, namely, the Network Function Virtualization (NFV) infrastructure, the Management and Orchestration (MANO) elements, and the Software Defined Networking (SDN) controllers.

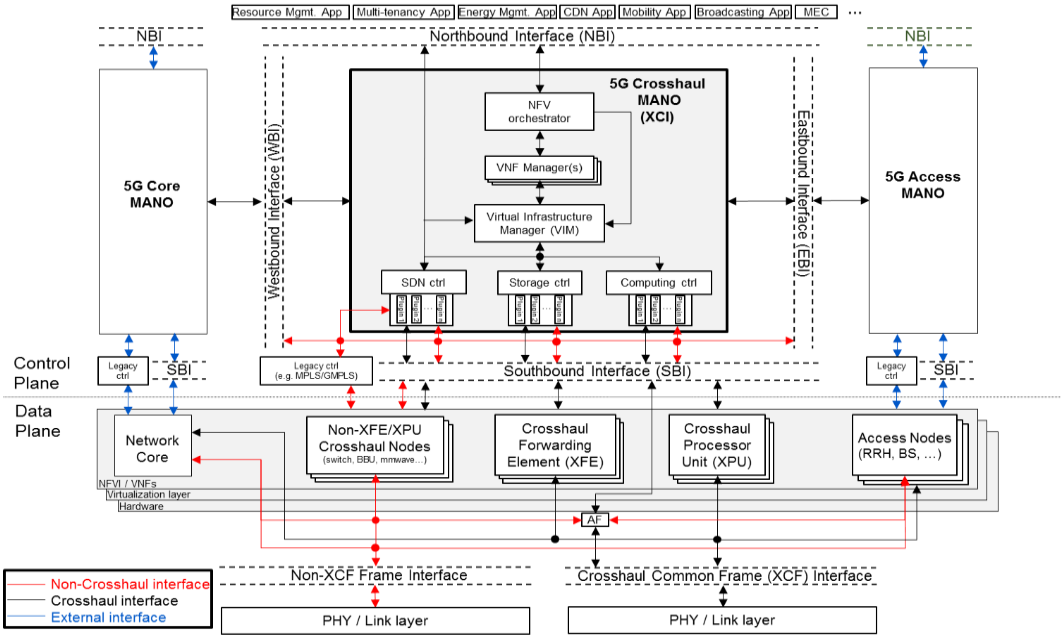


Figure 1: 5G-Crosshaul architecture illustration.

This document contains a short summary of the 5G-Crosshauling Forwarding Element (XFE) and the Southbound Interface among the XCI and the XFEs. This was joint work in collaboration with WP2 and is reported in more detail in D2.1 [1].

This description is completed with an explanation of the interaction of the XCI with the other architectural planes (data plane and application plane) of the SDN and NFV concepts by specifying a Southbound Interface (SBI) and Northbound Interface (NBI) respectively. Finally, this deliverable presents of the validation methodology and plans for the main 5G-Crosshaul components under development, not only the XCI but also the XFE software.

More in detail, the document is structured as follows:

- Section 4 summarizes the unified data plane architecture proposed within the 5G-Crosshaul project, focusing on the architecture of the XFE.
- Section 5 describes in detail the proposed control plane architecture, focusing on the different elements that constitute the 5G-Crosshaul XCI, their interrelations,

XCI’s different deployment considerations required within the scope of proposed 5G-Crosshaul network, and the identification of possible software frameworks and platforms to be used for their implementation.

- Section 6 defines analytic algorithms adopted at the XCI level. Further algorithms implemented at the application layer are addressed in WP4 and are out of scope of this document.
- Section 7 and Section 8 describe the design of the SBI and the NBI, where the former is required by the XCI to interact with the data plane and the latter is used by the XCI to interact with the application plane.
- Section 9 presents the validation methodology and plans of the different 5G-Crosshaul data plane and control plane elements: XFE prototypes, XCI MANO components and XCI SDN controllers.
- Section 10 provides the main conclusions of the activity so far developed within WP3.
- Appendix I and II, in Section 11 and Section 12, respectively, provide further detail procedures of the validation methodology summarized in Section 9 for the XCI components.

4. Data Plane Architecture

The data plane architecture has already been described in D2.1 [1]. For the sake of self-containment, this section provides a summary of the more relevant elements in the data plane architecture. The logical architecture of the 5G-Crosshaul data plane is illustrated in Figure 2.

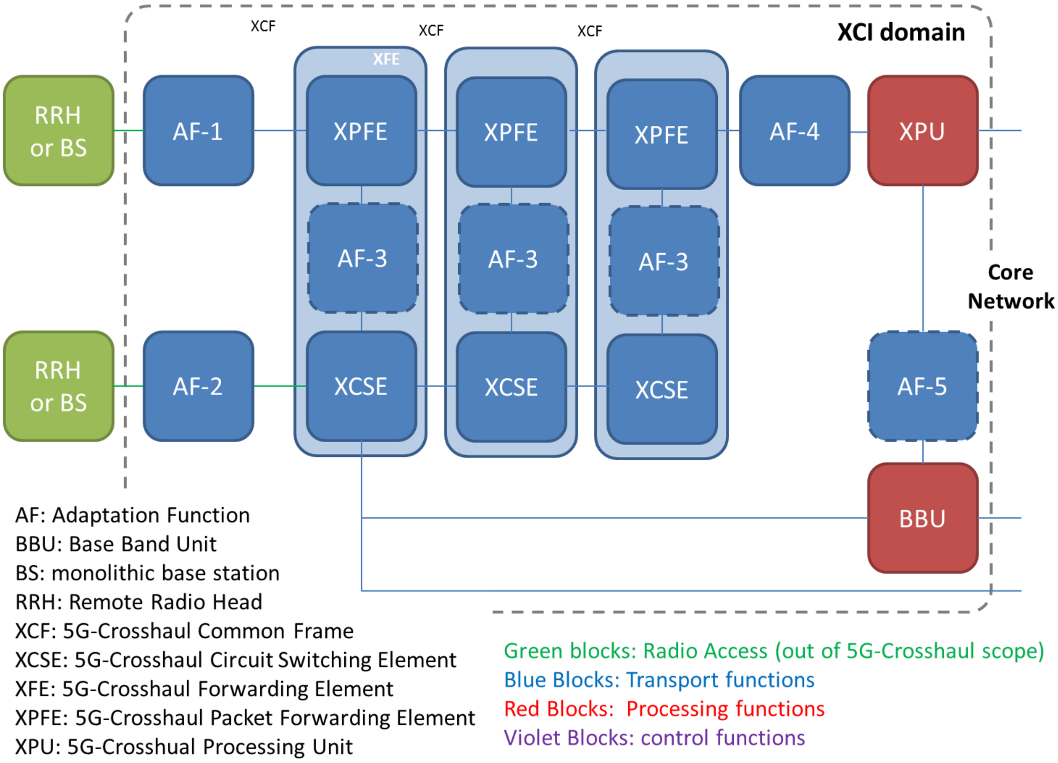

XCF
XCF
XCF
XCI domain
XFE
RRH or BS
AF-1
XPFE
XPFE
XPFE
AF-4
XPU
AF-3
AF-3
AF-3
Core Network
RRH or BS
AF-2
XCSE
XCSE
XCSE
AF-5
BBU
AF: Adaptation Function
BBU: Base Band Unit
BS: monolithic base station
RRH: Remote Radio Head
XCF: 5G-Crosshaul Common Frame
XCSE: 5G-Crosshaul Circuit Switching Element
XFE: 5G-Crosshaul Forwarding Element
XPFE: 5G-Crosshaul Packet Forwarding Element
XPU: 5G-Crosshual Processing Unit
Green blocks: Radio Access (out of 5G-Crosshaul scope)
Blue Blocks: Transport functions
Red Blocks: Processing functions
Violet Blocks: control functions

Figure 2: 5G-Crosshaul data plane architecture.

The fundamental block of the data plane architecture is the XFE that, in the most general implementation, is a multi-layer switch, made up of a packet switch called the 5G-Crosshaul Packet Forwarding Element (XPFE) and a circuit switch called the 5G-Crosshaul Circuit Switching Element (XCSE), see Figure 3. This two layer switching architecture is able to combine the bandwidth efficiency given by statistical multiplexing in the packet switch, with the deterministic latency ensured by the circuit switch.

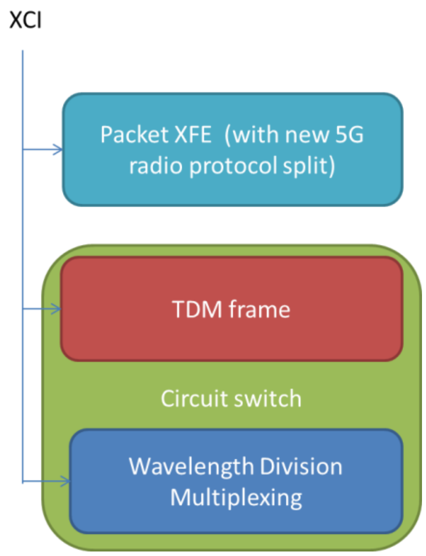

XCI
Packet XFE (with new 5G radio protocol split)
TDM frame
Circuit switch
Wavelength Division Multiplexing

Figure 3: Generic implementation of the 5G-Crosshaul switching node

It is not necessary that all the layers always coexist but one or two could be skipped depending on the type of deployed network. Examples are: a mesh of packet switches connected by dark fibers (where only the packet layer is exploited); 5G remote radio heads (RRHs), based on new radio protocol split and packetized fronthaul interface, connected to a Dense Wavelength Division Multiplexing (DWDM) network (where only wavelength and packet switch are present); the same network where also Common Public Radio Interface (CPRI) tributaries are carried and multiplexed over time-slots in a wavelength, so that a Time Division Multiplexing (TDM) switch needs to be added. This TDM switch can be based on optical transport network (OTN) or the simpler framing protocol described in D2.1 [1].

The 5G-Crosshaul Common Frame (XCF) is a packet interface based on an evolution of the Ethernet MAC-in-MAC standard, adding mechanisms to deal with time sensitive applications, see Section 8 of D2.1 [1]. The XPFEs talk to each other using the XCF. XCF is also the interface between XPFE and 5G-Crosshaul Processing Unit (XPU), the virtualized unit in charge of hosting baseband processing and other virtual functions. Packet switching enables statistical multiplexing when the peak to average radio access traffic load in 5G is high enough. Figure 4 depicts an initial functional architecture for the 5G-Crosshaul Packet Forwarding Element (XPFE).

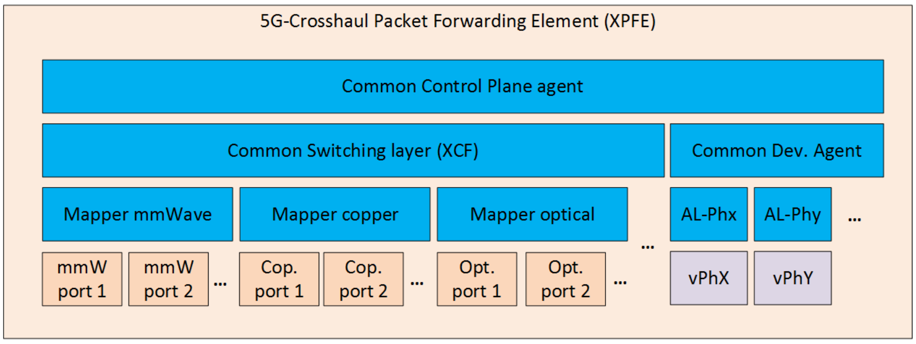


Figure 4: XPFE functional architecture

The XPFE includes the following key functions:

- A common control-plane agent to talk to the common control infrastructure (XCI).
- A common switching layer based on a common frame (XCF) to forward packets between interfaces.
- A device agent common to all peripheral systems to talk with system components. This agent exposes device-related information like CPU usage, RAM occupancy, battery status, GPS position, etc., to the control infrastructure.
- Mappers for each physical interface. XCF can be mapped on any physical interface as long as the XCF traffic requirements are satisfied.
- Physical interfaces to transmit the data on the link.

Figure 5 depicts an initial functional architecture for the 5G-Crosshaul Adaptation Function (XAF), similar to the XPFE. It includes the following key functions:

- A common control-plane agent, a common switching layer, a common device agent, mappers, and physical interfaces like in XPFE case.
- Adaptation layers from/to the common switching layer to/from the specific fronthaul and backhaul protocols.
- Fronthaul and backhaul protocols (Ethernet, NGFI, CPRI, etc.).

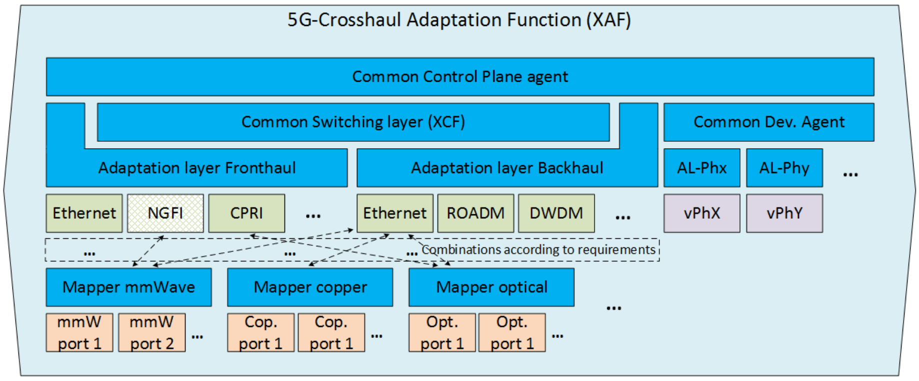


Figure 5: XAF functional architecture

The **adaptation layers** are in charge of translating/adapting fronthaul and backhaul protocols to XCF and enforcing the XCF forwarding control by adapting/translating the commands to specific protocol interfaces. The following is a non-exhaustive list of functions that will be defined in the adaptation layers:

- Mapping of fronthaul/backhaul traffic characteristics to the XCF format.
- Encapsulation and decapsulation of fronthaul and backhaul while dejittering and retiming the associated traffic.
- Framing of fronthaul and backhaul traffic with particular attention to the frame size in order to minimize delay, jitter, and to avoid fragmentation.

5. Control Plane Architecture

In this section, we present our initial design of the 5G-Crosshaul Control Infrastructure (XCI) platform. We can highlight that its design is led by the guidelines described in D1.1 [3] within the System Architecture task. Figure 1 illustrates the baseline 5G-Crosshaul system architecture presented in [3]. We divide the control plane into two layers: a top layer for external applications (duly described in D4.1 [2]) and the 5G-Crosshaul Control Infrastructure (XCI) below. The XCI is our 5G Transport Management and Orchestration (MANO) platform that provides control and management functions to operate all available types of resources (networking, computing, and storage). The XCI is based on the SDN/NFV principles and provides a unified platform that can be used by upper layer applications via a Northbound interface (NBI) to program and monitor the underlying data plane through a common set of core services and primitives. As mentioned in D1.1 [3], the XCI interacts with the data plane entities via a Southbound interface (SBI) in order to:

1) Control and manage the packet forwarding behavior performed by 5G-Crosshaul Forwarding Elements (XFEs) across the 5G-Crosshaul network;

2) Control and manage the PHY configuration of the different link technologies (e.g. transmission power on wireless links); and

3) Control and manage the 5G-Crosshaul Processing Units (XPU) computing operations (e.g. instantiation and management of Virtual Network Functions (VNFs) via Network Function Virtualization (NFV)).

5.1. High-level Architecture (5G-Crosshaul MANO)

In the following, we describe in detail the 5G-Crosshaul main architecture building blocks within the control plane briefly introduced above. The XCI is the brain controlling the overall operation of the 5G-Crosshaul. The XCI part dealing with NFV comprises three main functional blocks, namely: NFV orchestrator, VNF Manager(s) and Virtual Infrastructure Manager (VIM) (following the ETSI NFV architecture [4]):

- The **NFV-O (NFV Orchestrator)** is a functional block that manages a Network Service (NS) lifecycle. It coordinates the VNF lifecycle (supported by the VNFM) and the resources available at the NFV Infrastructure (NFVI) (supported by the VIM) to ensure an optimized allocation of the necessary resources and connectivity to provide the requested virtual network functionality.

- The **VNFMs (VNF Managers)** are functional blocks responsible for the lifecycle management of VNF instances (e.g. instance instantiation, modification, and termination).
- The **VIM (Virtualized Infrastructure Manager)** is a functional block that is responsible for controlling and managing the NFVI computing (via *Computing ctrl*), storage (via *Storage ctrl*) and network resources (via *SDN ctrl*).

In addition to these modules, which are in charge of managing the different VNFs running on top of the 5G-Crosshaul, the XCI includes a set of specialized controllers to deal with the control of the underlying network, storage and computation resources:

- SDN Controller: This module is in charge of controlling the underlying network elements following the conventional SDN paradigm. 5G-Crosshaul aims at extending current SDN support of multiple technologies used in transport networks (such as micro-wave links[1]) in order to have a common SDN controlled network substrate which can be reconfigured based on the needs of the network tenants.
- Computing/Storage Controllers: Storage and Computing controllers are included in what we call a Cloud Controller. A prominent example of this kind of software framework is OpenStack.

Note that the **SDN/Computing/Storage controllers** are functional blocks with one or multiple actual controllers (hierarchical or peer-to-peer structure) that centralize some or all of the control functionality of one or multiple network domains. We consider the utilization of legacy network controllers (e.g. MPLS/GMPLS) to ensure backward-compatibility for legacy equipment.

5.1.1. Multi-domain control

The multi-domain transport control is a relevant aspect to consider in 5G-Crosshaul both to enable the interaction of SDN with legacy control and to support the case where more SDN controllers should interwork. We will report later in Section 5.3 a description of the different models of control interaction (e.g. peer and hierarchical), providing their comparison in different network scenarios. In this section, instead some key requirements for the extensions of the 5G-Crosshaul MANO architecture to include multi-domain transport are reported. This is actually a relevant topic that is currently debated within the main standardization bodies though a conclusion has not been reached yet. For example the IETF ACTN BoF [5] proposes a hierachical architecture for the multi-domain transport but it is focused on the transport aspects with very limited description about the interaction with the virtualization functions. In [4] instead, most of the work is concentrated to the virtualization of the functions considering the transport as available Point to Point resources, without considering the constraints

[1] The ONF is actively working towards the definition of a southbound interface for micro-wave links: http://5g-crosshaul.eu/wireless-transport-sdn-proof-of-concept/

related to the interaction among different domains. In most cases the interaction between multi-domain transport and virtualization is solved considering a sort of overprovisioning of the networking resources, but that could be a solution not viable in future due to the need of reducing the cost of the transport. A practical solution to fix such issue is to define a layer architecture that assures a complete decoupling between the multi-domain transport and the virtualization layers. According to this principle the multi-domain transport is in charge of optimizing the network resources, assuring a suitable interworking among the heterogeneous transport domains, and providing a suitable exposition of the transport resources to the virtualization layer (e.g. to NFV orchestrator, VNF-I). The virtualization layer, instead, will operate on the network resources working on the suitable abstract view of the transport. In this model the multi-domain transport should provide the resource exposition according to service level agreement (SLA) hiding the technological details of the several domains and simplifing the tasks of the virtualization layer. Moreover, this model makes the management of the virtualizacion functions agnostic to the evolution of the several transport domains from legacy to the SDN. Consequently, this allows the XCI architecture to be quite general and applicable in concrete scenarios where operators can move towards pure SDN architecture smoothly.

The decoupling between the multi-domain transport and virtualization prevents any dependency of the general architecture of the XCI on the specific transport technology, and simplifies the interaction also with the legacy control. Anyway, some issues have to be fixed to make the solution efficient. For example, the virtualized view provided by the multi-domain transport should be quite stable in time, limiting the variation of parameters and facilitating the task of the NFV-O. Again, this could be in contrast with the resource optimization techniques applied on the transport layer where, continuous change of information and data could be necessary. Actually, the challenge is to jointly meet all previous requirements taking into account NFV and SDN reference paradigm. 5G-Crosshaul will address such topics in details in the rest of the project.

Figure 6 shows how to extend the XCI control architecture with multi-domain transport control. In the picture it is highlighted the border between the virtualization and transport layer represented by the abstract view of the transport resources, this border is within the end-to-end (E2E) abstract exposition of the transport resources. Actually, the multi-domain transport control is responsible to provide the E2E abstract view based on SLA parameters and to guarantee that such values are met; while the virtualization layer utilizes the E2E abstract view as networking resources to be combined with storage and computation according to the procedures defined in [4].

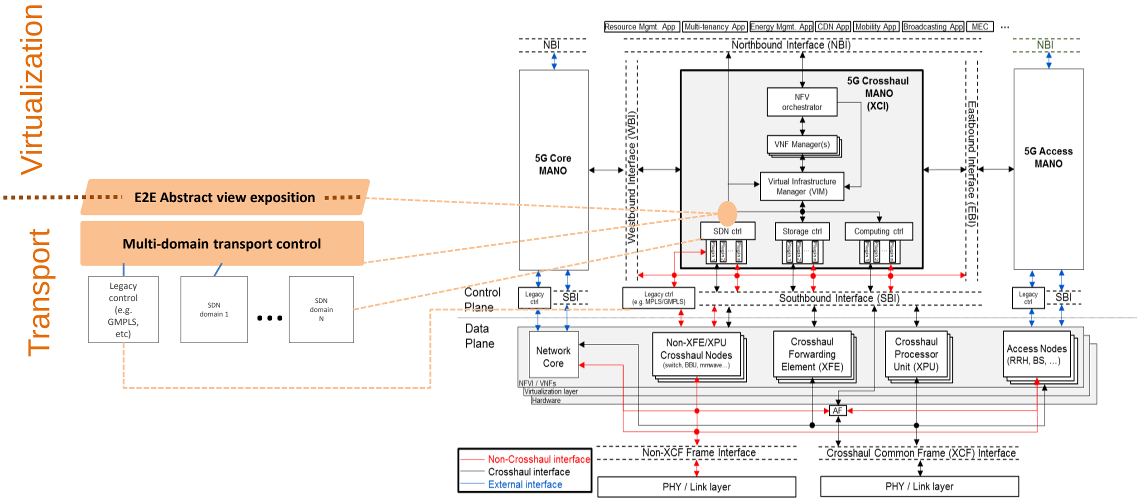


Figure 6: Multi-domain XCI architecture.

5.1.2. Multi-tenant design (XCI recursion)

The main discussion and agreements on the design of a hierarchical recursive XCI are presented in Section 3.3 of D1.1 [3]. We can report here that no substantial advances on this topic have taken placed since then, and thus we refer the reader to D1.1 [3].

5.2. XCI Design

Figure 7 depicts the XCI design, also showing its interactions with the 5G-Crosshaul SDN applications and VNFs, which are out of WP3 scope (represented in the green boxes).

The picture highlights the two macro-modules of the XCI:

- The XCI MANO components, responsible for the instantiation, orchestration and management of Virtual Network Functions and Network Services (in ETSI NFV terminology, a Network Service is a collection of VNFs interconnected through a VNF Forwarding Graph – VNFFG – and it can be considered as the equivalent of a Service Function Chain);
- The XCI SDN controller, responsible for the configuration and management of the network infrastructure.

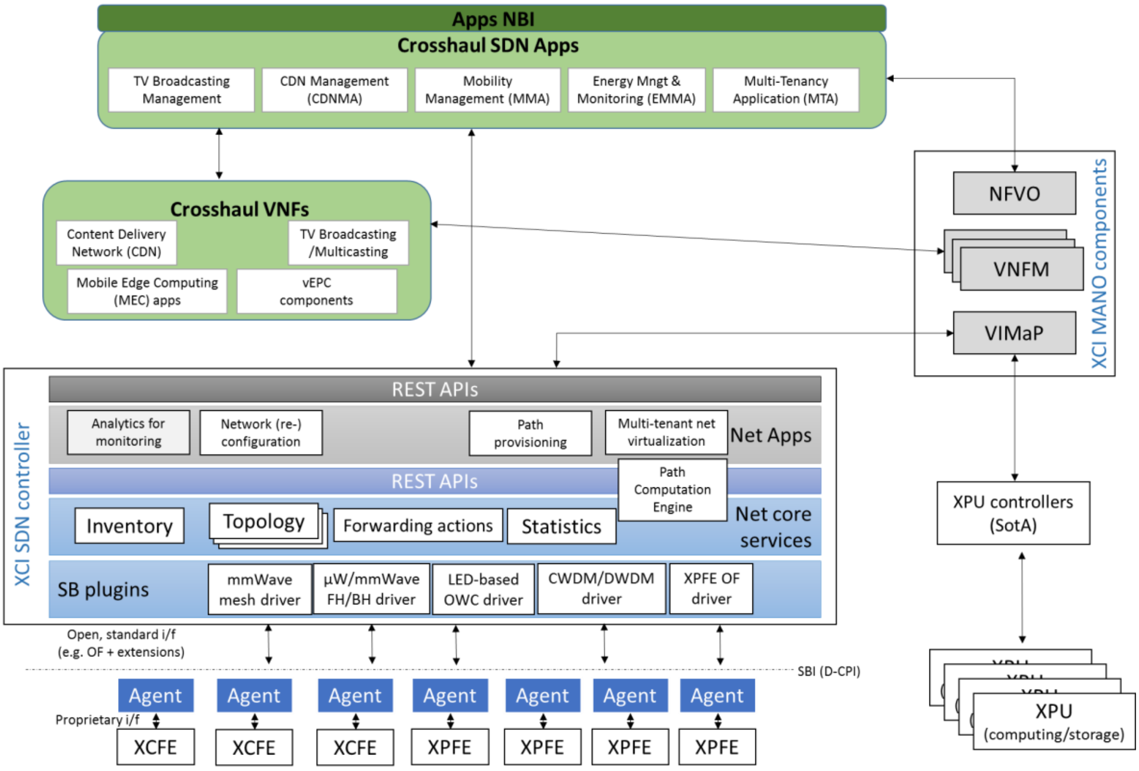


Figure 7: XCI design.

As explained in the previous section, the XCI MANO components include the NFV Orchestrator (NFV-O), the VNF Managers (VNFM) associated to the different 5G-Crosshaul VNFs, and the Virtual Infrastructure Manager (VIM). In 5G-Crosshaul, the VIM concept is extended with planning algorithms, which take efficient decisions about virtual machines (VMs) placement and network configuration, towards integrated Virtual Infrastructure Management and Planning (VIMaP) functions. The controllers, in particular the SDN controller on the network side and the XPU controllers, including both storage and computing, perform the enforcement of VIMaP decisions. XPU controllers rely on State of the Art components (e.g. OpenStack NOVA for computing controllers) and are out of scope for 5G-Crosshaul.

It should be noted that in this section we are assuming a single network domain, thus operated by a single SDN controller. In case of a network infrastructure deployed in multiple domains, the general considerations described in Section 5.3 should be applied. In particular, the role of the SDN controller should be decomposed in several “child” controllers operating at each domain and abstracting the internal details of the local resources, with a hierarchical “parent” controller in charge of computing and allocating end-to-end and inter-domain connections. This is done through the coordination of the lower controllers’ actions, which are the final responsible of the actual, technology-dependent resource configuration. Further details about the applicability of the SDN hierarchical approach is provided in Section 5.3.2, where different deployment models are investigated and compared.

The XCI SDN controller macro-architecture is structured in three internal layers: *SBI drivers* for the interaction with heterogeneous devices at the data plane; *network core services* implementing basic monitoring, configuration and inventory functionalities; and *network applications* implementing the network level logic. Both network core services and network applications provide north-bound interfaces (NBI) which can be used by the XCI MANO components (mainly the VIMaP) and the 5G-Crosshaul applications to program the network.

The list of network applications embedded in the SDN controller are the following:

- Analytics for monitoring: an advanced monitoring service used to correlate raw network monitoring data originally provided by the statistics service. It can be used, for example, to elaborate monitoring information related to a virtual network based on statistics data on flows and physical ports. This functionality offers a NBI, which is detailed in Section 8.8, to upper-layer applications.
- Network (re-)configuration: used to re-configure specific network elements, mainly for management purposes. It is also used by the EMMA application to change the status of the devices for energy saving issues. This functionality is embedded in the Local Management service, whose NBI is detailed in Section 8.9.
- Path provisioning: service which establishes a network path between a source and one (or more) destination end-point(s). It can take as input several constraints, including the specification of the path itself. This option allows upper layer applications to implement their own allocation algorithms and use the path provisioning service as a sort of “configuration arm”. If the path is not included in the input parameters, the internal path computation engine core service is invoked. This functionality offers a NBI detailed specification in Section 8.2.
- Multi-tenant network virtualization: service which builds isolated and virtualized networks over the shared physical infrastructure. The multi-tenancy logic (i.e. the mapping between a Virtual Infrastructure –VI– and its tenant) can also be kept at the SDN controller level as optional, but it may be also implemented at the upper layers only (e.g. at the VIMaP, at the NFV-O or at the Multi-Tenancy Application (MTA) level). This is related to an on-going discussion in WP4. For what concerns WP3, the mandatory features of the network virtualization service are the consistency between the VI description and the VI exposed at the SDN controller NBI level and the isolation between coexisting VIs.

The SBI drivers have been specialized according to the latest outcomes of WP2 which has identified the different technologies of the 5G-Crosshaul data plane. Each SBI driver implements mechanisms for receiving inventory and monitoring data from the devices, including technology specific parameters, and for configuring some of their management parameters and their forwarding behavior. Further details are provided in the Section 7.

The following sections provide a preliminary matching between the components identified in the XCI design and possible software frameworks and platforms that can be used for their implementation.

5.2.1. Alternatives for NFV-O/NFVM/VIM

Some open source software alternatives for the XCI MANO components have been analyzed in D1.1 [3]. The XCI architecture does not mandate any specific software platform; the same XCI functions can be developed starting from different open source or proprietary projects, with the unique constraint of being compliant with the interfaces defined at the NBI and SBI of each component in order to support the proper workflows and interactions.

Initial proof of concept prototypes planned by different 5G-Crosshaul partners will be based on the components described in the following table:

Table 1: Software platforms used in XCI MANO prototypes.

Functional Component	Software baseline	Features / Use case	5G-Crosshaul partner
NFV-O	OpenBaton [6]	Orchestration of virtual Evolved Packet Core (vEPC).	NXW
	Proprietary orchestrator for CDNMA	Orchestration of CDN nodes.	ATOS
VNFM	OpenBaton VNFM SDK (REST API)	Management of vEPC VNFs lifecycle.	NXW
	Proprietary VNFM for CDN VNFs	Orchestration of CDN nodes VNFs lifecycle.	ATOS
VIMaP	OpenStack [7]	Provisioning of vEPC VNFs and their interconnections with Quality of Service (QoS) and energy constraints.	NXW
	OpenStack	Provisioning of CDN origin and replica servers on XPUs and Service Function Chaining (SFC) configuration.	ATOS
	OpenStack + proprietary	Allocation and management of VMs and their interconnections.	CTTC

5.2.2. Alternatives for storage and computing control

Storage and computing controllers will be based on state-of-the-art components, for example based on the corresponding OpenStack modules. No further extensions are required.

5.2.3. SDN controllers

As in the XCI MANO components case, the architecture does not impose any specific choice on the SDN controller software platform for the reference implementation. An analysis of possible alternatives has been provided in D1.1 [3].

The table below reports the plans of the 5G-Crosshaul partners for the implementation of proof-of-concept prototypes.

Table 2: Software platforms used in XCI SDN controller prototypes.

Functional Component	**Software baseline**	**Features / Use case**	**5G-Crosshaul partner**
SBI driver	OpenDayLight (ODL) [8]	XPFE forwarding control via OF. Collection of energy consumption parameters from XPFE. Configuration of XPFE device state.	NXW
	Ryu [9]	Configuration and control of mmWave nodes, based on OF for forwarding and REST for retrieval and configuration of port parameters.	CTTC
	ODL	Collection of energy consumption parameters.	POLITO
	ODL	Configuration, control and management of mmWave mesh nodes. Forwarding control via OF for mmWave mesh nodes.	IDCC
Network core services (inventory, topology, statistics, flow actions)	ODL	Extensions to topology & inventory modules for power-consumption information.	NXW
	Ryu	Core services for mmWave technology.	CTTC
	ODL	Core services for mmWave mesh nodes.	IDCC
Path computation	ODL	Path computation for energy-efficient network path	NXW
	--(analytical algorithms)	Network optimization	UC3M
	--(analytical algorithms)	Algorithms for optimal path computation and service embedding in multi-technology transport network (ETH + mmWave).	CREATE-NET
	ONOS[10]/ODL	Computation of path between RRHs and XPUs	NEC
Path provisioning	ODL	Path provisioning for energy-efficient network path	NXW
	ODL/ONOS	Provisioning of SFC networking in support of CDN infrastructure	ATOS
Network re-configuration	ODL	NBI methods to change the status of XPFE (in support of EMMA app)	NXW
Analytics for	ODL	Elaboration of energy-related	NXW

monitoring		monitoring information for paths and virtual infrastructures.	
Parent SDN controller (see section 5.3)	Proprietary	ABNO-based parent controller with COP (open source YANG-based) interaction with child SDN controllers.	CTTC
	ODL/Ryu	Distributed on-board/in-vehicle controller and on-land controller.	ITRI

5.3. Deployment models of XCI

In this section, we will discuss different deployment instances that can fit within our XCI architectural design. The following gathers the discussion taken place within the project on different models to control and manage networking resources. The extension to other types of resources (e.g. computing and storage) can be discussed in subsequent reports. It is commonly accepted that deploying a single, integrated controller for a large or complex network may present scalability issues, or may not be doable in practice. In particular:

- The network size, in terms of controllable elements, has a direct impact on the controller requirements on aspects such as the number of e.g. active and persistent TCP connections on top of which control sessions are established, memory requirements to store in memory e.g. a data structure representing the network graph abstracting the network and CPU requirements for processing message exchange, or implement control logic.
- The network complexity in terms of multiple deployed technologies (such as a packet-switched layer for Layer2/Layer3 transport over a circuit-switched optical layer, each having intrinsic and non-generalizable parameters and attributes) has an impact on functionalities and protocols to be implemented by the controller. For example, at the south-bound interface the controller needs to implement protocol extensions depending on the specific network layer of the controlled elements. Moreover, an inter-layer coordination function is also needed to deal with end-to-end connections and associated inter-layer technology adaptation, increasing the complexity of such unique controller.

To address such shortcomings, a current trend within SDN control plane design is to consider the deployment of multiple controllers, arranged in a specific setting, along with inter-controller protocols. Such architectures apply both to heterogeneous and homogeneous control (different or same control plane and data plane technologies within the domain of responsibility of a given controller). As detailed next, a straightforward scheme to arrange the controllers is either in a flat (peer) or hierarchical setting, but we will later qualify and challenge such simple model.

It is important to state that nothing precludes the deployment of two or more SDN controllers for a given set of controlled network elements for robustness and availability purposes as depicted in in Figure 8. For example, the OpenFlow protocol supports the notion of primary or main and backup or secondary controllers. Two or more controllers can cover the same, overlapping or disjoint sets of network elements (this is a straightforward deployment choice for high availability reasons, where usually only a controller, e.g. the master one, has control over a given resource at a given time). For simplicity, from now on we assume a single SDN controller covering or spanning a set of controlled network elements. Having two or more controllers (e.g. for redundancy purposes) adds additional considerations such as inter-controller synchronization, and whether such controllers are synchronized e.g. by means of a dedicated protocol between them or via/by virtue of obtaining the information from the same set of network elements.

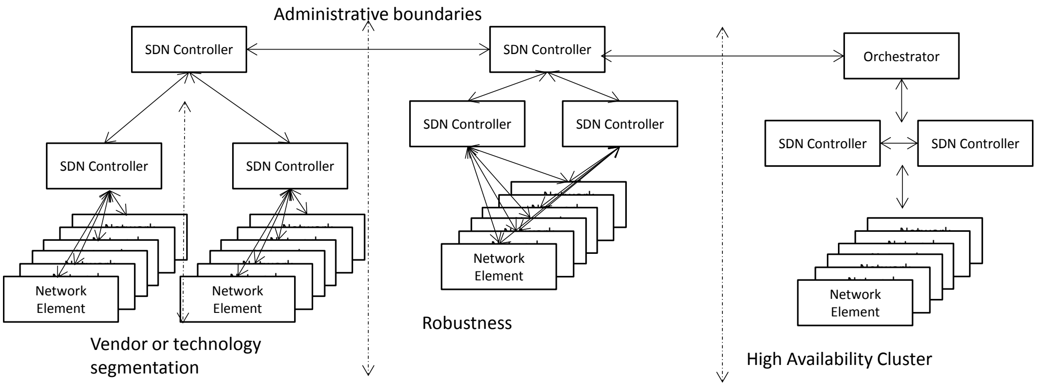


Figure 8: Arrangements for SDN controllers, combining hierarchical settings and peer models.

5.3.1. Basic SDN controller interconnection models

From a very basic, simplistic approach, SDN controllers can be arranged in two canonical models: peer or flat model and hierarchical model (Figure 8).

5.3.1.1. Peer or flat model

This usually corresponds to a set of controllers, interconnected in an arbitrary mesh, which cooperate to provision end-to-end services. In this setting, we can often assume that the mesh is implicit by the actual (sub)domains connectivity. The controllers hide the internal control technology and synchronize state using e.g. East/West interfaces. Further, the controllers manage detailed information of their own, local topology and connection databases, as well as abstracted views of the external domains and the East/West interfaces should support functions such as network topology abstraction, control adaptation, path computation, and segment provisioning.

5.3.1.2. Hierarchical model

In this model, controllers are ranged in a topology, which is typically a tree with a given root being the top-most controller. For a given hierarchy level, a centralized “controller of controllers” or orchestrator (also referred to as parent controller) handles the automation and has a certain number of high-level functions, while low-level controllers (usually referred to as children) cover low-level, detailed functions and operations. A recurring example is a 2-level hierarchy in which a parent SDN controller is responsible for connectivity provisioning at a higher, abstracted level, covering inter-domain aspects, while specific per-domain (child) controllers map the abstracted control plane functions into the underlying control plane technology. Proper interfaces and protocols are needed to enable this interaction between child and parent controllers; more generic interfaces and protocols enable a wider applicability of the architecture to an arbitrary number of hierarchy levels.

5.3.2. Generalizing hierarchical SDN controller interconnection models

In technological contexts such as the one defined by 5G-Crosshaul, there are multiple considerations that challenge the somehow simplistic hierarchical models. Let us identify a non-exhaustive list, noting that they are also strongly inter-dependent, since, for example, different network segments may belong to one or more administrative domains/operators who internally arrange the network in technological domains, which, in turn, are commonly provided by different vendors.

- **Network segment splitting.** Defining an SDN control architecture for a network that encompasses multiple network segments (such as access, aggregation, metro, core) may be constrained in what concerns whether it is feasible to deploy a hierarchical SDN control or not.
- **Vendor constraints.** Arranging controllers in a specific setting will highly depend on the available interfaces and protocols (ideally open and standard) and the corresponding level of support/implementation for a given vendor. It is reasonable to expect that arranging SDN controllers from the same vendor in a hierarchy will be straightforward if the vendor provides such functionality and there will be a high level of expected inter-operability in that case.
- **Redundancy, high availability, robustness.** It is accepted that deployments of SDN control in carriers’ networks will be strongly constrained by the expected requirements in terms of robustness and high-availability. Best common practices commonly consider deploying multiple elements and synchronizing state between them. This is sometimes referred to as fat-trees, or forests, e.g. where the parents communicate to each other.
- **Widest definition and scope of “domain”.** Strongly related to the previous ones, the term domain has multiple definitions and sometimes applies to the arrangement

of network elements by their technology but also by their vendor or network segment, or even administrative or geographical domains.

- **Fitness for purpose.** A specific controller arrangement may not fully correspond to the intended logical SDN controller relationships, which sometimes can be better mapped to a client/server or master/slave model. While master/slave can correspond to a hierarchical setting, other relationships, functional splits and responsibilities may fit better to a flat model
- **Administrative domains, control and ownership.** An often-recurring critique of pure hierarchical models comes from the issue of top-most parent ownership. Unless there is a clear function definition and demarcation points, business arrangements and inter-connection models are based on a peer relationship in which no entity is under the control or supervision of a higher-level entity.
- **Confidentiality.** This applies to either peer or hierarchical models, although depends on the specifics of the northbound and west/east interfaces.
- **Domains of applicability.** The initial designs for hierarchical models addressed the problem of arranging SDN controllers considering only e.g. the networking and data communications for network service provisioning aspects. When considering, as in 5G-Crosshaul, the need to offer 5G services that involve heterogeneous resources beyond the network (i.e. also storage and computing resources), the adoption of a hierarchical, peer or hybrid model is not clear. A set of network or cloud controllers may be under the control of an ETSI NFV VIM, or the VIM may include a cloud controller that includes a network controller, and combinations hereof.
- **Provisioning workflows.** Intended provisioning workflows may also affect the choice of hierarchical or peer models. For example, “end user driven”, “data driven” or “event-driven” provisioning services may be better suited to a peer model (e.g. a request from the Radio Access Network (RAN) to the core) while a “operator-driven” pre-provision action may be better suited to a hierarchical model.

A main, direct consequence of the previous consideration and analysis, is that, in general, a given deployment of a carrier class SDN-based control architecture for 5G services (which combine heterogeneous networking/cloud resources over an infrastructure spanning multiple network segments and/or technological domains and vendors) will not correspond to a pure hierarchical or flat but will present a combination of centralized/distributed and hierarchical/flat/peer models constrained by the identified requirements and actual implementation choices (see Figure 9).

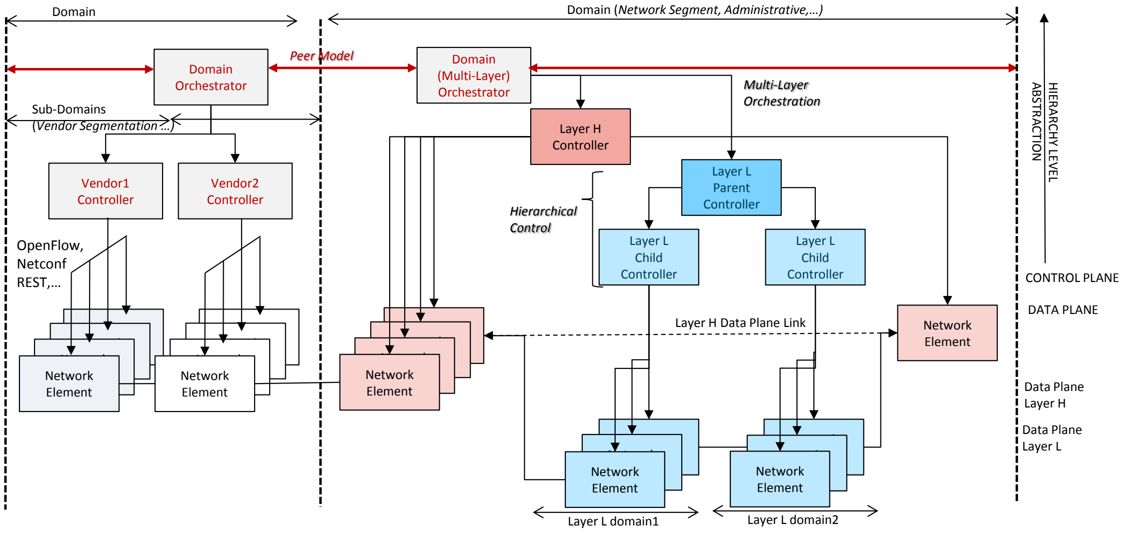


Figure 9: Example of SDN control within a mixture of administrative, technological and vendor domains, showing different peer and hierarchical modes.

5.3.2.1. Hierarchical SDN approaches based on the definition of domain

As a summary of the previous section, constraining aspects that may condition the deployment of hierarchical SDN controllers are mainly defined by the widest definition of domain and the associated requirements of confidentiality, functional split and interconnection and business agreements. The following table summarizes and illustrates the main cases.

Case examples Deployments	**Remarks in Hierarchical SDN architectures.**
Same vendor within a given technology and administrative domain	• Best-fit model depends on vendor defined criterion; e.g. peer or hierarchy and abstraction support provided by the SDN vendor. • Straightforward and interoperable hierarchical SDN setting • Straightforward and interoperable peer SDN setting • Hierarchy introduced for scalability reasons • Within a hierarchical setting, peer-models may be used for robustness, or high-availability reasons (e.g. backup controller, distributed systems acting as a logically centralized entity) • Peer models commonly used when adhering to a distributed approach (e.g controller mesh, redundancy or high availability solutions) • Hierarchical or peer architectures work at low-level interfaces with binary encodings and byte-level protocol. High degree of interworking, when applicable, proprietary extensions used. • Hybrid approaches complementing peer/hierarchical
Different vendors within a single technology and administrative	• Peer models available (e.g. GMPLS controllers) but with strict constraints in interoperability. Issues in real deployments where operators chose to scope and segment into vendor islands.

domains	• Hierarchy as a means to orchestrate multiple vendors, scope inter-operability to a limited subset of interfaces and protocols, minimize risks. • Hierarchy done by means of "plugins" constrained to what is available or offered by vendors NBIs. Adopt an "NBI" standard if available and agreed upon.
Multiple Network Technologies within single administrative domains	• Peer models very constrained, requiring complex implementations and frameworks (e.g. GMPLS Multi-layer and Multi-region networks). • Peer model not adapted to a market where vendors cover mostly a horizontal technology or network segment • Suitable hierarchical models in which an "orchestrator" coordinates topology management and service provisioning o Example: IP over optical. • Hierarchy done by means of "plugins" constrained to what is available or offered by vendors NBIs. Adopt an "NBI" standard if available and agreed upon. The NBI is less straightforward since it needs to cover multiple technologies applicability
Heterogeneous Technologies and Resources within single administrative domains	• High-level orchestration of e.g. cloud / storage / network controllers based on high-level requirements and systems behavior. • Ad-hoc developments
Different administrative domains	• Peer models adopted due to business and peering agreements, trust models • Confidentiality and security issues. • Issues of Ownership and subordination. • *Forests models*. Commonly abstracts hierarchy within the administrative domain.
Wider scope infrastructures spanning multiple domains and network segments	• Hybrid approaches combining centralized, distributed elements and architectures. • Hybrid SDN-based architectures combining hierarchy and peer models, depending on inter-operability requirements, orchestration models and feasible choices. • Heavy use of abstraction and aggregation in hierarchies. • Instances of hierarchical SDN architectures for low-level interfaces (e.g. within vendor islands) and instances of hierarchical SDN architectures for high-level orchestration. • Constrained by inter-connection agreements between providers and operators. • Peer models for "data-triggered" or "event-triggered"

	provisioning across multiple segments.

5.3.2.2. Hierarchical SDN approaches based on API classes

Although not fully decupled from the previous one, it is also possible to use API classes as a criterion to identify potential SDN hierarchical architectures. For example, at a given level, a hierarchical relationship may be based on a high-level API and framework, relying on e.g. Intent based operation, control or orchestration. On the other hand, another hierarchical relationship may apply at a low-level interface, in which, macroscopically, the operation of children and parent (or sibling controllers) is fundamentally similar and the portioning is motivated by scalability, confidentiality and robustness reasons.

Uses	**Remarks**
Low-level APIs	• Often tied to a specific low-level byte protocol. • Difficult interoperability, often only reasonable if within the same SDN controller vendor. • Strongly dependent on the hierarchy support provided by the SDN controller vendor. • Implemented to scale by combining homogeneous small-size systems into bigger ones in “stages”. • Theoretical support or a loosely constrained or arbitrary number or hierarchy levels. • *~ “By design”*
High-level APIs	• Commonly associated to high-level operations using high-level frameworks and constructs (e.g. REST). • APIs “exported” by the SDN controllers and “consumed”/”used” by orchestration systems. It does not preclude the use of a common, standard protocol within an implementation agreement or standard. • Often relies on implementing dedicated plugins at the orchestration (parent) entity, drives the specification of (standard) implementation agreements for APIs. • Number of hierarchy levels limited (e.g. two in most common deployments). • Quite adapted to common uses of orchestration of heterogeneous systems or vendors. • *~ “By agreement”*

6. Analytical Description of Relevant Problems

Network applications in the SDN controllers attempt to use the network in an optimal way. The applications perform the optimizations based on models of the network. This section presents initial analytical descriptions for both network optimization and computing power consumption.

6.1. Network optimization model

This section explains an algorithm to optimize the forwarding behavior of a 5G-Crosshaul network comprised of multiple backhaul and fronthaul flows. Fronthaul traffic is constraint to pass through an XPU first, in order to process Base Station (BS) raw data, and then it is forwarded towards the core network. Backhaul (BH) traffic does not necessarily have to be relayed by an XPU before leaving to the Internet.

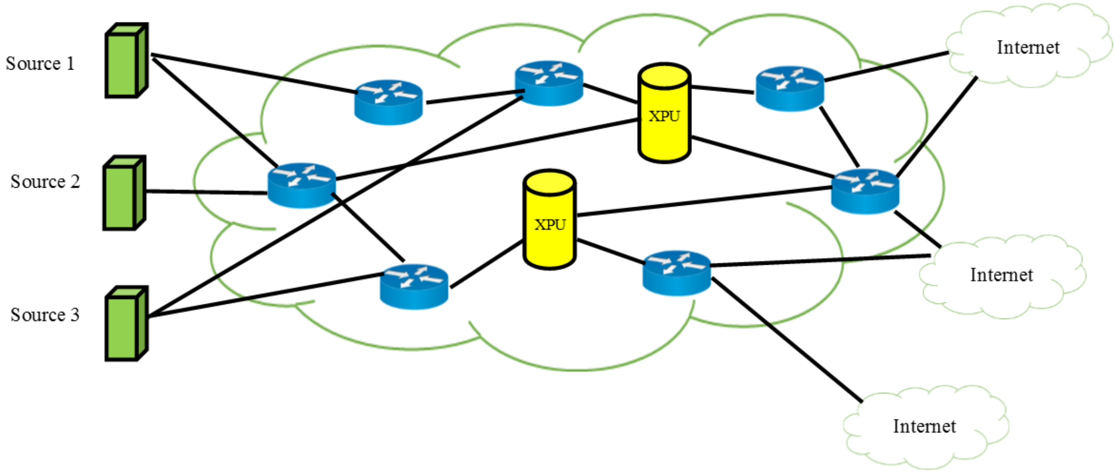


Figure 10: General Scenario to optimize

We consider a scenario like the one shown in Figure 10. This scenario contains source nodes generating backhaul traffic, source nodes generating fronthaul traffic, intermediate nodes which simply forward traffic, XPUs that contain Base-Band Unit (BBU) functionality to process fronthaul traffic, Internet gateways, and network links with specific capacities connecting different nodes. As explained earlier, fronthaul traffic must pass through at least one XPU providing BBU functionality, while backhaul traffic has no such constraint. Finally, an XPU may provide multiple BBU functions (up to a defined maximum). To avoid an overly complex model, we assume that each BBU can process at most one fronthaul flow. Our task is thus to set the paths or routes from sources (both fronthaul and backhaul flows) to one gateway to the Internet.

Our goal is to maximize the number of flows in the network by deciding on the paths followed by each flow and by assigning BBU functions to XPUs, subject to the constraints stated above.

Assumptions: We assume that links can use any layer 2 forwarding technology, providing a specific capacity on this link. Delay and jitter constraints are not considered.

Problem formulation: We formulate the problem with binary variables, which derives in an Integer Linear Programming problem (ILP).

Input parameters:

- f_k^l flow k from source l that goes to a BBU (fronthaul traffic)
- g_k^l flow k from source l that goes to the Internet (backhaul traffic)
- c_{ij} maximum capacity for the link (i,j)

All of these parameters, including f_k^l and g_k^l, are numerical values denoting a bitrate.

Variables:

- $\beta_{f_k^l}$ binary variable to determine if the f_k^l flow enters in the network or not.
- $\beta_{g_k^l}$ binary variable to determine if the g_k^l flow enters in the network or not.
- $x_{ij\, f_k^l}$ binary variable to determine if the (i, j) link is used for the f_k^l flow.
- $x_{ij\, g_k^l}$ binary variable to determine if the (i, j) link is used for the g_k^l flow.
- δ_i binary variable to determine if the i-th XPU is used or not.
- z_{rs} binary variable to determine if the s-th BBU in the r-th XPU is used or not.
- $z_{rs\, f_k^l}$ binary variable to determine if the s-th BBU in the r-th XPU is used for the f_k^l flow.

Objective function: $max \sum_{k,l} (\beta_{f_k^l} + \beta_{g_k^l})$

Constraints:

- to determine if the f_k^l flow exits from the source: $\beta_{f_k^l} = \sum_j x_{s_l\, j\, f_k^l}$ $\forall$ f_k^l flow
- to determine if the g_k^l flow exits from the source: $\beta_{g_k^l} = \sum_j x_{s_l\, j\, g_k^l}$ $\forall$ g_k^l flow
- to impose that flows in a link cannot exceed the capacity of the link:

$$\sum_{k,l} f_k^l \cdot x_{ij\, f_k^l} + \sum_{k,l} g_k^l \cdot x_{ij\, g_k^l} \ \leq \ c_{ij} \qquad \forall \text{ (i,j) link}$$

- to impose that all backhaul flows in the network arrives to the Internet:

$$\sum_{k,l} (g_k^l \cdot \beta_{g_k^l} + f_k^l \cdot \beta_{f_k^l}) = \sum_{r,i} (\sum_{k,l} g_k^l \cdot x_{i\, Int_r\, g_k^l} + \sum_{k,l} f_k^l \cdot x_{i\, Int_r\, f_k^l})$$

- to impose that all fronthaul flows in the network arrives to the BBUs:

$$\sum_{k,l} f_k^l \cdot \beta_{f_k^l} = \sum\nolimits_{r,s} \sum\nolimits_{k,l} f_k^l \cdot z_{rs\, f_k^l}$$

- if a BBU is used, the XPU where it is allocated is also used:

$$z_{r\,s\,f_k^l} \le \sum\nolimits_i x_{i\,XPU_r\,f_k^l} \qquad \forall \text{ BBU s in the XPU r, } \forall\, f_k^l \text{ flow}$$

- a fronthaul flow uses one XPU only: $\sum_i x_{i\,XPU_r\,f_k^l} \le \beta_{f_k^l}$ ∀ XPU r, ∀ f_k^l flow
- a fronthaul flow uses one BBU only: $\sum_{r,s} z_{r\,s\,f_k^l} = \beta_{f_k^l}$ ∀ f_k^l flow
- all the fronthaul flows that arrive to a XPU must be allocated in a BBU:

$$\sum\nolimits_i x_{i\,XPU_r\,f_k^l} = \sum\nolimits_{r,s} z_{r\,s\,f_k^l} \qquad \forall \text{ XPU r, } \forall\, f_k^l \text{ flow}$$

- all the traffic that enters in a node must go out from it:

$$\sum\nolimits_j x_{jr\,f_k^l} = \sum\nolimits_i x_{ri\,f_k^l} \qquad \forall \text{ node r, } \forall\, f_k^l \text{ flow}$$

$$\sum\nolimits_j x_{jr\,g_k^l} = \sum\nolimits_i x_{ri\,g_k^l} \qquad \forall \text{ node r, } \forall\, g_k^l \text{ flow}$$

- to determine if a BBU is used or not:

$$z_{ij\,f_k^l} \le z_{ij} \qquad \forall \text{ BBU j in the XPU i, } \forall\, f_k^l \text{ flow}$$

$$z_{ij} \le \sum\nolimits_{k,l} z_{ij\,f_k^l} \qquad \forall \text{ BBU j in the XPU i}$$

$$\sum\nolimits_{k,l} z_{ij\,f_k^l} \le 1 \qquad \forall \text{ BBU j in the XPU i}$$

- to determine if a XPU is used or not:

$$z_{ij} \le \delta_{ij} \qquad \forall \text{ BBU j in the XPU i}$$

$$\delta_i \le \sum\nolimits_j z_{ij} \qquad \forall \text{ XPU i}$$

- backhaul traffic cannot enter in a XPU: $\sum_i \sum_{k,l} x_{i\,XPU_r\,g_k^l} = 0$ ∀ XPU r
- to impose a maximum number of BBUs in a XPU: $\sum_j z_{ij} \le N_i$ ∀ XPU i
- single path for each fronthaul flow: $\sum_j x_{ij\,f_k^l} \le \beta_{f_k^l}$ ∀ node i, ∀ f_k^l flow

- single path for each backhaul flow: $\sum_j x_{ij\, g_k^l} \leq \beta_{g_k^l}$ $\forall$ node i, $\forall$ g_k^l flow
- binary variables:

$\beta_{f_k^l} \in \{0,1\}$ $\forall$ f_k^l flow

$\beta_{g_k^l} \in \{0,1\}$ $\forall$ g_k^l flow

$x_{ij\, f_k^l} \in \{0,1\}$ $\forall$ (i,j) link, $\forall$ f_k^l flow

$x_{ij\, g_k^l} \in \{0,1\}$ $\forall$ (i,j) link, $\forall$ g_k^l flow

$z_{rs} \in \{0,1\}$ $\forall$ XPU r, $\forall$ BBU s

$z_{rs\, f_k^l} \in \{0,1\}$ $\forall$ XPU r, $\forall$ BBU s, $\forall$ f_k^l flow

$\delta_i \in \{0,1\}$ $\forall$ XPU i

Applying integer linear programming to maximize the objective function under the given constraints maximizes the number of flows that enter the network and are routed until its destination, it solves the placement of BBUs to XPUs, and determines the route each flow follows to reach its destination. Our goal is to minimize the number of used resources while the maximum number of flows is maintained and to prioritize fronthaul traffic over the backhaul traffic, which is not yet achieved by the initial objective function given above. Work is ongoing to achieve our objective and introduce more parameters in the network to control and optimize more aspects of it. Eventually, we expect to control most of the parameters of a 5G network to allow the controller determine the best route each flow has to follow based on the information available.

6.2. Power consumption computation model

Energy consumption of a backhaul network can be minimized by limiting the number of active links and nodes, i.e., by (i) turning off link drivers whenever possible, resulting in proportional (possibly non-linear) changes, and (ii) turning off those nodes whose links are inactive.

Both approaches can be studied by building a directed network graph whose vertices represent the network nodes and edges correspond to links connecting the nodes. Let us then consider that the network includes *N* nodes and *L* links and denote by *N* and *L* the set of nodes and links, respectively. Let a link *(i,j)* $\in$ *L*, with *i,j* integers, have a capacity *C(i,j)* bits/s. Let *F(t)* denote the set of flows at time *t*, with each flow, $f^{sd} \in F(t)$, characterized by a source-destination pair, a traffic volume, and QoS constraints that in our case correspond to the required data rate $R(f^{sd})$.

Let $x_{ij}(t)$ be a binary variable indicating whether link (i,j) $\in L$ is "on" ($x_{ij}(t) = 1$) or "off" ($x_{ij}(t) = 0$), at time t. Likewise, $y_i(t)$ is a binary variable indicating whether node $i \in N$ is active at time t ($y_i(t)=1$) or not ($y_i(t)=0$). Also, let a path π be an ordered sequence of links. We indicate by the binary variable $z_{\pi,f^{sd}}$ (t) whether flow $f^{sd} \in F$ (t) is routed through path π at time t ($z_{\pi,f^{sd}}$ (t) = 1) or not ($z_{\pi,f^{sd}}$ (t) = 0).

We consider that the generic node *i* has zero power consumption when "off", and P_{idle} when "on" but idle. The power consumption associated with a link *(i,j)* at time *t* linearly depends on the traffic that flows over the link and is denoted by $P_{(i,\,j,\,t)}$. It follows that the total power consumption of a node *i* that is "on" is given by:

$$P(i,t) = P_{idle} + \sum_{j\in\mathcal{N},j\neq i} [P(i,j,t)x_{ij}(t) + P(j,i,t)x_{ji}(t)]$$

The traffic flowing over link *(i, j)* at time *t*, $\tau_{ij}(t)$, is expressed in bit/s and is given by the sum of the traffic associated with all flows that are routed through the link, i.e.,

$$\tau_{ij}(t) = \sum_{f^{sd}\in\mathcal{F}(t)} \sum_{\boldsymbol{\pi}:(i,j)\in\boldsymbol{\pi}} R(f^{sd}) z_{\boldsymbol{\pi},f^{sd}}(t)$$

The power consumption of an OpenFlow switch that is "on" can be written as the sum of the power consumed by its three major subsystems: $P_{ctr} + P_{evn} + P_{data}$, where P_{ctr} accounts for the power needed to manage the switch and the routing functions, P_{evn} is the power consumption of the environmental units (such as fans), and P_{data} indicates the data plane power consumption. The latter can be decomposed into (i) a constant baseline component, and (ii) a traffic load dependent component. In other words, when a switch is powered on but it does not carry any data traffic, it consumes a constant baseline power. When a device is carrying traffic, it consumes additional load-dependent power for header processing, as well as for storing and forwarding the payload across the switch fabric. Combining the power model in [11] with that for OpenFlow switches in [12], we can write P_{idle} as the sum of P_{ctr} , P_{evn} and the baseline component of P_{data} , while the load-dependent component of P_{data} is given by:

$$P(i,j,t) = (E_{lookup} + E_{rx} + E_{xfer} + E_{tx})\tau_{ij}(t)$$

In the above expression,

- E_{lookup} is the energy consumed per bit in the *lookup* stage of a switch, which involves searching the Ternary Content Addressable Memory (TCAM) for the received flow-key and retrieving the forwarding instructions;

- E_{rx} is the energy consumed per bit in the *reception* stage, which involves receiving a packet from the physical media, extracting important fields to build a flow-key and streaming the packet into the input memory system;
- E_{xfer} is the energy consumed per bit in the *xfer* stage, which involves reading a packet from the inbound memory, all of the logic required to initiate a transfer across the fabric, driving the fabric connections and crossbar, as well as writing the packet into the remote outbound memory;
- E_{tx} is the energy consumed per bit in the *transmission* stage, which involves reading a packet from the outbound memory and transmitting it on the physical media.

Example values for energy consumption are E_{rx} = E_{tx} = 0.2 nJ/bit, E_{xfer} = 0.21 nJ/bit, E_{lookup} = 0.034 nJ/bit, and P_{idle} = 90 W [13].

Let us now consider that a generic node *i* is used by $\nu_i(t)$ slices at time *t*. Also, let us denote by $\eta_\sigma(i,j,t)$ the fraction of traffic that flows over link *(i,j)* at time *t* belonging to slice σ, and by $w_\sigma(i,t)$ the binary indicator function that is 1 if node *i* is part of slice σ at time *t*, and 0 otherwise. Note that $\eta_\sigma(i,j,t)=0$ if link *(i,j)* is not part of slice σ. We can therefore write the total power consumption associated with slice σ as:

$$P(\sigma,t) = \sum_{i\in\mathcal{N}} w_\sigma(i,t)\frac{P_{idle}}{\nu_i(t)}y_i + \sum_{(i,j)\in\mathcal{L}} \eta_\sigma(i,j,t)P_\sigma(i,j,t)x_{ij}(t)$$

To derive the above equation, we considered that the baseline power cost of an active switch is evenly divided among the slices insisting on the node, while the variable component depends on the actual traffic carried by the slice.

The power consumption related to IT and VM migration is currently under study.

7. Southbound Interface

The goal of this section is to provide a brief summary of the main outcomes of the initial specification of the XCI controller interfaces at the Southbound, namely SBI. Such interface is the common language spoken between the controller and the switches and enables remote entities to control the forwarding behaviour of the switches. Although the SBI is usually considered part of the control plane, it can be also considered part of the data plane since switches require a SBI agent in charge of communicating with the controller and enforcing its decisions that are encoded according to the SBI protocol. In this project we followed a bottom-up approach where SBI is data-plane focused and its design is constrained by XFE and its ability of forwarding XCF over different transmission technologies. On the contrary, a top-down approach would mainly focus on the controller requirements and partially on the switching capabilities. Consequently, XCF and SBI have been already described in D2.1 [1], however a summary is given here to keep this deliverable self-contained.

There are two differentiated types of southbound protocols depending on their purpose: control and management. The control protocols primarily control the forwarding/routing and the management protocols convey information regarding the configuration and administration of the network elements. A number of SBI protocols have been studied for controlling the forwarding/routing of packets through the 5G-Crosshaul network (e.g. OpenFlow, ForCES – Forwarding and Control Element Separation), for management of the switch/device (e.g. OF-Config, OVSDB, SNMP, NETCONF), and for interacting with legacy systems (e.g. BGP, CoAP, etc). The detailed introduction of each protocol can be found in [1].

The OpenFlow (OF) protocol [21] follows a bottom-up approach and is considered as the main candidate for the SBI interface for controlling the 5G-Crosshaul network elements (e.g. XFEs) on the forwarding/routing. The selection is motivated by the high support of the OF protocol within the current SDN deployments, and moreover the latest version of this protocol (OpenFlow 1.5.1 specification [21]) can fulfil the XPFE, AF, and XCF design requirements, as explained in the analysis presented in [1]. ForCES is an example of SBI designed following a top-down approach that, despite of being well standardized, is proven to be difficult to implement on hardware switches.

However, the OF protocol will require extensions to handle the heterogeneous transmission technologies considered at the data plane of the 5G-Crosshaul network, including millimeter, microwave, optical wireless, optical fixed access, copper, etc. [1] presents a methodology for such purpose and a detailed analysis of the required extensions for each of the identified transmission technologies. It is worth noting that this analysis provides a choice of the relevant SBI parameters for each considered technology. The selection of new parameters and their granularity was based on the modus operandi adopted in the definition of Optical Transport Protocol Extensions by ONF [14] that has been extended to 5G-Crosshaul transmission technologies.

8. Northbound Interface

This section provides a preliminary specification of the XCI controller interfaces at the Northbound, namely the Northbound Interface (NBI). In particular, the specification of the XCI NBI API is based on the REST paradigm, with HTTP as reference protocol for the exchange of messages between NBI services consumers and providers. REST is indeed one of the most used paradigms for specifying Northbound services for both SDN and IT controllers (e.g., ODL [8], ONOS [10], and OpenStack [7]). Note that this does not preclude the use of other protocols such as proprietary ones or the use of WebSockets [15]. In particular, the use of WebSockets can be of primal importance for certain modules to provide, for instance, asynchronous notifications of network or compute events from XCI services towards 5G-Crosshaul applications.

The NBI specification is still in progress in the main standardization fora. Organizations such as the ONF Northbound Interface Working Group and the ETSI NFV Industry Specification Group (ISG) are working towards this goal, defining APIs for the most relevant services provided by SDN controllers and ETSI MANO components. Regarding the specifics on the REST APIs, we leave open the choice of whether REST APIs to implement will have full compliance with RESTCONF protocol [16]. In this way, there is no need to define a YANG data model associated to the REST APIs. This will be a design decision that will be later defined in stages closer to the implementation phase.

The selection of XCI services is the result of the work initiated in the initial phase of WP3 activities. Specifically, the set of XCI NBI services is based on the Northbound functionalities identified in this initial phase and the requirements of the 5G-Crosshaul applications, which are included in D4.1 [2]. Note that the details reported in this section will be used as initial input for the subsequent development of the 5G-Crosshaul NBI. The XCI NBI services will be implemented internally in the SDN controller, the IT controller, which is mapped to the compute and storage controller in the Crosshaul architecture and as part of the MANO components, namely: the NFV Orchestrator (NFV-O), the VNF Manager (VNFM), and the Crosshaul VIM referred to as the Virtual Infrastructure Manager and Planner (VIMaP).

The following subsections provide the specification of the APIs for the NBI services exposed by the XCI towards the 5G-Crosshaul applications or towards other XCI modules. In particular, for each NBI service we provide the APIs, the more relevant information elements used to model the data associated to this NBI service, and a workflow illustrating the use of this service by generic 5G-Crosshaul applications or by internal modules inside the XCI that, in turn, can expose their own NBI. As for the APIs, we include a table indicating the protocol, the URIs, the operation, and the input/output parameters associated to the operation. As for the data model, we provide a UML diagram specifying the most relevant elements involved in the NBI service. Finally, we include a workflow example, in the shape of a sequence diagram, which illustrates some examples of each NBI service. Note that the low-level implementation details of the APIs, workflows, and information data models are beyond the scope of this deliverable.

8.1. Topology and Inventory

The Topology service maintains information regarding the physical network and all created networks on top of the physical network. It abstracts the learnt physical topology information that may be collected by some automated process also involving the network elements (e.g., using protocols like LLDP). This abstraction offers the possibility of creating subnetworks (or subnets) that are formed by a subset of the whole physical topology, and/or to enhance the topology with other kind of information (e.g., TE topology including TE metrics, SRLGs, etc.) providing the REST APIs to create, remove networks (i.e., a subnet which is different concept from that of tenant), add/remove node links to a network.

On the other hand, the Inventory component provides query network inventory services. In particular, it provides REST APIs to query inventory data from the inventory database. In this way, the network node and port capabilities can be obtained from this service.

8.1.1. APIs

The Northbound APIs defined in the following table give access to the network topology stored and maintained by the SDN controller (i.e., the one formed by XFEs). These APIs also provide primitives to create, delete, and modify the subnetworks in the physical network infrastructure. The APIs detailed below also expose network inventory resources, such as the list of nodes and their capabilities. In what follows, we provide a description of the APIs offered by the Topology and Inventory services.

Table 3: Topology and Inventory API.

Prot.	**Type**	**URI**	**Parameters**	
REST	GET	*../topology/default* Retrieve the whole physical network infrastructure. *../ topology/{network_id}* Retrieve the specified network topology with identifier network_id.	Input	*network_id* (optional)
			Output	*network_object*
REST	POST	*../topology/{network_id}*	Input	*network_id*

		Register a new network.	Output	Success: Status Code of normal end
				Failure: Error code
REST	PUT	*../topology/{network_id}* Add subnetwork with id *network_id* to the physical network as specified by *network_object.*	Input	*network_id* *network_object*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	PUT	*../topology/{network_id}/{link_id}* Add new link *link_id* to *network_id.*	Input	*network_id* *link_id*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	DELETE	*../topology/{network_id}* Delete an existing network with identifier *network_id.*	Input	*network_id*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	GET	*../topology/default/nodes/* Retrieve all the nodes. *../topology/{network_id}/nodes* Retrieve the specific nodes in subnet with *network_id.*	Input	*network_id* (optional)
			Output	*node_list*
REST	GET	*../topology/nodes/node_id* Retrieve node information details of *node_id.*	Input	*node_id*
			Output	*node_object*
REST	GET	*../topology/{src_node_id}_{dst_node_id}* Get the shortest path in terms of number of hops between two infrastructure devices.	Input	*src_node_id* *dst_node_id*
			Output	*path_object*
WebSockets	SUBSCRIBE	../topology/*{network_id}_{event}*	Input	*network_id* *event*
			Output	Success: Status Code of normal end
				Failure: Error code
WebSockets	ASYNC	notification event	Input	*event*
			Output	N/A

8.1.2. Information Model

We consider the notion of a physical topology that is comprised of a list of the networks. These networks are subnets from the physical network. Each of the multiple topologies has the notion of multiple nodes. Each node has multiple ports. Each network has multiple links, where each one connects two ports of different nodes. Based on the technology, the object "Port" can have different technologic-specific attributes.

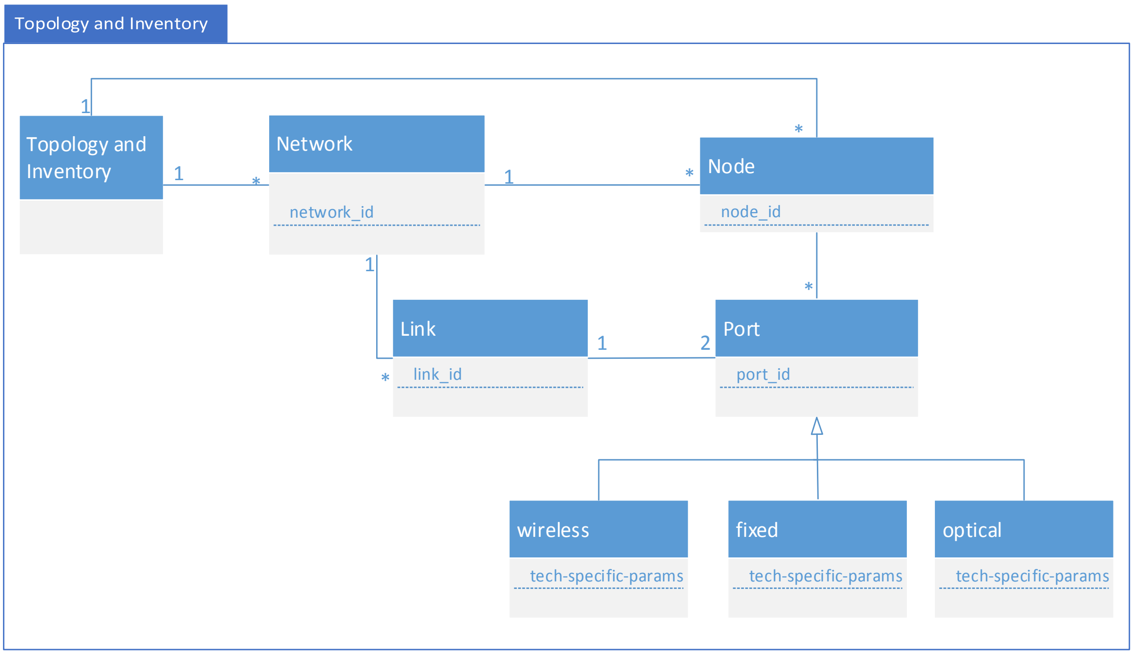


Figure 11: Topology and Inventory Information model.

A more detailed description of the attributes of some relevant information models associated with the Topology and Inventory services is provided next.

Table 4: Topology and Inventory Information model.

Parameters	Type	Description
network_id	String	Identifier of the network.
network_object	Set<Links>	Object describing the network as a set of links in JSON or XML format. Each link has the following attributes: • EndpointA; String; Node • EndpointB; String; Node • Bandwidth; Integer; Bandwidth of the link. • State: Enum, whether the link is active or not • Technology; Enum; Indicate type of the link (e.g., mmWave, optical)
network_event	Enum	The specific set of network event will be decided in the implementation stage. {node_up, node_down, link_up, link_down, port_up, port_down}.
node_id	String	Identifier of the node.
node_object	Node	• Contains the identifier and the type of the node, as a set of properties in JSON or XML format.

		Parameters (variable; type; description):Identifier: String, identifier of the node • Type: String, type of node • Name: String, name of the node • Number of Ports: Integer, number of ports
path_object	Object	Object describing the path as set of links between two endpoints (in XML or JSON format).

Request/response media types can be in the form JSON and/or XML. It is important to note that the URIs specified in the request by the consumer shall be independent of the chosen representation in the implementation. For instance, in what follows we illustrate a part of the response body in JSON format of a network topology corresponding to a link connecting a node with a switch. In particular, the request will have the following form: GET *../topology/n1*, where *n1* is the identifier of the network. A part of the output of the physical network topology corresponding to subnetwork *n1* follows:

```
"network":[
{
   "id":"n1",
   "name":"net_example",
   "link":{
              "id": "1",
              "EndPointA":{
                                   "id":"00:00:00:00:00:00:00:01",
              },
              "EndPointB":{
                                  "id":"00:00:00:00:00:00:00:51",
              },
              "name": {
                           "value": "s1-eth1",
              },
              "state": {
                          "value": 1,
              },
              "bandwidth": {
                                  "value": 100; // Data rate in Mbps
               }
         }
         "link":{….} //More links can be specified
}
]
```

8.1.3. Workflow

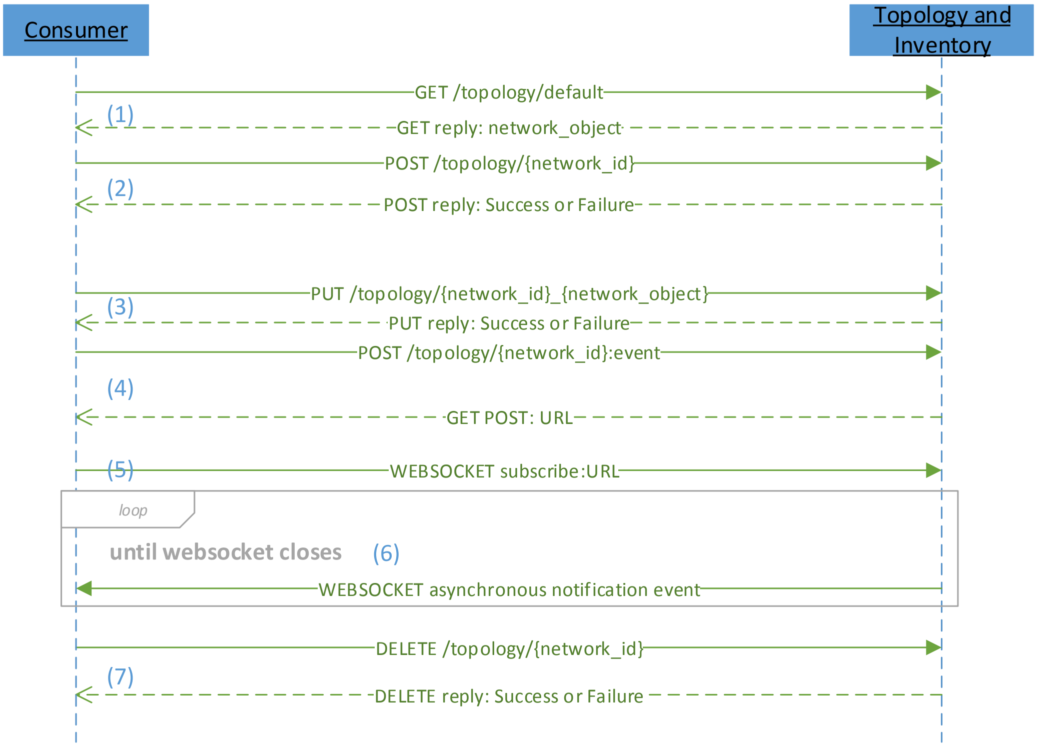

Consumer
Topology and Inventory
GET /topology/default
(1)
GET reply: network_object
POST /topology/{network_id}
(2)
POST reply: Success or Failure
PUT /topology/{network_id}_{network_object}
(3)
PUT reply: Success or Failure
POST /topology/{network_id}:event
(4)
GET POST: URL
(5)
WEBSOCKET subscribe:URL
loop
until websocket closes
(6)
WEBSOCKET asynchronous notification event
DELETE /topology/{network_id}
(7)
DELETE reply: Success or Failure

Figure 12 : Topology and Inventory workflow example.

In what follows, we provide a message exchange sequence to illustrate the use of the service by an application referred to as the consumer. This consumer can be an application located on top of the XCI. The goal is to illustrate the use of the Topology and Inventory services. In particular, the workflow in Figure 12 illustrates the creation of a physical subnetwork forming part of the physical network topology. As shown in Figure 12, this process can consist of the following major steps between an application (represented as *Consumer* in the picture) and the Topology and Inventory service:

1. The consumer application gets the physical network inventory from the Topology and Inventory service.
2. The consumer application decides to register a new subnetwork. A physical subnetwork is specified and registered from the consumer to the Topology and Inventory service.
3. The elements of the physical subnetwork are specified for the previously registered network identifier. A physical subnetwork from the consumer to the Topology and Inventory service.
   3.1. Selection of a private or public subnetwork based on the range of defined IPs.
   3.2. Selection of VLAN associated to the registered physical subnetwork.
4. Subscription to an event of interest related to this physical subnetwork (e.g., link failure). An URL is returned as a result of a successful subscription.
5. Creation of a WebSocket to receive asynchronous notifications from the Topology and Inventory service for the event to which the consumer application is subscribed.

6. Asynchronous notifications of the previously subscribed event by the Topology and Inventory service to the consumer.
7. The consumer application decides to deallocate the physical subnetwork.

8.2. Provisioning and Flow Actions

This service provides the flow programming service for 5G-Crosshaul applications. In particular, this service is in charge of offering, on the northbound (applications or certain modules inside the XCI), an API that allows to perform queries related with the management of flow rules in network nodes.

8.2.1. APIs

The table below provides the set of Northbound APIs that are available to perform different operations on the flows whereby an SDN controller (e.g. ONOS, ODL). These interfaces, in the form of REST API, allow creating, deleting, and modifying flow rules in physical nodes. In this specific context, nodes are intended as XPFE. The response to these operations is in XML or JSON format.

Table 5: Provisioning and flow actions API: flow rules in physical devices.

Prot.	**Type**	**URI**	**Parameters**	

REST	POST	*../sdn_ctrl/flows/{node_id}*
		or
		../sdn_ctrl/flows/{node_id}/{table_id}
		Create a new flow rule.
			Input	*node_id*
				table_id (optional)
				flow_object

			Output	Success: *flow_id*
				Failure: Error code

REST	DELETE	*../sdn_ctrl/flows/{node_id}/{flow_id}*
		or
		../sdn_ctrl/flows/{node_id}/{table_id}/{flow_id}
			Input	*node_id*
				table_id (optional)
				flow_id

			Output	Success: Status Code of normal end

		Delete an existing flow rule.		Failure: Error code

REST	GET	*../sdn_ctrl/flows/{node_id}*
		or
		../sdn_ctrl/flows/{node_id}/{table_id}
		Retrieve the list of all flow rules on a specific node or table.
			Input	*table_id*
				node_id

			Output	*flow_object* List of flow rules on the specified node, or out of a specific flow table
REST	GET	*../sdn_ctrl/flows/* Retrieve all the flow rules	Input	-
			Output	List of all flow rules
REST	GET	*../sdn_ctrl/statistics/flows/{node_id}/{port_id}/* Retrieve statistics for all flows passing over a port of a node	Input	*node_id* *port_id*
			Output	Aggregate values • Bytes counter • Rate [bytes/s] Time of last statistics collection
REST	GET	*../sdn_ctrl/streams/flows/{event_id}*	Input	*event_id*
			Output	URL to the notification service (e.g. Websocket)
WEBSOCKE	SUBSCRIBE	*../sdn_ctrl/streams/flows/{event_id}*	Input	*event_id*

T			Output	Success: Status Code of normal end
Websocket	ASYNC	notification event	Input	*event_id*
			Output	*event_object*

8.2.2. Information Model

Description of the parameters used for setting up flow rules in the switching elements are shown herein below in Table 6. The UML diagram describing the process of assigning flow rules in a node (i.e. switching element) is shown in Figure 13.

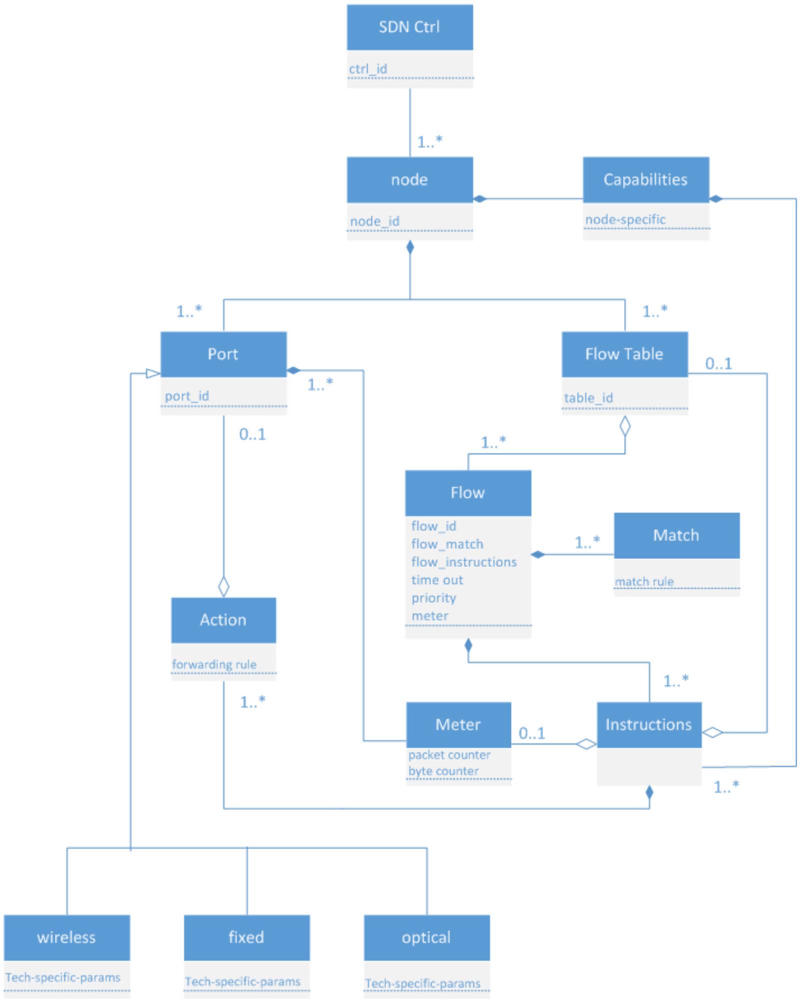


Figure 13: Provisioning and flow actions information model.

Next, the main data objects are presented in more detail:

Table 6: Provisioning and flow actions information model.

Parameters	**Type**	**Description**
node_id	String	Identifier of the node
table_id	String	Identifier of the flow table
port_id	String	Identifier of a port
flow_id	String	Identifier of a flow
flow_object	Object	• table_id: string • flow_id: string • flow_match: string denoting the matching rule • flow_instruction: string denoting the instructions • Time out: Integer denoting the time the flow rule exists • Priority: Integer indicating the priority of a flow rule • Meter: Integer denoting statistics such as packet and byte counters
event_id	String	Identifier of a specific event to subscribe
event_object	Object	Object which contains the set of information specific to the subscribed event

8.2.3. Workflow

The workflow provided in Figure 14 illustrates the installation of flow rules (e.g. FlowMod in OpenFlow) in a node. Any 5G-Crosshaul application (e.g. MMA) or even the VIM, can request the SDN controller to create, delete and modify flow rules in one or more switching elements at the data plane.

The workflow shown in Figure 14 includes the following steps:

1. A consumer application requests the creation of new flow rules in a specific device
   a. The SDN controller will create new flow rules specifying the matching rules, instructions, priority and time out, just to name a few.
2. A consumer application decides to request the details of a specific flow rule
   a. The SDN controller returns an object describing the parameters of the flow (i.e. flow identification, flow match, flow actions, flow priority and time out)
3. A consumer application decides to request the statistics for all the flows passing through a specific port
   a. Upon receiving the request, the SDN controller shall return related statistics such as bytes counter, rate and the time at which statistics were collected.

4. A consumer application decides to subscribe to the notification of a specific event of interest (e.g. arrival of a new flow). An URL is returned in response.
5. A WebSocket is created in order to subscribe to the notification service for the event of interest.
6. Asynchronous notifications are received for the specific event of interest. An object is returned in response containing the set of information for the subscribed event.
7. A consumer application decides to remove (delete) a specific flow rule from a device.

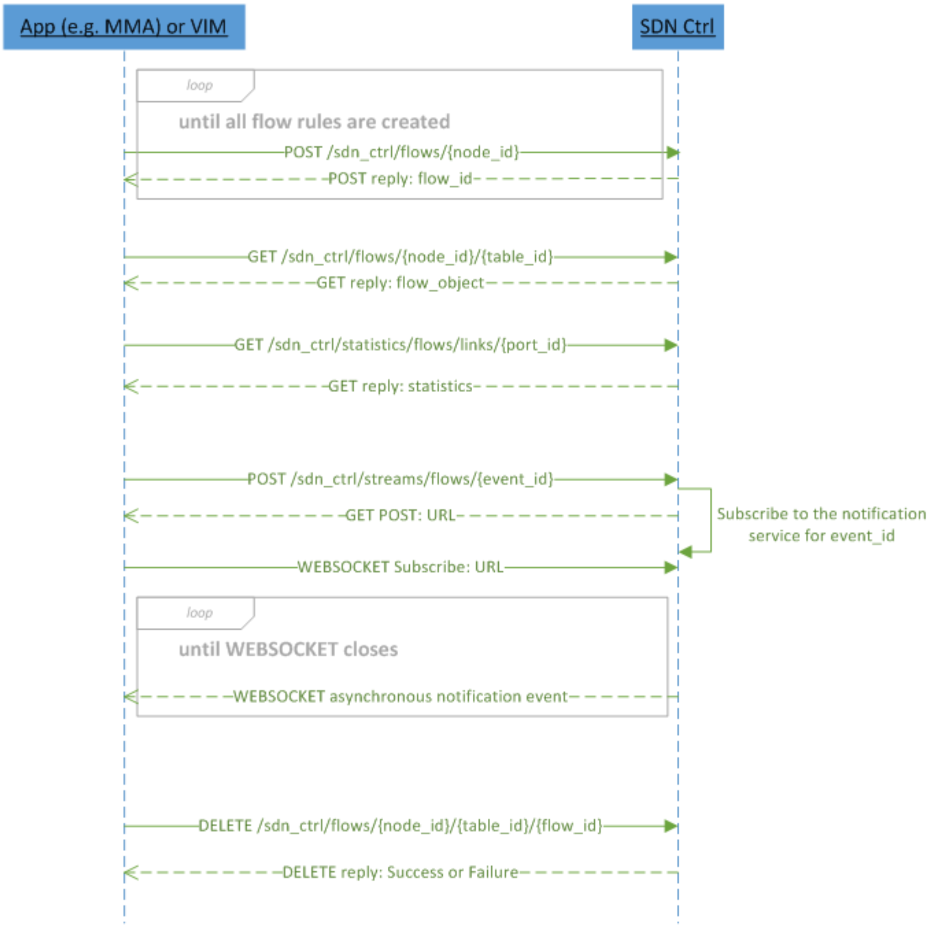

App (e.g. MMA) or VIM
SDN Ctrl
loop
until all flow rules are created
POST /sdn_ctrl/flows/{node_id}
POST reply: flow_id
GET /sdn_ctrl/flows/{node_id}/{table_id}
GET reply: flow_object
GET /sdn_ctrl/statistics/flows/links/{port_id}
GET reply: statistics
POST /sdn_ctrl/streams/flows/{event_id}
GET POST: URL
Subscribe to the notification service for event_id
WEBSOCKET Subscribe: URL
loop
until WEBSOCKET closes
WEBSOCKET asynchronous notification event
DELETE /sdn_ctrl/flows/{node_id}/{table_id}/{flow_id}
DELETE reply: Success or Failure

Figure 14: Flow actions example.

8.3. IT Infrastructure and Inventory

The IT infrastructure and inventory NBI is based on the REST protocol. In particular, we use the NBI of several OpenStack [7] modules (e.g., Nova,), as the NBI offered by the IT infrastructure and inventory service. These modules will form part of the IT controller inside the XCI as a set of plugins.

To offer such an API, the IT infrastructure and inventory service must comprise several modules. For instance, the set of instantiated VMs must be maintained by an IT infrastructure and inventory database. Also, an IT infrastructure and inventory scheduler will need the list of available servers to determine where the VM must be instantiated.

8.3.1. APIs

In what follows, we provide a description of the APIs offered by the IT infrastructure and inventory services.

Table 7: IT infrastructure and inventory API.

Prot.	Type	URI	Parameters	
REST	GET	*../it/vms* *../it/tenant/{tenant_id}/vms* Retrieve all/per_tenant VM elements. *../it/vms?type=”type”* *../it/{tenant_id}/vms?type=”type”* Retrieve all VM elements of specified type.	Input	*type* (optional) *tenant_id*
			Output	*vm_list*
REST	GET	*../it/vm/{vm_id}* Retrieve information of VM with identifier *vm_id*.	Input	*vm_id*
			Output	*vm_object*
REST	POST	*../it/vm* *../it/tenant/{tenant_id}/vm* Create a new VM for *tenant_id*.	Input	*vm_object* *tenant_id*
			Output	*vm_id*

REST	PUT	*../it/vm/{vm_id}*
		../it/tenant/{tenant_id}/vm/{vm_id}
		Update information of VM with identifier *vm_id*.
			Input	*vm_id*
				tenant_id
				vm_object

			Output	Success: Status Code of normal end
				Failure: Error code

REST	DELE TE	*../it/vm/{vm_id}*
		../it/tenant/{tenant_id}/vm/{vm_id}
		Delete the VM (in *tenant_id*) with identifier *vm_id*.
			Input	*vm_id*
				tenant_id

			Output	Success: Status Code of normal end
				Failure: Error code

REST	GET	*../it/tenant/{tenant_id}/os_host* List compute/storage nodes in *tenant_id*	Input	*tenant_id*
			Output	List of compute/storage nodes
REST	GET	*../it/tenant/{tenant_id}/os_hosts/{host_name}* List details of *host_name*	Input	*tenant_id* *host_name*
			Output	*host_object*
REST	GET	*../it/tenant/{tenant_id}/os_hosts/{host_name}/start* Start a compute node *host_name* in tenant *tenant_id*	Input	*tenant_id* *host_name*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	GET	../it/tenant/{tenant_id}/os_hosts/{host_name}/shutdown Shutdown a compute node *host_name* in tenant *tenant_id*	Input	*tenant_id* *host_name*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	GET	../it/tenant/{tenant_id}/os_hypervisors Get list of hypervisor in tenant *tenant_id*	Input	*tenant_id*
			Output	*Hypervisor_list*
REST	GET	../it/tenant/{tenant_id}/os_hypervisors/{hypervisor_id} Get hypervisor object from hypervisor id	Input	*tenant_id* *hypervisor_id*
			Output	*hypervisor_object*

8.3.2. Information Model

The data model comprises the computing, storage, and network functions associated to a given tenant (or owner/user). Each tenant has associated a set of VMs and can comprehend one or more virtual networks. Each virtual network is formed by a set of endpoints defining the connection patterns between each pair of VMs.

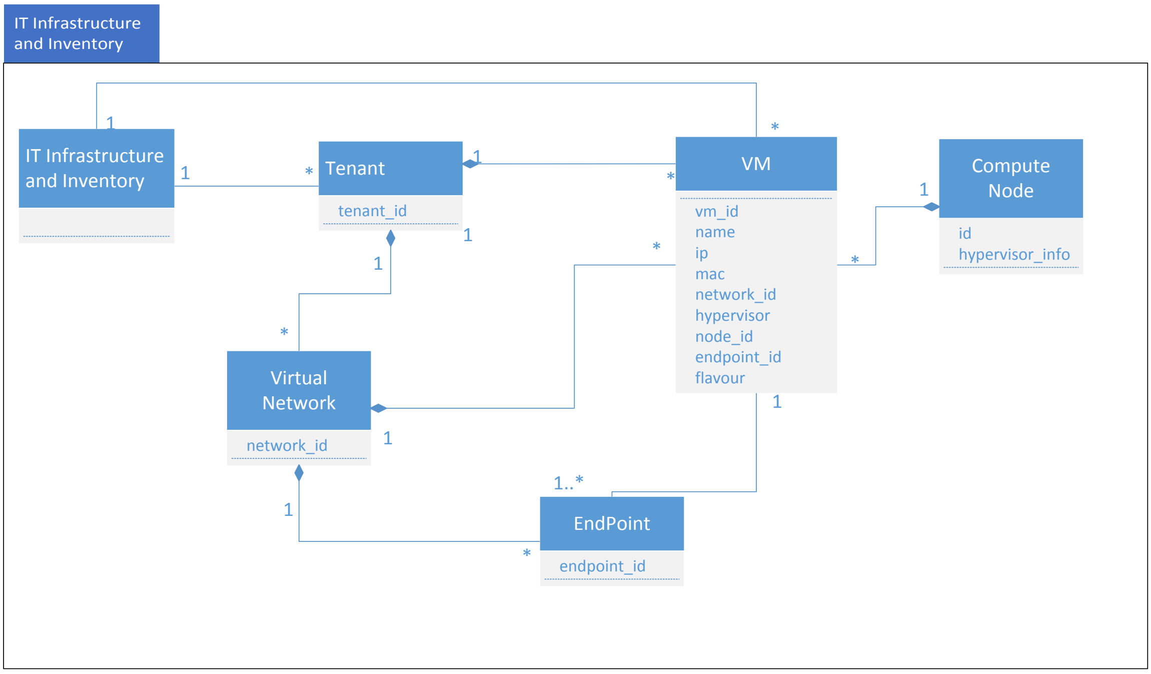


Figure 15: IT Infrastructure and Inventory information model.

A more detailed set of the attributes for the object VM is detailed in Table 8:

Table 8: IT infrastructure and inventory information model.

Parameters	Type	Description
vm_id	String	Identifier of the VM.
vm_object	Object	Object describing the VM as a set of properties in JSON or XML format. Parameters (variable; type; description): • Name; String; name of the VM. • Flavor**;** String; Hardware template, • Image_name; String; VM template, • NetworkId; String; L2 Network identifier. • SubnetId; String; L3 Network identifier. • Id; String; unique identifier of the VM. • MAC address; String; L2 address • IP address; Ipv4; L3 address • Hypervisor; String; Identifier of the compute node on which is deployed the VM. • Node_id; String; DpId of the switch attached

		- Endpoint_id; String; Network identifier of where the VM is attached.
type	String	Type of the VM (compute or storage).
Endpoint_id	String	Unique identifier of the endpoint.
Endpoint_object	Object	The endpoint object represents the termination point of a network connectivity service. It is described as a set of properties in JSON or XML format: Parameters (variable; type; description): - endpoint_id; String; Unique identifier of the termination point of a network connectivity service (Call). - node_Id; String; Identifier of the network node to which the endpoint is connected. - net_interface_id; String; Identifier of the network port component to which the endpoint is connected.

8.3.3. Workflow

In what follows, we provide a message exchange sequence to illustrate the use of the IT Infrastructure and Inventory service by the VIMaP upon the creation and management of Over the Top (OTT) network services. In this example, the consumers are the NFV-O and the VIMaP, both located inside the XCI. The goal is to illustrate the use of the IT infrastructure and inventory services for the ETSI NFV use case. In particular, the workflow in Figure 16 illustrates the creation of a network service layout composed by a set of VMs and their direct interconnection. As shown in Figure 16, this process can consist of the major steps:

1. The NFV-O requests the instantiation of a network service layout to the VIMaP composed by a set of VMs. Note that the network service layout is associated to a tenant, which is the owner of this network service layout.
2. The VIMaP parses the template specified by the NFV-O and requests to the IT infrastructure the instantiation of the following resources:
   2.1. The creation of a virtual network layout, returning an identifier in case of successful creation.
   2.2. In case of successful creation, the IT controller returns an identifier for this virtual network layout.
   2.3. The instantiation of the VMs forming the virtual network layout with the identifier previously returned by the IT infrastructure and inventory.
      2.3.1. Based on the template introduced to the VIMaP, the IT controller creates instantiates a set of VMs.
      2.3.2. The number of instantiated VMs will be based on the profile requested by the NFV-O.
      2.3.3. A zone (physical location) will be selected where each VM will be geographically deployed.

3. The VIMaP (e.g., by request of the VNF Manager or the NFV-O, or internally by request of the VIMaP) may request at any point in time of the lifecycle of the network service layout to update the profile of the network service layout previously created.
4. The VIMaP receives a request to terminate the network service layout. This implies the interaction with the IT infrastructure and inventory service for the subsequent deallocation of VMs.

The entities involved in this process are the NFV-O, the VNF manager, and the IT infrastructure and inventory.

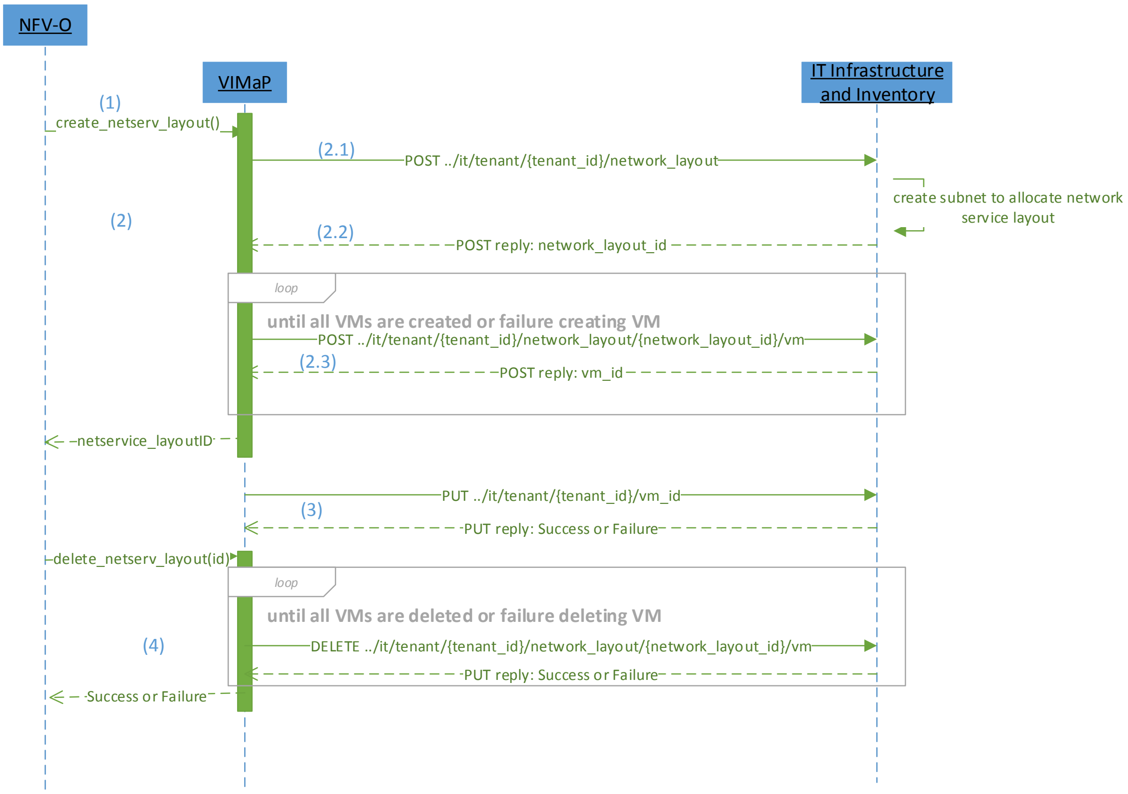

NFV-O
VIMaP
IT Infrastructure and Inventory
(1)
create_netserv_layout()
(2.1)
POST ../it/tenant/{tenant_id}/network_layout
create subnet to allocate network service layout
(2)
(2.2)
POST reply: network_layout_id
loop
until all VMs are created or failure creating VM
POST ../it/tenant/{tenant_id}/network_layout/{network_layout_id}/vm
(2.3)
POST reply: vm_id
netservice_layoutID
PUT ../it/tenant/{tenant_id}/vm_id
(3)
PUT reply: Success or Failure
delete_netserv_layout(id)
loop
until all VMs are deleted or failure deleting VM
(4)
DELETE ../it/tenant/{tenant_id}/network_layout/{network_layout_id}/vm
PUT reply: Success or Failure
Success or Failure

Figure 16: IT infrastructure and inventory workflow example.

8.4. Statistics

Initally, two services have been considered for the collection of monitoring information (one for network-related statistics and the other for IT-related statistics). Our discussions have now led us to design a unified common service, namely Statistics service, which integrates the functionality of the two. Note that this API is based on OpenStack's Ceilometer [17]. This service shall offer to any consumer (or client) a network- computing- and storage- related statistics service per tenant basis, including metering, alarm, and collection of samples.

8.4.1. APIs

In the following, we provide a description of the APIs offered by the Statistics service.

Table 9: Statistics API

Prot.	Type	URI	Parameters	
REST	GET	*../stats/{tenant_id}* Retrieve a list of *stats* that can be polled	Input	*tenant_id*
			Output	Success: Return *list of stats* in the response body.
				Failure: Error code
REST	GET	*../stats/{tenant_id}/{stat_id}/samples* Retrieve samples	Input	*tenant_id* *stat_id*
			Output	Success: Return *stat* structure in the response body.
				Failure: Error code
REST	POST	*../stats/{tenant_id}/{stat_id}/samples* Post a list of samples	Input	*tenant_id* *stat _id* *samples*
			Output	Success: Return *list of meters* in the response body.
				Failure: Error code
REST	GET	*../stats/{tenant_id}/{stat_id}/statistics* Make statistics out of samples	Input	*tenant_id* *stat _id* *stat_info*
			Output	Success: Return statistics of samples according to *stat_info* in the response body.
				Failure: Error code
REST	GET	*../stats/{tenant_id}/{stat_id}/statistics/functions* Retrieve list of	Input	*tenant_id* *stat_id*

		available aggregate	Output	*List of aggregate functions XCI can perform to obtain aggregated statistics (e.g., mean, stdev, percentiles..).*
REST	GET	*../stats/{tenant_id}/{stat_id}/alarms* Retrieve list of available alarms	Output	Success: Return list of *alarms* structure in response body.
			Output	Failure: Error code
				Failure: Error code
REST	PUT	*../stats/{tenant_id}/{stat_id}/alarms/{alarm_id}* Set alarm_id	Input	*tenant_id* *stat_id* *alarm_id* *alarm_info*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	DELETE	*../stats/{tenant_id}/{stat_id}/alarms/{alarm_id}* Delete alarm_id	Input	*tenant_id* *stat_id* *alarm_id*
			Output	Success: Status Code of normal end
				Failure: Error code
WEBSOCKET	SUBSCRIBE	*../stats/{tenant_id}/{stat_id}/alarms/{alarm_id}* Subscribe to an alarm	Input	*tenant_id* *stat_id* *alarm_id*
			Output	Alarm flag
WebSockets	ASYNC	notification event	Output	*tenant_id* *stat_id* *alarm_id* *alarm_info*

8.4.2. Information Model

The Statistics service information model supports the collection of samples of different types of information: bytes transmitted/received, bytes stored, free storage space, CPU load, etc., a simple data processing (e.g. aggregation of samples), and setting up alarms for monitoring. Note that this information is collected per tenant_id.

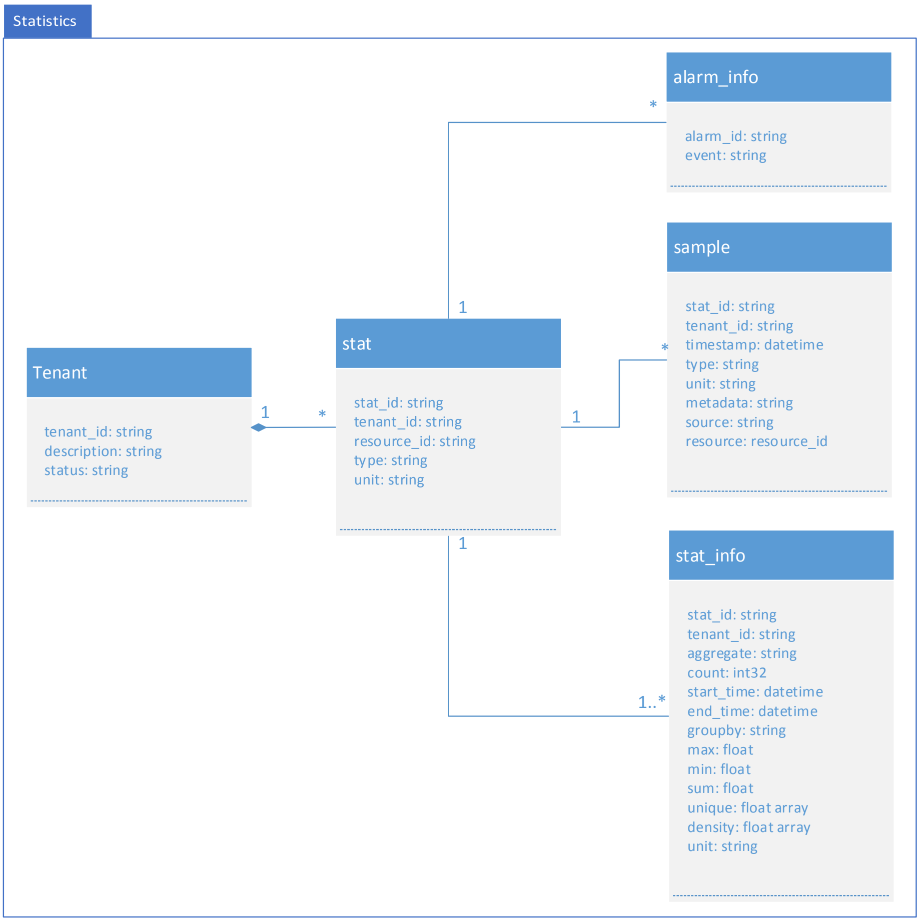


Figure 17: Statistics information model.

In the following table, the main data objects are presented in more detail.

Table 10: Statistics information model.

Parameter	Type	Description
stat_id	Integer	Unique identifier of a type of data (bytes transmitted/received, storage used, storage free). The identifier is unique across different tenants which removes the need to specify a *tenant_id.*
tenant_id	Integer	Unique identifier of a virtual tenant

alarm_info	Object	It contains an *alarm_id* to uniquely identify the alarm and a description of the event that triggers the alarm, e.g. when bytes transmitted reaches a threshold.
sample	Object	Object that contains a sample collected for a *stat*, including timestamps, source of the sample and other metadata.
stat_info	Object	This object is aimed to generate and store process information for a stat. This structure may contain a numeric value (e.g. maximum sample in a certain period of time), or an array of samples (number of occurrence of each unique sample). An important element of this object is the *aggregate* function which let us request personalized processing of a data shape by providing a mathematical function.

8.4.3. Workflow

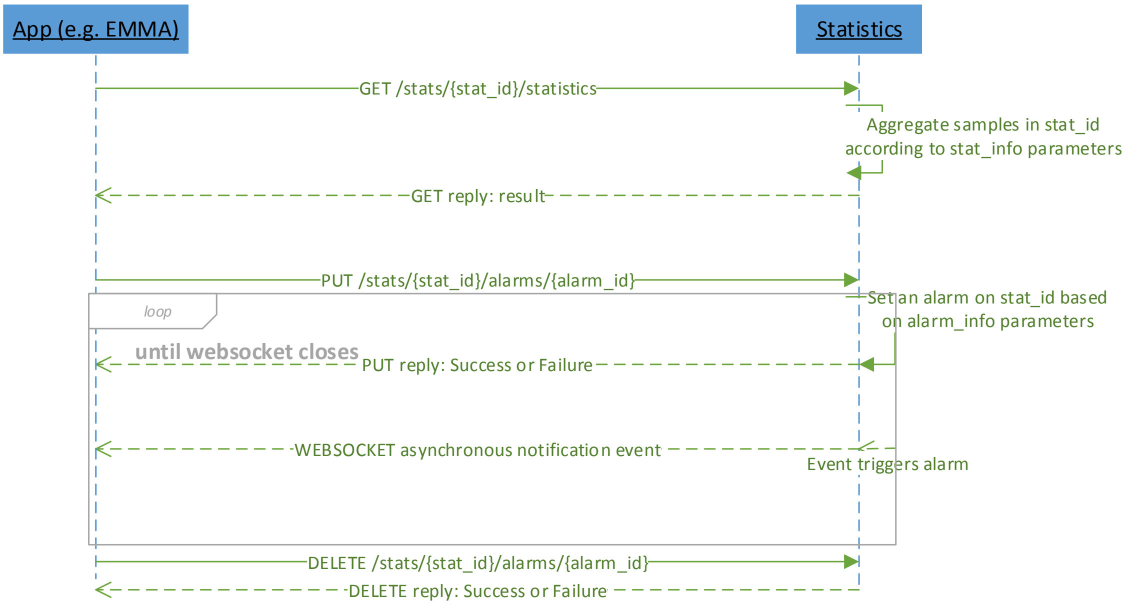

App (e.g. EMMA)
Statistics
GET /stats/{stat_id}/statistics
Aggregate samples in stat_id
according to stat_info parameters
GET reply: result
PUT /stats/{stat_id}/alarms/{alarm_id}
loop
Set an alarm on stat_id based
on alarm_info parameters
until websocket closes
PUT reply: Success or Failure
WEBSOCKET asynchronous notification event
Event triggers alarm
DELETE /stats/{stat_id}/alarms/{alarm_id}
DELETE reply: Success or Failure

Figure 18: Statistics workflow example.

Figure 18 illustrates how a consumer (e.g., an application like EMMA) can poll for statistics performed over a dataset:

1. Consumer requests Statistics service to aggregate a set of samples of *stat_id* according to certain parameters e.g. within a period of time, and an aggregate function, e.g. average. The statistics service replies with the processing result.
2. A consumer may set up an alarm should an event occur. A WebSocket is created so that Statistics service can notify asynchronously the consumer when such an event occurs.

8.5. Virtual Infrastructure Manager and Planner

The NBI of the VIMaP service is based on the REST paradigm. The VIMaP is in charge of performing CRUD operations for the network service layout elements (a set of VMs and their interconnections) for the ETSI NFV architecture [4]. The VIM also offers an API to conduct CRUD operations for the network slice or virtual infrastructure concept (i.e., a set of not only VMs but also virtual switches, and virtual routers associated to the creation of virtual infrastructures for the Mobile Virtual Network Operator (MVNO) use case. As for the former, it corresponds to the deployment of Network Service elements as defined within the ETSI MANO architecture. In particular, it is in line with the ETSI use case #4 VNF Forwarding Graphs in [20]. As for the latter, it tackles the instantiation of virtual infrastructures with ultimate user control composed by a coherent set of network, compute, and storage infrastructure. In this case, the infrastructure is totally provided to the tenant (e.g., XFEs, cards, ports) including XPU resources. In fact, the VIMaP main goal is to offers to the consumer the services offered by the SDN and IT (compute and storage) controller in a unified manner.

8.5.1. APIs

In what follows, we provide a description of the APIs offered by the IT infrastructure and inventory services.

Table 11: Virtual Infrastructure Manager and Planner API.

Prot.	**Type**	**URI**	**Parameters**	
REST	POST	*../vimap/tenant/{tenant_id}/vm* Create a new VM for *tenant_id.*	Input	*vm_object* *tenant_id*
			Output	*vm_id*
REST	GET	*../vimap/tenant/{tenant_id}/vm/{vm_id}* Get information for *vm_id* in tenant *tenant_id.*	Input	*vm_id* *tenant_id*
			Output	*vm_object*
REST	DELETE	*../vimap/tenant/{tenant_id}/vm/{vm_id}* Delete VM *vm_id* for	Input	*vm_id* *tenant_id*
			Output	Success: Status Code of normal end

		tenant_id.		Failure: Error code
REST	PUT	*../vimap/tenant/{tenant_id}/vm/{vm_id}* Update *vm_id* information with *vm_object* in tenant *tenant_id.*	Input	*vm_id* *tenant_id* *vm_object*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	GET	*../vimap/tenant/{tenant_id}/vm* Retrieve all VM elements in *tenant_id.* *../ vimap / tenant/{tenant_id}/ vm?filter={“networkId”:”net1”}* Retrieve all VM elements within *net1*	Input	*tenant_id* *filter* (optional) *JSON_Object (*Object containing a key/value array with properties to filter the list)
			Output	*vm_list*
REST	POST	*../vimap/tenant/{tenant_id}/connectivity_service/* Specify connectivity provisioning between endpoints with *call_object* in *tenant_id.*	Input	*tenant_id* *call_object*
			Output	*tenant_id* *call_id*
REST	GET	*../vimap/tenant/{tenant_id}/connectivity_service/{call_id}* Get connectivity provisioning object between endpoints identified by *call_id* in *tenant_id.*	Input	*tenant_id* *call_id*
			Output	*call_object*

REST	PUT	*../vimap/tenant/{tenant_id}/connectivity_service/{call_id}* Modify connectivity provisioning between endpoints in *tenant_id* specified in *call_id*.	Input	*tenant_id* *call_id* *call_object*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	DELETE	*../vimap/tenant/{tenant_id}/connectivity_service/{call_id}* Delete connectivity provisioning between endpoints in *tenant_id.*	Input	*tenant_id* *call_id*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	GET	*../vimap/tenant/{tenant_id}/connectivity_service* Obtain connectivity provisioning information between endpoints in *tenant_id*	Input	*tenant_id*
			Output	*connectivity_service_list*
REST	GET	*../vimap/tenant/{tenant_id}/hypervisor/{hypervsisor_id}* Get hypervisor information of hypervisor *hypervisor_id* in tenant *tenant_id.*	Input	*tenant_id* *hypervisor_id*
			Output	*hypervisor_object*
REST	GET	*../vimap/tenant/{tenant_id}/hypervisor* Get list of hypervisors in tenant *tenant_id.*	Input	*tenant_id*
			Output	*hypervisor_list*
REST	GET	*../vimap/tenant/{tenant_id}/network_topology/{network_id}*	Input	*tenant_id* *network_id* (sec.2.1)

		Get network information for *network_id* in *tenant_id.*	Output	*network_object* (sec.2.1)
REST	POST	*../vimap/ tenant/{tenant_id}/ Slice_provisioning_service* Creation of a slice specified by *slice_object* in *tenant_id.*	Input	*tenant_id* *slice _object*
			Output	*tenant_id* *slice_id*
REST	GET	*../vimap/ tenant/{tenant_id}/ Slice_provisioning_service/{slice_id}* Obtain information details of *slice_id* in *tenant_id.*	Input	*tenant_id* *slice_id*
			Output	*slice _object*
REST	PUT	*../vimap/ tenant/{tenant_id}/ Slice_provisioning_service/{slice_id}* Update information details of *slice_id* in *tenant_id* with *slice_object.*	Input	*tenant_id* *slice_id* *slice _object*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	DELETE	*../vimap/ tenant/{tenant_id}/ Slice_provisioning_service/{slice_id}* Delete *slice_id* in *tenant_id.*	Input	*tenant_id* *slice_id*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	GET	*../vimap/ tenant/{tenant_id}/ Slice_provisioning_service* Obtain information details of all slices in *tenant_id.*	Input	*tenant_id*
			Output	*slice_objects_list*
REST	POST	*../vimap/ tenant/{tenant_id}/ NetworkServiceSupportLayout*	Input	*tenant_id* *netserv_layout_object*

		Create *netser_layout_object* in *tenant_id*.	Output	*tenant_id* *netserv_layout_id*
REST	GET	*../vimap/tenant/{tenant_id}/NetworkServiceSupportLayout/{ netserv_layout_id}* Obtain information details of *netserv_layout_id in tenant_id.*	Input	*tenant_id* *netserv_layout_id*
			Output	*netserv_layout_object*
REST	PUT	*../vimap/tenant/{tenant_id}/NetworkServiceSupportLayout/{ netserv_layout_id}* Update netserv_*layout_id* with *netser_layout_object* in *tenant_id*.	Input	*tenant_id* *netserv_layout_id* *netserv_layout_object*
			Output	Success: Status Code of normal end
				Failure: Error code
REST/	DELETE	*../vimap/tenant/{tenant_id}/NetworkServiceSupportLayout/{ netserv_layout_id}* Delete netserv_*layout_id in tenant_id.*	Input	*tenant_id* *netserv_layout_id*
			Output	Success: Status Code of normal end
				Failure: Error code
REST/	GET	*../vimap/tenant/{tenant_id}/NetworkServiceSupportLayout* Obtain information details of all network service layouts in *tenant_id.*	Input	*tenant_id*
			Output	*netserv_layout_objects_list*
REST/	GET	*../vimap/tenant/{tenant_id}/streams/topology_update/{network_id}* Obtain URL to subscribe to network topology events.	Input	*tenant_id* *network_id* (sec.2.1)
			Output	URL to subscribe to notification service (e.g., websocket)

REST	GET	*../vimap/ tenant/{tenant_id}/ streams/ connectivity_service_update/{ call_id}* Obtain URL to subscribe to connectivity service events.	Input	*tenant_id* *call_id*
			Output	URL to subscribe to notification service (e.g., websocket)
WEBSOCKET	SUBSCRIBE	*../vimap/ tenant/{tenant_id}/ streams/ connectivity_service_update/{ call_id}* Subscription to connectivity. service specified by *call_id.*	Output	*call_object*

8.5.2. Information Model

The VIMaP information data model supports the concept of *tenant*. A tenant, at the VIMaP service, may be composed of different slices (virtual infrastructure use case) and different network service layouts (OTT use case). Both the network *slice* and *network_service_layout* objects are considered a virtual network from the point of view of the IT infrastructure and inventory service. A slice or a network service layout entails a set of calls defining the flow patterns between these components.

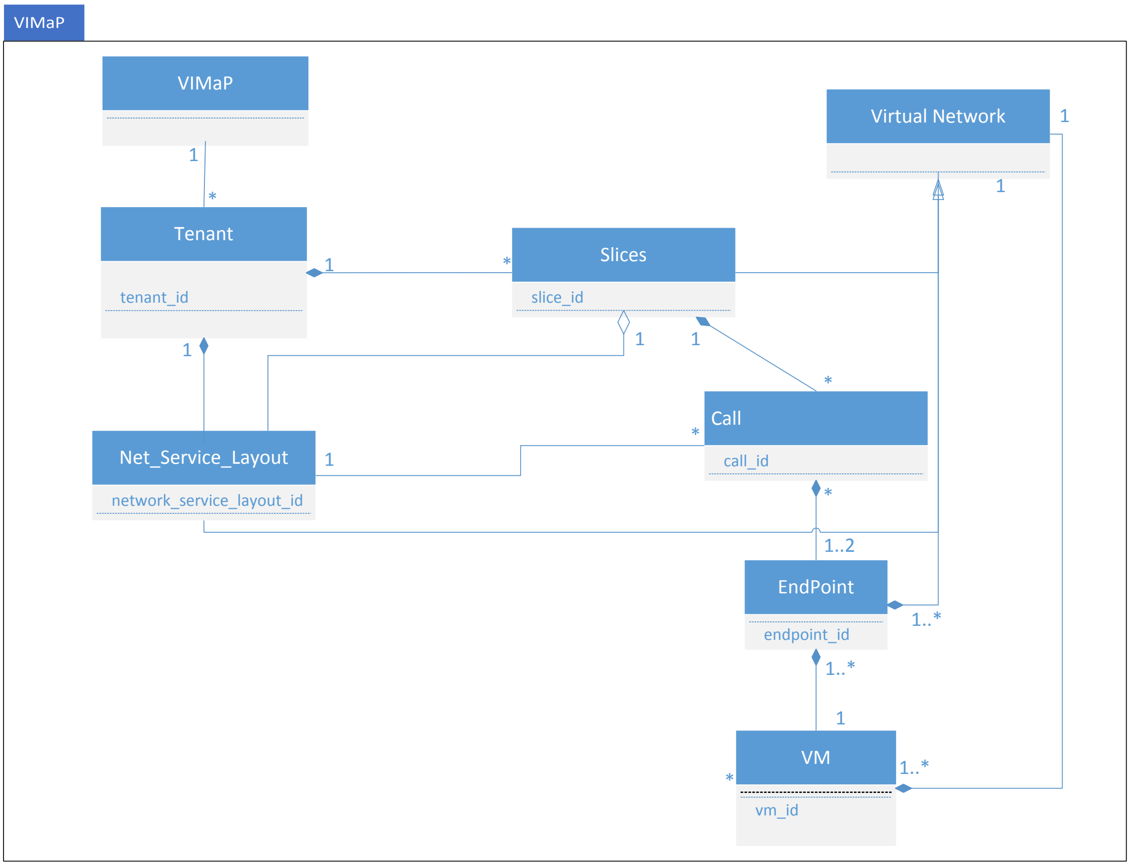


Figure 19: Virtual Infrastructure Manager and Planner information model.

In what follows, see a more detailed information of the most relevant objects forming part of the VIMaP NBI service.

Table 12: Virtual Infrastructure Manager and Planner information model.

Parameters	**Type**	**Description**
tenant_id	String	Identifier of the tenant who is requesting the VIMaP service.
vm_id	String	Identifier of the Virtual Machine (VM).
vm_object	Object	Object describing the VM as a set of properties in JSON or XML format: Parameters (variable; type; description): • Name; String; name of the VM. • Flavor**;** String; Hardware template. • Image_name; String; VM template. • NetworkId; String; L2 Network identifier. • SubnetId; String; L3 Network identifier. • Id; String; unique identifier of the VM.

		• MAC; String; L2 address • IP; Ipv4; L3 address • Hypervisor; String; Identifier of the compute (physical) node on which is deployed the VM. • Node_id; String; DpId of the switch attached • Endpoint_id; String; Network identifier of where the VM is attached.
call_id *call_object*	String Object	Identifier of the connectivity service (Call) Intent-based connectivity service request object described as a set of properties in JSON or XML format: Parameters (variable; type; description): • aEnd**;** Endpoint; Source connectivity service endpoint • zEnd**;** Endpoint; Destination connectivity service endpoint • transport_layer; TransportLayerType; Connectivity service description (L0, L2, L3…) • traffic_parameters; TrafficParams; QoS parameters describing the connectivity service (Bandwidth, Latency…).
Endpoint_id	String	Unique identifier of the endpoint.
Endpoint_object	Object	The endpoint object represents the termination point of a network connectivity service. It is described as a set of properties in JSON or XML format: Parameters (variable; type; description): • endpoint_id; String; Unique identifier of the termination point of a network connectivity service (Call). • node_Id; String; Identifier of the network node to which the endpoint is connected. • net_interface_id; String; Identifier of the network port component to which the endpoint is connected.
slice_id *Slice _object*	*String* *Object*	Identifier of the virtual infrastructure slice (Slice) Slice object described as a set of properties in JSON or XML format: Parameters (variable; type; description): • virtual_IT_infrastructure; JSON; object containing the description of the virtual IT infrastructure requested. It should contain the description of computing and storage resources,

		requested for the slice. • virtual_tenant_network; VTN (sec 2.10.1); abstract representation of the network slice requested. • network_service_layout: NetworkServiceSupportLayout; object containing a set of service endpoints (VMs) connected by a graph representation (layout). • control_stack: JSON: object describing the Software-Defined control stack for the slice. This object may contain the addressing and security parameters to access the control instances.
netserv_layout _object	Object	A network service layout object described as a set of properties in JSON or XML format: Parameters (variable; type; description): • service_endpoints; list(Endpoint); list of service endpoints • virtual_machine_list; list(VM); list of VMs. • transport_layer; TransportLayerType; Connectivity service description (L0, L2, L3…) • traffic_parameters; TrafficParams; QoS parameters describing the connectivity service (Bandwidth, Latency…). • topology_layout; Topology; graph representation of service endpoints connectivity.

8.5.3. Workflow

In what follows, we provide a message exchange sequence to illustrate the use of the VIMaP service by an NFV-O when managing the creation and management of OTT network services. In this example, the consumer is the NFV-O, which is located inside the XCI. The goal is to illustrate the use of the VIMaP services for the ETSI NFV use case. In particular, the workflow in Figure 20 illustrates the creation of a network service layout composed by a set of VMs and their direct interconnection.

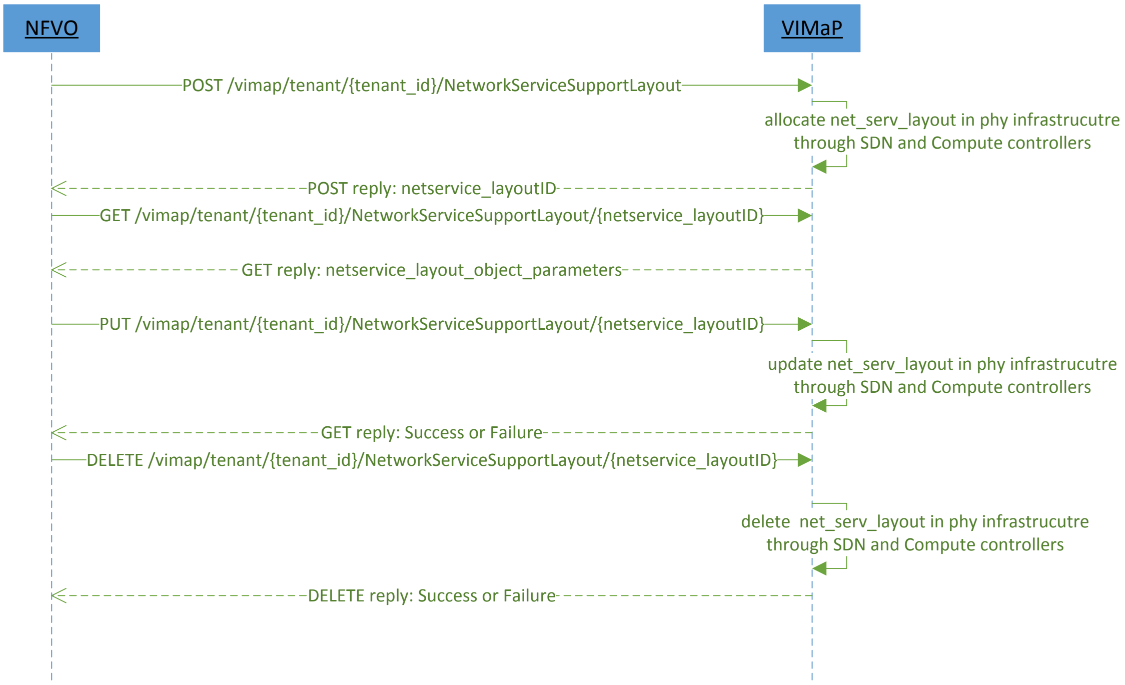

NFVO
VIMaP
POST /vimap/tenant/{tenant_id}/NetworkServiceSupportLayout
allocate net_serv_layout in phy infrastrucutre through SDN and Compute controllers
POST reply: netservice_layoutID
GET /vimap/tenant/{tenant_id}/NetworkServiceSupportLayout/{netservice_layoutID}
GET reply: netservice_layout_object_parameters
PUT /vimap/tenant/{tenant_id}/NetworkServiceSupportLayout/{netservice_layoutID}
update net_serv_layout in phy infrastrucutre through SDN and Compute controllers
GET reply: Success or Failure
DELETE /vimap/tenant/{tenant_id}/NetworkServiceSupportLayout/{netservice_layoutID}
delete net_serv_layout in phy infrastrucutre through SDN and Compute controllers
DELETE reply: Success or Failure

Figure 20: Virtual Infrastructure Manager and Planner workflow example.

As shown in Figure 20, this process can consist of the major steps:

1. The NFV-O creates a network service layout formed by a set of VMs.
   1.1. The VIMaP conducts the necessary operations to create and interconnect the VMs forming the network service layout
   1.2. Note that in this case the default algorithm used by VIMaP offers the logic for the placement of the VMs. However, this does not preclude the specification of the placement of VMs by other entities (e.g., the NFV-O or the MTA).
2. In case the VIMaP can allocate the indicated set of VMs (with their associated characteristics specified by a template) returns an id of the successfully created network service layout.
3. The NFV-O can request the characteristic and status of the created network service layout.
4. At a given point, the NFV-O can update the characteristics or the template of the previously created network service layout. For instance, it can request to change the location, the characteristics of a given VM, or the way the VMs are connected between them. This change triggers the interaction of the VIMaP either with the SDN or compute controllers, or with both entities.
5. At a given point in time, the NFV-O may decide to deallocate the network service layout from the physical infrastructure. This requires the interaction of the VIMaP with both the SDN and compute controllers.

8.6. NFV Orchestrator

The NFV-O offers an NBI that allows 5G-Crosshaul applications to request the instantiation, orchestration and management of Network Services (NSs). A NS, as in ETSI NFV terminology, is composed by multiple virtual or physical network functions (VNFs or PNFs), which are interconnected through a VNF Forwarding Graph.

8.6.1. APIs

The NFV-O provides mechanisms to create, retrieve and remove NSs, as described in the following table.

Table 13: NFV-O API.

Prot.	Type	URI	Parameters	
REST	POST	*../nfvo/ns* Create a new network service for a given tenant	Input	*NS Id* *ServiceDeploymentFlavour* *NS Tenant Id*[2]
			Output	*NS Id*
REST	GET	*../nfvo/ns/ns_id* Retrieve information about the given network service	Input	*NS Id* *NS Tenant Id*
			Output	*NS record*
REST	DELETE	*../nfvo/ns/ns_id* Remove an existing network service	Input	*NS Id* *NS Tenant Id*
			Output	Success: Status Code of normal end
				Failure: Error code

8.6.2. Information Model

The main entity managed by the NFV-O is the Network Service (NS), a chain of VNFs interconnected through a VNF Forwarding Graph (VNFFG). The characteristics of a NS are defined according to a standard template, called NS Descriptor (NSD), which defines:

[2] A tenant is one or more NFV MANO service users sharing access to a set of physical, virtual or service resources. An NS tenant is a tenant to which NSs are assigned. (See ETSI GS NFV-IFA 010, v2.1.1, April 2016)

- NS generic information, like vendor, version, human readable description, etc.
- The VNFs that compose the NS, identified through their VNF Descriptors (see the VNFM in section 8.7).
- The Physical Network Functions (PNF) that compose the NS (optional).
- The Virtual Network Function Forwarding Graphs (VNFFGs) that interconnect the VNFs (and the PNFs if available), identified through their VNFFG Descriptor. Both Virtual Network Function Descriptors (VNFDs) and VNFFG Descriptors are stored in repositories at the NFV-O level.
- The dependencies between the VNFs.
- The scripts and configuration parameters to be launched at the various stages of the NS lifecycle (e.g. during the instantiation, scale up/down or termination).
- The KPIs to be monitored.
- The criteria and the constraints for automated and on-demand scaling of the NS.

The VNF Descriptior information model is reported in Figure 24. VNFDs are stored in the VNFD Database (DB), a shared DB which is accessed by both VNFM and NFV-O. Suitable management APIs are exposed by the NFV-O to load new VNFDs in the repository. This procedure is defined by ETSI NFV MANO standard and it is out of the scope of this document. During the instantiation of a VNF, the VNFD is specified in the request through its unique identifier.

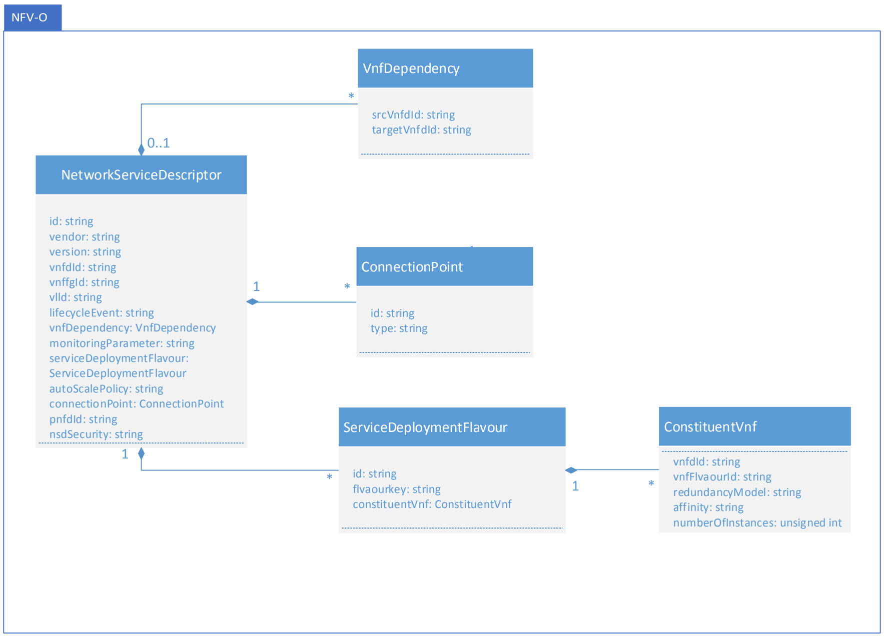


Figure 21: NFV-O information model.

Next, the main data objects are presented in more detail:

Table 14: NFV-O information model.

Parameter	Type	Description
NS Id	String	Descriptor of the NS to be instantiated, , including deployment flavours, external connections points and dependencies among NS components.
ServiceDeploymentFlavour	Object	Defines the size, the characteristics and the components (VNFs and virtual links) of the NS to be instantiated.
NS record	Object	Description of the instantiated NS, its VNF instances, its status and its parameters

8.6.3. Workflow

Figure 22 and Figure 23 show the workflow to create and terminate a Network Service when triggered by a generic NFV-O client. In the two figures, we have assumed a simplified scenario where the Network Service includes only VNFs, without any Physical Network Function (PNFs). If this requirement is not met, an additional interaction between the NFV-O and the SDN controller responsible for the physical network infrastructure would be required in order to enable the interconnection between the VNFs and the PNFs at the physical network level (which is not managed by the VIM). Moreover, we are also assuming the following:

1. The VNF Managers are already up and running.
2. No preliminary check of resource availability or resource reservation is performed before allocation (these actions are considered as optional in ETSI NFV specifications, but are usually not supported at the VIM level in state-of-the-art cloud platforms).

The detailed workflows for the instantiation and termination of VNFs are described in the section dedicated to the VNFM (see Section 8.7).

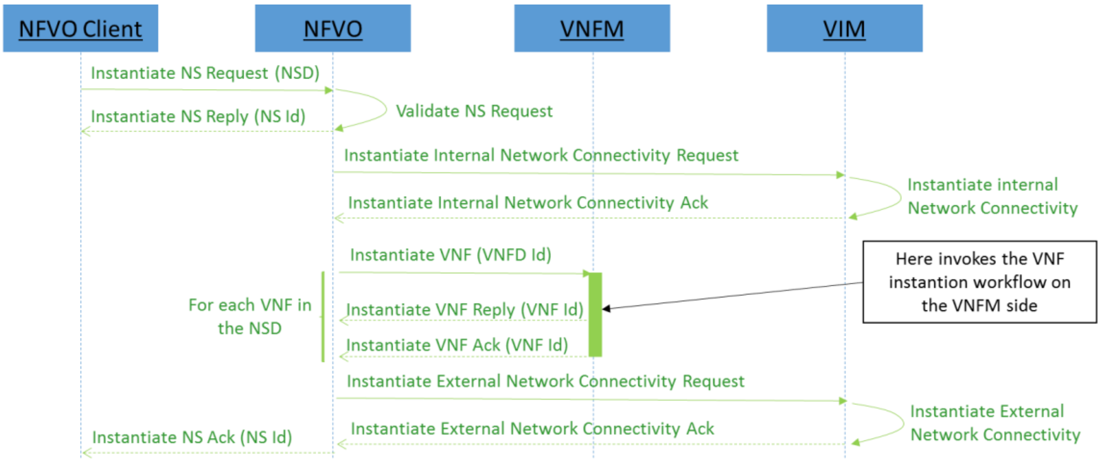

NFVO Client
NFVO
VNFM
VIM
Instantiate NS Request (NSD)
Validate NS Request
Instantiate NS Reply (NS Id)
Instantiate Internal Network Connectivity Request
Instantiate internal Network Connectivity
Instantiate Internal Network Connectivity Ack
Instantiate VNF (VNFD Id)
Here invokes the VNF instantion workflow on the VNFM side
For each VNF in the NSD
Instantiate VNF Reply (VNF Id)
Instantiate VNF Ack (VNF Id)
Instantiate External Network Connectivity Request
Instantiate External Network Connectivity
Instantiate External Network Connectivity Ack
Instantiate NS Ack (NS Id)

Figure 22: NFV-O workflow example: Instantiate NS.

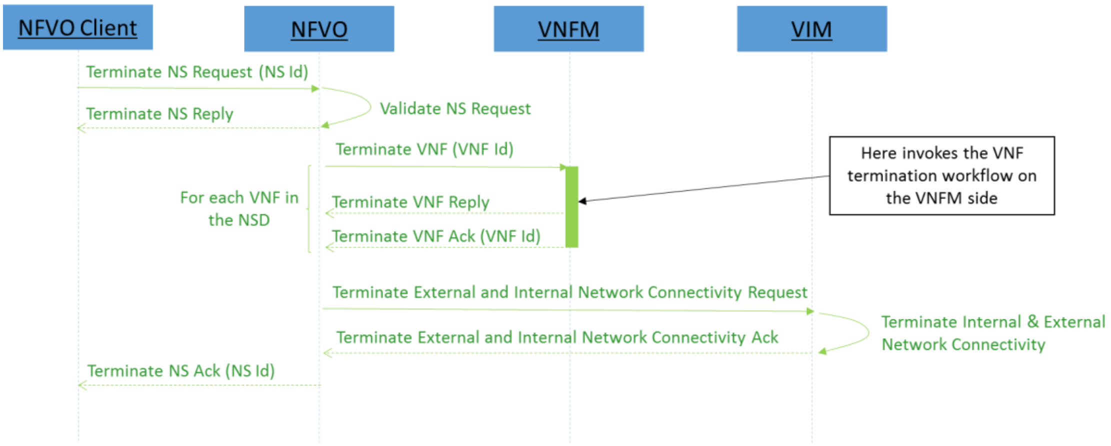

NFVO Client
NFVO
VNFM
VIM
Terminate NS Request (NS Id)
Validate NS Request
Terminate NS Reply
Terminate VNF (VNF Id)
Here invokes the VNF termination workflow on the VNFM side
For each VNF in the NSD
Terminate VNF Reply
Terminate VNF Ack (VNF Id)
Terminate External and Internal Network Connectivity Request
Terminate Internal & External Network Connectivity
Terminate External and Internal Network Connectivity Ack
Terminate NS Ack (NS Id)

Figure 23: NFV-O workflow example: Terminate NS.

8.7. VNF Manager

This module exposes NBI services to manage single VNF instances. In particular, this module offers an API to the NFV-O to conduct CRUD operations in VNFs. The role of this service is aligned with the role specified by ETSI NFV [4]. An open source implementation compliant with the ETSI NFV specification can be found, for instance, in OpenBaton [6], bundled with an NFV-O.

8.7.1. APIs

The VNF Manager provides mechanisms to create, retrieve and remove VNFs, as described in the following table.

Table 15: VNF Manager API.

Prot.	Type	URI	Parameters	
REST	POST	*../vnfm/vnf* Create a new VNF for a given tenant.	Input	*VNFD Id* *VNF Tenant Id*[3] *Deployment flavour Id* *NS Id* External virtual links Ids (List<String>): identify the external virtual links the VNF must be connected to (through its external connection points).
			Output	*VNF ID*
REST	GET	*../vnfm/vnf/vnf_id* Retrieve the information related to a given VNF.	Input	*VNF Id* *VNF Tenant Id*
			Output	*VNF record*
REST	DELETE	*../vnfm/vnf/vnf_id* Remove an existing VNF.	Input	*VNF Id* *VNF Tenant Id*
			Output	Success: Status Code of normal end
				Failure: Error code

8.7.2. Information Model

The main entity managed by the VNF Manager is the VNF. The VNFs are defined according to a standard template, called VNF Descriptor (VNFD), which defines:

- VNF generic information, like vendor, version, human readable description, etc.
- The VNF Components (VNFC), which compose the VNF and their characteristics, through the definition of associated Virtual Deployment Units (VDUs).
- The internal and external Connection Points and the virtual links the VNF and VNFCs are attached to.
- The dependencies between the VNFC, i.e. between VDUs.
- The scripts and configuration parameters to be launched at the various stages of the VNF lifecycle.
- The KPI to be monitored at the whole VNF and single VNFCs level.

[3] A tenant is one or more NFV MANO service users sharing access to a set of physical, virtual or service resources. A VNF tenant is a tenant to which VNFs are assigned. (See ETSI GS NFV-IFA 010, v2.1.1, April 2016)

- The criteria and the constraints for automated and on-demand scaling of the VNF.

The VNF Descriptor (VNFD) information model is reported in Figure 24. VNFDs are stored in the VNFD DB, a shared DB which is accessed by both VNFM and NFV-O. Suitable management APIs are usually exposed by the NFV-O to load new VNFDs in the repository. This procedure is defined by ETSI NFV MANO standard and it is out of the scope of this document. During the instantiation of a VNF, the VNFD is specified in the request through its unique identifier.

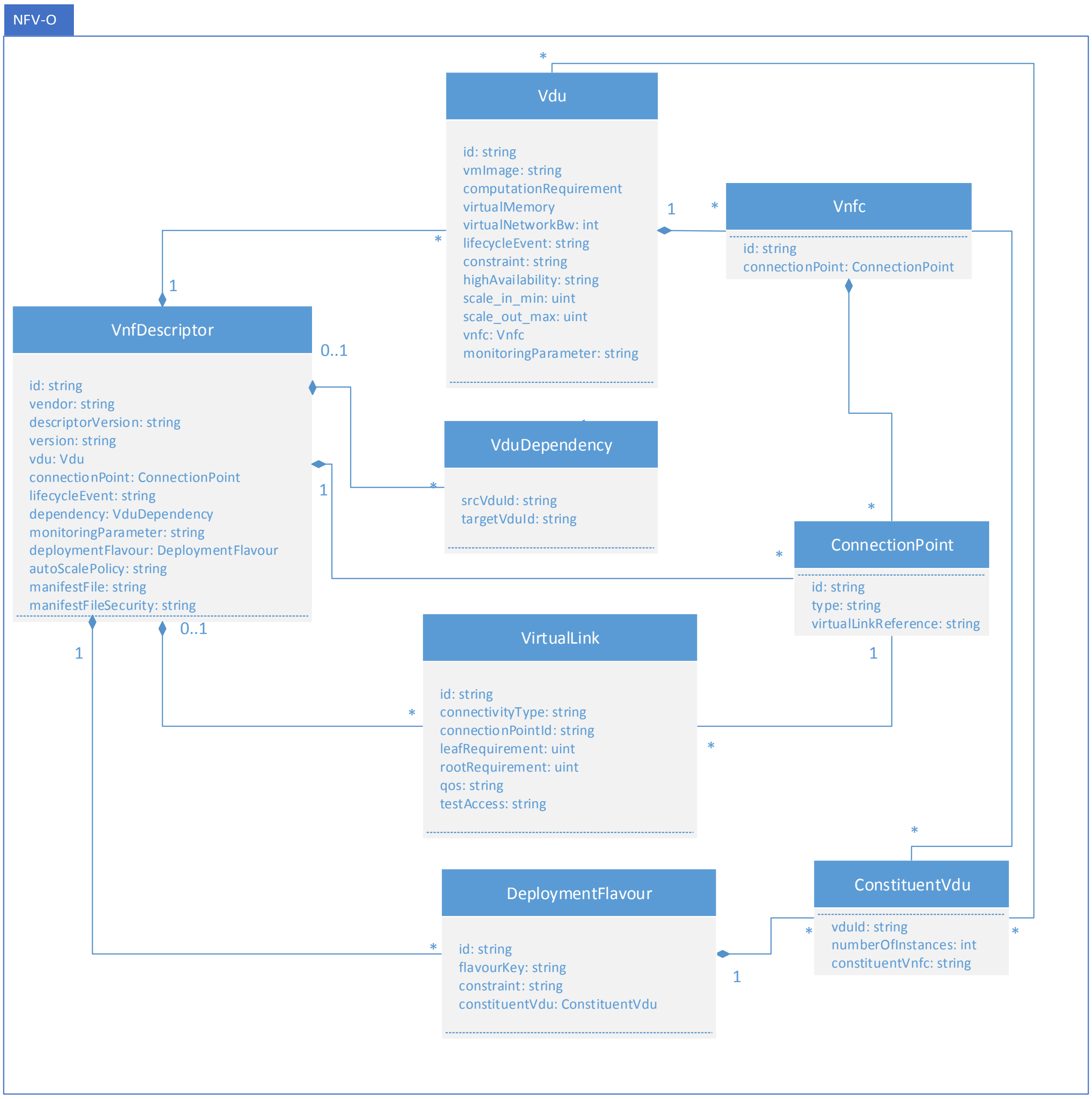


Figure 24: VNF descriptor information model.

Next, the main data objects are presented in more detail:

Table 16: VNF Manager information model.

Parameter	Type	Description
VNFD Id	String	ID of the VNFD according to the VNF needs to be instantiated.
Deployment flavour Id	Object	Defines the size and the characteristics of the VNF to be instantiated.
VNFD descriptor	Object	Description of the VNF to be instantiated, including its elements (e.g. VNFC and VDU, virtual links, connection points).
VNF record	String	ID of the VNFD according to the VNF needs to be instantiated.

8.7.3. Workflow

This section describes the workflows for instantiation and termination of single VNFs, modelled according to the option of resource allocation specified by the NFV-O. In other terms, while the VNFM is responsible for coordinating the whole instantiation procedure, the request for resource allocation to the VIM is mediated by NFV-O. These workflows are parts of the whole Network Service instantiation and termination workflows; in this case, the VNFM client is actually the NFV-O itself.

As shown in Figure 25, the VNFM receives a request to instantiate a VNF, receiving as input the VNF Descriptor (VNFD), together with other parameters which indicate the size of the VNF (i.e. its deployment flavour) and how the VNF must be interconnected to the whole Network Service (e.g. through the specification of already established external virtual links). The VNFM generates a VNF Id which is immediately returned and used as reference ID for further asynchronous notifications or requests related to the lifecycle of that VNF. The VNFM elaborates the VNFD and identifies the virtual resources (network, storage, computing) which must be allocated to build all the VNF Components (VNFCs) and the internal virtual links that compose the VNF itself. The resulting resources are requested to the NFV-O that can optionally perform some algorithms to decide the optimal resource placement and finally forwards the request to the VIM. The VIM allocates all the resources, typically starting from the network side, and when finished notify the NFV-O which, in turns, sends an acknowledgement to the VNFM. Once the allocation procedure is finished, the VNFM takes care of the configuration of the VNF and its VNFCs and finally notifies the originating requester about the instantiation result.

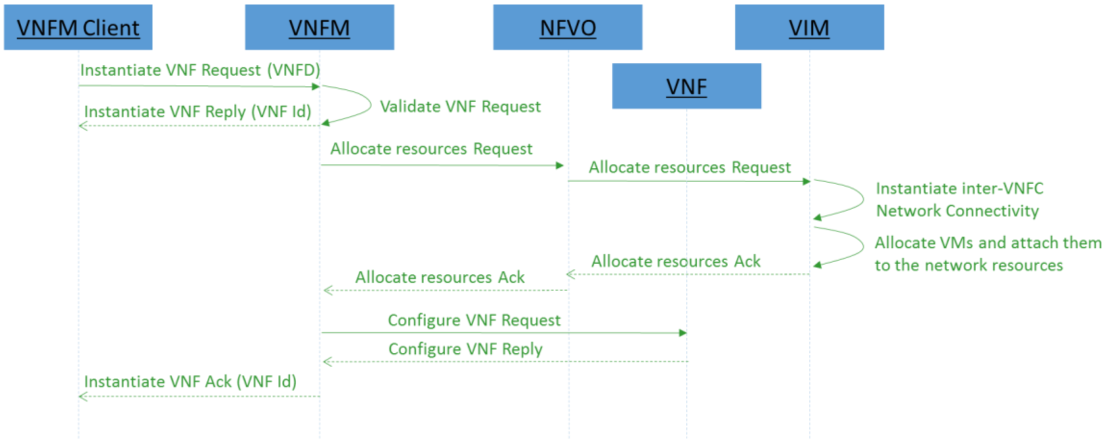

VNFM Client
VNFM
NFVO
VIM
VNF
Instantiate VNF Request (VNFD)
Instantiate VNF Reply (VNF Id)
Validate VNF Request
Allocate resources Request
Allocate resources Request
Instantiate inter-VNFC Network Connectivity
Allocate VMs and attach them to the network resources
Allocate resources Ack
Allocate resources Ack
Configure VNF Request
Configure VNF Reply
Instantiate VNF Ack (VNF Id)

Figure 25: VNF Manager workflow example: Instantiate VNF.

Figure 26 shows the termination procedure. The VNFM sends the VNF the commands required to (gracefully) shutdown the running applications, as specified in the VNFD. Then it asks the NFV-O to delete the virtual resources previously allocated, an action which is executed interacting with the VIM to remove VMs and network resources. Once all the resources are deleted, the originating requester is information with an asynchronous message.

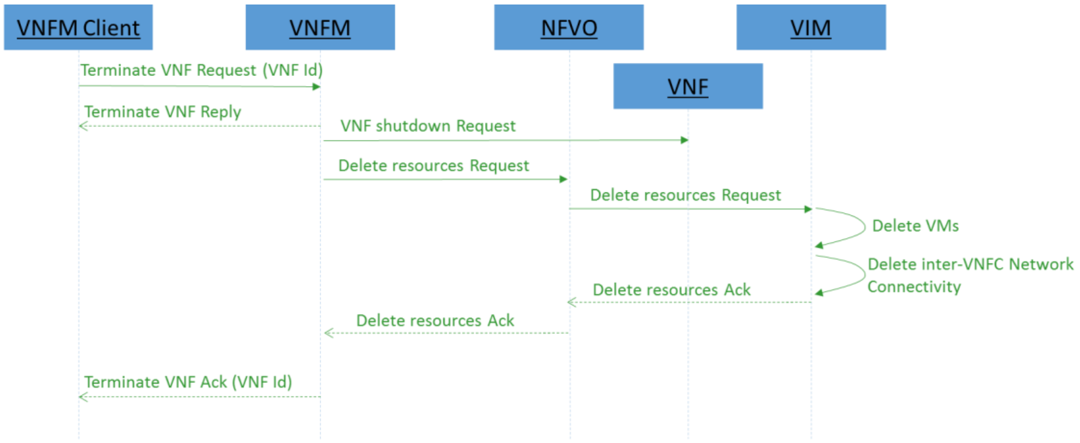


Figure 26: VNF Manager workflow example: Terminate VNF.

8.8. Analytics for Monitoring

This service is in charge of offering to the consumer elaborated information obtained from the processing of the network and computing statistics gathered by the stats module specified in Section 8.4. Analytics involve studying data to evaluate the performance or to analyze the effects of certain decisions or events, and produces specific results. This elaborated information could be, for instance, used by the Energy

Management and Monitoring Application (EMMA) to determine whether it is convenient to change the power status of some XFE or XPU in the infrastructure.

8.8.1. APIs

The analytics for monitoring module through the Northbound APIs provided in the following table allows getting specific information about different resources or defined meters. These APIs also provide primitives to create new meters in case the services require it, where a meter can be seen as a measure of a specific resource. This dataset can be used for subsequent retrieval and analysis, and trigger actions when it is necessary.

Table 17: Analytics for Monitoring API.

Prot.	**Type**	**URI**	**Parameters**	
REST	GET	*../analytics/resources* Retrieve the whole resources *../analytics/resources/*(resource_id) Retrieve details about one resource with the *resource_id*	Input	*resource_id* (optional)
			Output	*list(resource)* *resource*
REST	GET	*../analytics/meters* Return all known meters *../analytics/meters/*(meter_name) Return samples for the meter with the meter_name	Input	*meter_name* (optional)
			Output	list (meter) list (samples)
REST	POST	*../analytics/meters/*(meter_name) Post a new meter	Input	meter_name

			Output	Success: Status Code of normal end
				Failure: Error code
REST	GET	*../analytics/samples* Return all known samples *../analytics/samples/*(sample_id) Return a sample with the sample_id	Input	*sample_id* (optional)
			Output	*list (sample)* *sample*
REST	GET	*../analytics/event_types* Get all event types *../analytics/event_types/*(event_type) Return an event_type *../analytics/events* Return all events matching the filters *../analytics/events/*(event_id) Return an event with the event_id	Input	*event_type* (optional) *event_id* (optional)
			Output	*List (event types)* *event_type* *list (event)* *event*

8.8.2. Information Model

The information data model supported by the analytics for monitoring service is shown in Figure 27:

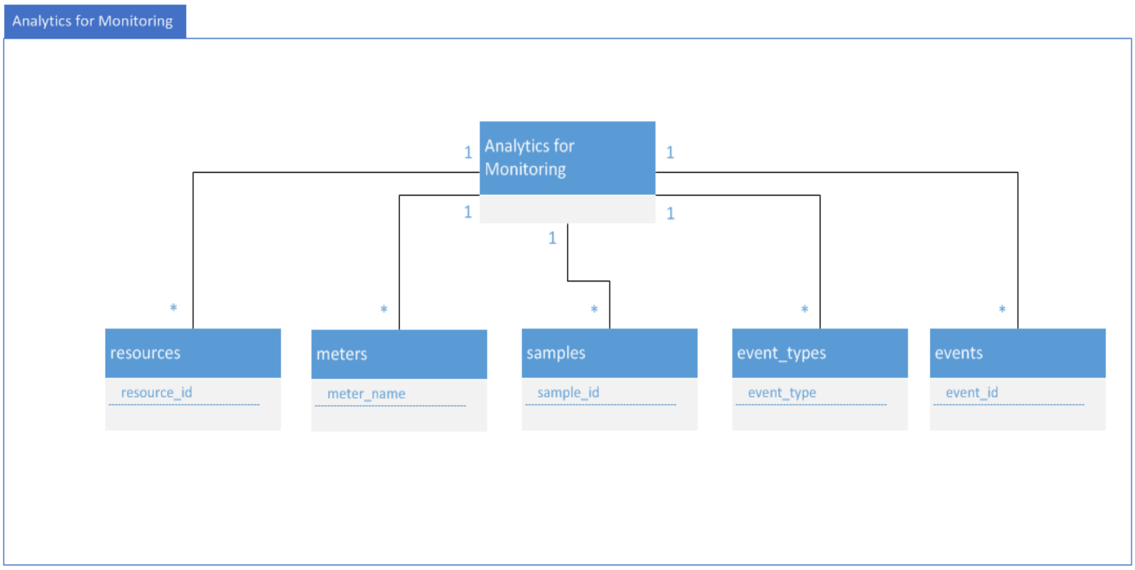


Figure 27: Analytics for Monitoring Information model.

The main parameters are described in the following table:

Table 18: Analytics for Monitoring Information model.

Parameters	Type	Description
resource_id	Unicode	Identifier of the resource
meter_name	Unicode	Identifier of the meter
sample_id	Unicode	Identifier of the sample
event_type	Unicode	Identifier of the event type
event_id	Unicode	Identifier of the event

8.8.3. Workflow

This section shows the workflow followed by an application that requires the analytics for monitoring service. Figure 28 illustrates the request of the samples for a specific meter and the request for creating a new meter for monitoring in these two steps:

1. Gathering the monitoring data from existing services or by polling the infrastructure.
2. Configuring the kind of data gathered to meet different operating requirements.

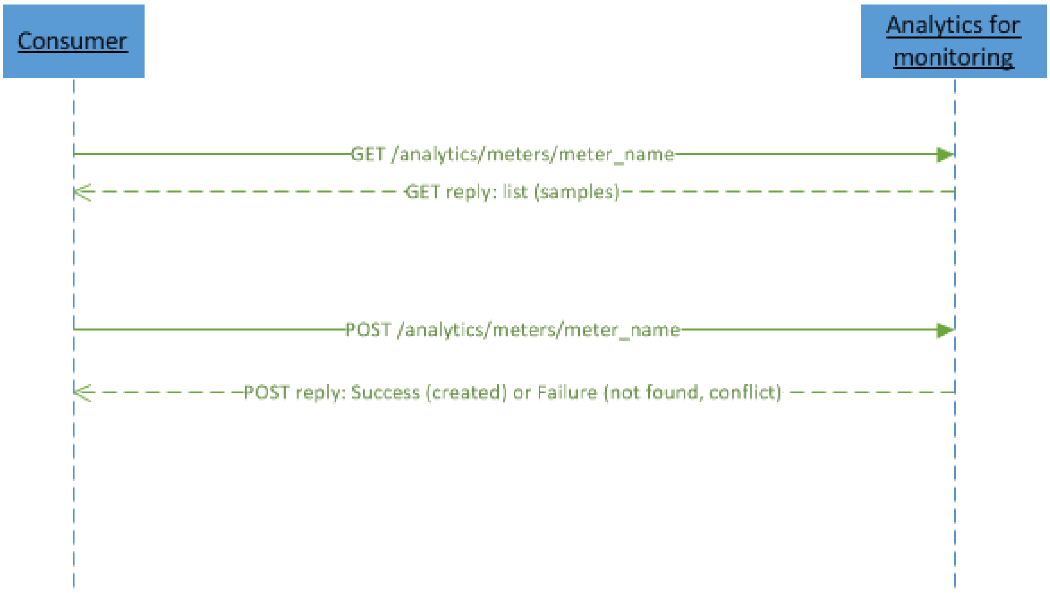

Consumer
Analytics for monitoring
GET /analytics/meters/meter_name
GET reply: list (samples)
POST /analytics/meters/meter_name
POST reply: Success (created) or Failure (not found, conflict)

Figure 28: Analytics for monitoring workflow example.

8.9. Local Management Service

The Local Management Service offers to Northbound applications the possibility of managing the status of the 5G-Crosshaul XFEs and XPUs. By status in an XFE we refer, for instance, to the capability of reconfiguring the properties of a port, or a set of ports. Also, the reconfiguration of XFE and XPU status (e.g., device on or device off) and their associated properties is also considered as a potential feature offered by this service. Consequently, this module will require a DB related to the status of network and IT components that can be potentially modified by a 5G-Crosshaul application.

8.9.1. APIs

The Local Management Service is in charge of the modification of the status of XFEs and XPUs. For instance, it provides the Energy Management and Monitoring Application (EMMA) with REST APIs that can be used to perform the network (re-)configuration (switching on/off physical nodes) if such action leads to the minimization of the energy expenditure while ensuring an acceptable quality of service.

Table 19: Local Management Service API.

Prot.	Type	URI	Parameters	
REST	GET	*../lms/{node*	Input	*Node id*

		id}/stts Retrieve Node Status (On/Off)	Output	Success: *n_status*
				Failure: Error Code
REST	POST	*../lms/{node id}/stts/n_status}* Set Node Status to On or Off	Input	*Node id*
			Output	Success: *n_status*
				Failure: Error Code

8.9.2. Information model

The information data model supported by the Local Management service is shown in Figure 29, and it supports the identification of a specific node and its status.

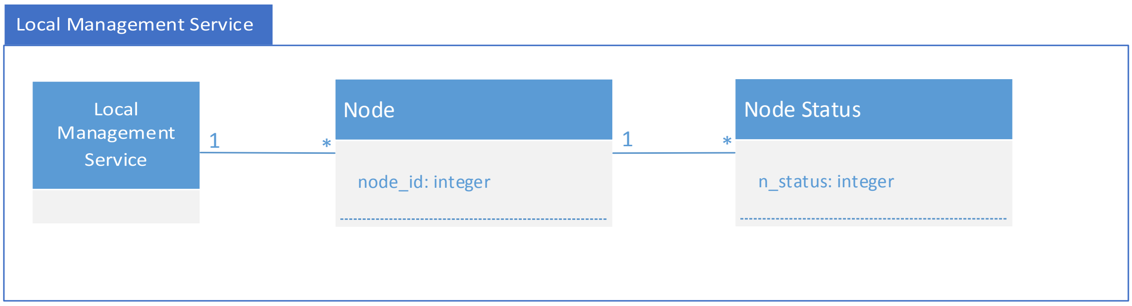


Figure 29: Local Management Service information model

In the following, we describe the relevant parameters:

Table 20: Local Management Service information model.

Parameter	**Type**	**Description**
node_id	Integer	Unique identifier of a physical node
n_status	Integer	Status (e.g., On/Off/Sleep) of a physical node

8.9.3. Workflow

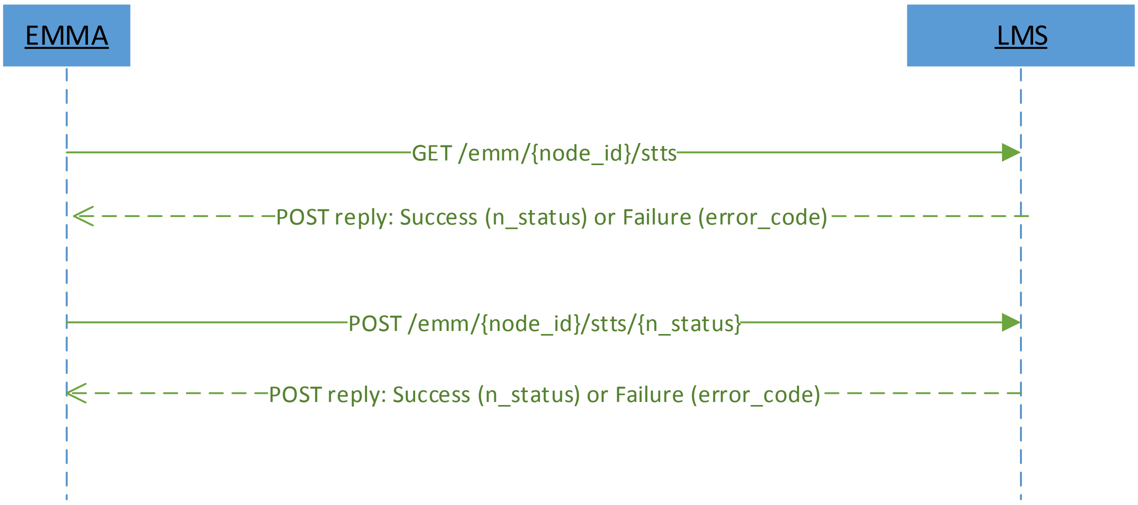

EMMA
LMS
GET /emm/{node_id}/stts
POST reply: Success (n_status) or Failure (error_code)
POST /emm/{node_id}/stts/{n_status}
POST reply: Success (n_status) or Failure (error_code)

Figure 30: Local Management service workflow example.

As shown in the first interaction of the workflow of Figure 30, the Energy Management and Monitoring Application (EMMA) can ask the local management service (LMS) to return the status of a physical node, univocally identified by its node ID. Possible status indicators (*n_status)* are associated to On, Off or Sleep status. This status is then used by EMMA to run its energy saving algorithms. The LMS service can reply with an error code if, e.g., the node is non-existent or its status cannot be determined (if the node has been disconnected). A possible output of EMMA algorithms is the request that a physical node is turned On, Off, or set to Sleep (low energy consumption) state. This can be achieved (second interaction of the workflow of Figure 30) by asking the LMS to set the node to a specific status (On, Off or Sleep status). The LMS service can reply with an error code if, e.g., the node is non-existent or its status changed is denied (different error codes can be defined in order to specify possible reasons why the status change has been denied).

8.10. Multi-tenancy

The Multi-tenancy service allows the MTA application [2] to enforce its decisions on slicing the physical 5G-Crosshaul infrastructure and allocating virtual resources to multiple tenants. It provides REST APIs to map virtual components such as virtual L2/L3 forwarding elements and virtual links that belong to a virtual network to the physical substrate.

8.10.1. APIs

The Northbound APIs in the following table provide primitives to create, modify, delete and visualize the mapping between Virtual Tenant Networks (VTNs) and the physical infrastructure.

Media types can be in the form of JSON and/or XML. It is important to note that the URIs specified in the request by the consumer shall be independent of the chosen representation in the implementation. Table 21 shows the most important functions to create, modify and delete a VTN.

Table 21: Multi-tenancy service API: Virtual Tenant Network (VTN) functions.

Prot.	Type	URI	Parameters	
REST	GET	*../mt/vtns/{tenant_id}* Retrieve list of Virtual Tenant Networks	Input	*tenant_id*
			Output	Success: List of *VTN_info* structures in response body
				Failure: Error code
REST	POST	*../mt/vtns/{tenant_id}* Creates a Virtual Tenant Network	Input	*tenant_id*
				Request body contains VTN_info
			Output	Success: Status Code of normal end Returns *tenant_id*
				Failure: Error code
REST	GET	*../mt/vtns/{tenant_id}/{vtn_id}* Retrieve information related to a VTN	Input	*tenant_id* *vtn_id*
			Output	Success: *VTN_info* of *tenant_id* structure in response body
				Failure: Error code
REST	POST	*../mt/vtns/{tenant_id}/{vtn_id}* Modify information of a VTN	Input	*tenant_id* *vtn_id*
				Request body contains *VTN_info*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	DELETE	../mt/vtns/{*tenant_id*}/{*vtn_id*}	Input	*tenant_id* *vtn_id*

		Remove a VTN	Output	Success: Status Code of normal end
				Failure: Error code

Table 22 shows the most important functions to add, modify and delete virtual switches (layer-2 forwarding elements) into a VTN.

Table 22: Multi-tenancy service API: Virtual L2 forwarding element functions (virtual switches).

Prot	Type	URI	Parameters	
REST	GET	*../mt/{tenant_id}/{vtn_id}/v_switches* Retrieve list of virtual switches that belong to *tenant_id*	Input	*tenant_id* *vtn_id*
			Output	Success: List of *v_switch_info* structures in response body
				Failure: Error code
REST	POST	*../mt/{tenant_id}/{vtn_id}/v_switches/* Creates a virtual switch	Input	*tenant_id*
				Request body contains *v_switch_info*
			Output	Success: Status Code of normal end Returns *v_switch_id*
				Failure: Error code
REST	GET	*../mt/{tenant_id}/{vtn_id}/v_switches/{v_switch_id}* Retrieve information related to a virtual switch	Input	*tenant_id* *vtn_id* *v_switch_id*
			Output	Success: *v_switch_info* of *v_switch_id* structure in response body
				Failure: Error code
REST	POST	*../mt/{tenant_id}/{vtn_id}/v_switches/{v_switch_id}* Modify information of virtual switch	Input	*tenant_id* *vtn_id* *v_switch_id*
				Request body contains *v_switch_info*
			Output	Success: Status Code of normal end
				Failure: Error code

REST	DELETE	../mt/{*tenant_id*}/{*vtn_id*}/v_switches/{*v_switch_id*} Remove a virtual switch	Input	*tenant_id* *vtn_id* *v_switch_id*
			Output	Success: Status Code of normal end
				Failure: Error code

Table 23 shows the most important functions to add, modify and delete virtual routers (layer-3 forwarding elements) into a VTN.

Table 23: Multi-tenancy service API: Virtual L3 forwarding element functions (virtual routers).

Prot	**Type**	**URI**	**Parameters**	
REST	GET	../mt/{*tenant_id*}/{*vtn_id*}/v_routers Retrieve list of virtual routers that belong to *tenant_id*	Input	*tenant_id* *vtn_id*
			Output	Success: List of *v_router_info* structures in response body
				Failure: Error code
REST	POST	../mt/{*tenant_id*}/{*vtn_id*}/v_routers/ Creates a virtual router	Input	*tenant_id* *vtn_id*
				Request body contains *v_router_info*
			Output	Success: Status Code of normal end Returns *v_router_id*
				Failure: Error code
REST	GET	../mt/{*tenant_id*}/{*vtn_id*}/v_routers/{*v_router_id*} Retrieve information related to a virtual router	Input	*tenant_id* *v_router_id*
			Output	Success: *v_router_info* of *v_router_id* structure in response body
				Failure: Error code
REST	POST	../mt/{*tenant_id*}/{*vtn_id*}/v_routers/{*v_router_id*} Modify information	Input	*tenant_id* *vtn_id* *v_router_id*

		of virtual router		Request body contains *v_router_info*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	DELETE	*../mt/{tenant_id}/{vtn_id}/v_routers/{v_router_id}* Remove a virtual router	Input	*tenant_id* *vtn_id* *v_router_id*
			Output	Success: Status Code of normal end
				Failure: Error code

Table 24 shows the most important functions manage the mapping between physical ports (e.g. of XFEs) and virtual forwarding elements.

Table 24: Multi-tenancy service API: port mapping functions.

Prot	**Type**	**URI**	**Parameters**	
REST	GET	*../mt/{tenant_id}/{vtn_id}/v_switches/{v_switch_id}/v_ifaces* *or* *../mt/{tenant_id}/{vtn_id}/v_routers/{v_router_id}/v_ifaces* Retrieve list of interface mapping that belong to *a virtual switch or a virtual router*	Input	*tenant_id* *vtn_id* *v_switch_id/v_router_id*
			Output	Success: List of *v_iface_info* structures in response body
				Failure: Error code
REST	POST	*../mt/{tenant_id}/{vtn_id}/v_switches/{v_switch_id}/* *or* *../mt/{tenant_id}/{vtn_id}/v_routers/{v_r*	Input	*tenant_id* *vtn_id* *v_switch_id/v_router_id*
				Request body contains *v_iface_info*
			Output	Success: Status Code of normal end Returns *v_iface_id*

		outer_id}/ Creates an interface mapping		Failure: Error code

REST	GET	*../mt/{tenant_id}/{vtn_id}/v_switches/{v_switch_id}/{v_iface_id}*
		or
		../mt/{tenant_id}/{vtn_id}/v_routers/{v_router_id}/{v_iface_id}
		Retrieve information related to a virtual interface mapping
			Input	*tenant_id*
				vtn_id
				v_switch_id/v_router_id
				v_iface_id

			Output	Success: *v_iface_info* structure in response body
				Failure: Error code

REST	POST	*../mt/{tenant_id}/{vtn_id}/v_switches/{v_switch_id}/{v_iface_id}*
		or
		../mt/{tenant_id}/{vtn_id}/v_routers/{v_router_id}/{v_iface_id}
		Modify information of virtual interface mapping
			Input	*tenant_id*
				vtn_id
				v_switch_id/ v_router_id
				v_iface_id

				Request body contains *v_iface_info*
			Output	Success: Status Code of normal end
				Failure: Error code

REST	DELETE	*../mt/{tenant_id}/{vtn_id}/v_switches/{v_switch_id}/{v_iface_id}*
		or
		../mt/{tenant_id}/{vtn_id}/v_routers/{v_router_id}/{v_iface_id}
			Input	*tenant_id*
				vtn_id
				v_switch_id/ v_router_id
				v_iface_id

			Output	Success: Status Code of normal end
				Failure: Error code

		Remove a virtual interface mapping		

Table 25 shows the most important functions to manage virtual links between virtual forwarding elements.

Table 25: Multi-tenancy service API: virtual link functions.

Prot	**Type**	**URI**	**Parameters**	
REST	GET	*../mt/{tenant_id}/{vtn_id}/v_links* Retrieve list of virtual links	Input	*tenant_id* *vtn_id*
			Output	Success: List of *v_link_info* structures in response body
				Failure: Error code
REST	POST	*../mt/{tenant_id}/{vtn_id}/v_links/* Creates an virtual link	Input	*tenant_id* *vtn_id*
				Request body contains *v_link_info*
			Output	Success: Status Code of normal end Returns *v_link_id*
				Failure: Error code
REST	GET	*../mt/{tenant_id}/{vtn_id}/v_links/{v_link_id}* Retrieve information of a virtual link	Input	*tenant_id* *vtn_id* *v_link_id*
			Output	Success: *v_link_info* structure in response body
				Failure: Error code
REST	POST	*../mt/{tenant_id}/{vtn_id}/v_links/{v_link_id}* Modify information of virtual link	Input	*tenant_id* *vtn_id* *v_link_id*
				Request body contains *v_link_info*
			Output	Success: Status Code of normal end
				Failure: Error code
REST	DELE	*../mt/{tenant_id}/{vtn*	Input	*tenant_id*

	TE	*_id}/v_links/{v_link_i d}* Remove a virtual link		*vtn_id* *v_link_id*
			Output	Success: Status Code of normal end
				Failure: Error code

8.10.2. Information Model

The Multi-tenancy information model described in

Figure 31 supports virtualization of networking resources through a mapping between virtual and physical entities. The main parameters are described in Table 26.

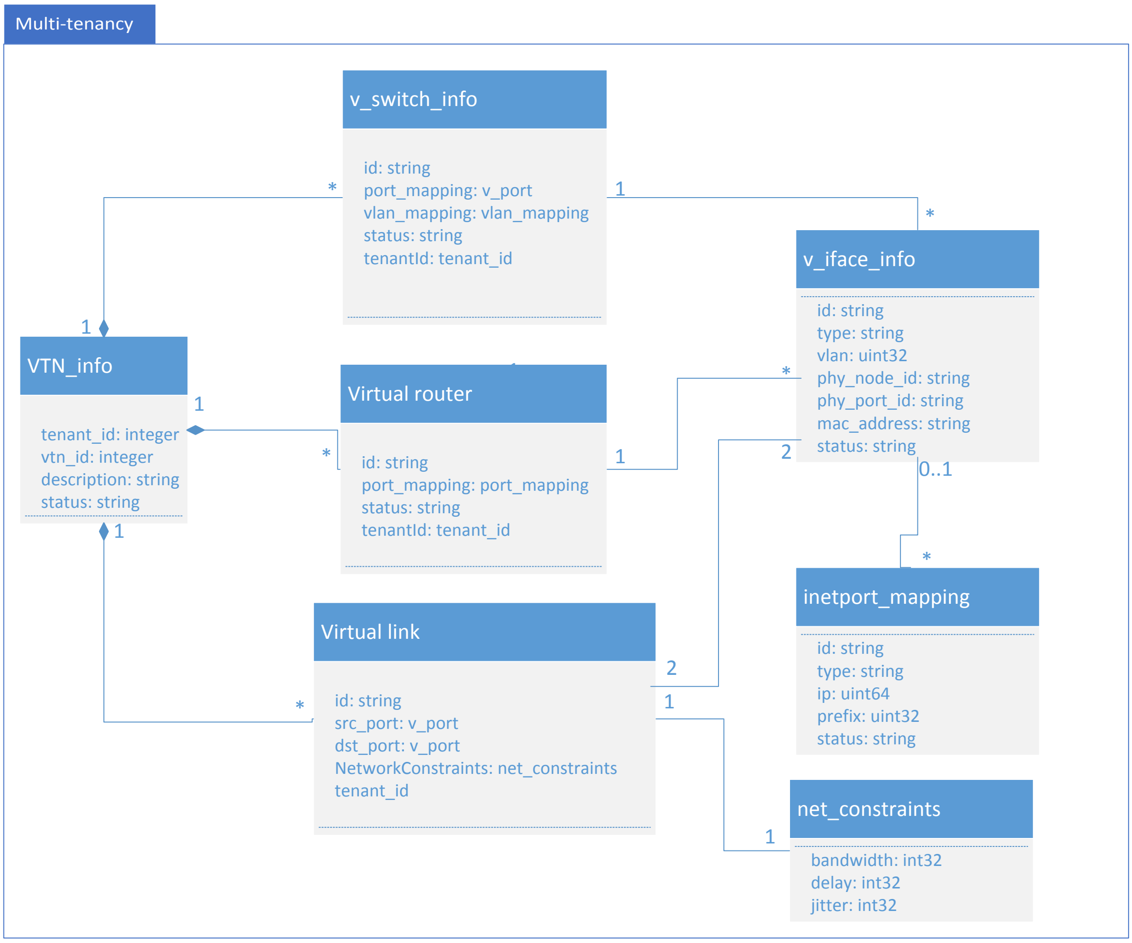


Figure 31: Multi-tenancy service information model

Table 26: Multi-tenancy service information model

Parameter	Type	Description
tenant_id	Integer	Unique identifier of a virtual tenant
vtn_id	Integer	Unique identifier of a virtual network
v_switch_id	Integer	Unique identifier of a virtual switch
v_router_id	Integer	Unique identifier of a virtual router
v_link_id	Integer	Unique identifier of a virtual link
v_switch_info	Object	Object describing a virtual switch (e.g., status)
v_router_info	Object	Object describing a virtual router (e.g., status)
v_iface_info	Object	Object describing a virtual interface mapping (e.g. status, mapping to physical interfaces)
v_link_info	Object	Object describing a virtual link mapping (e.g. status, mapping to physical links/paths)

8.10.3. Workflow

Figure 32 illustrates an example of usage of the multitenancy service by MTA or VIMaP, who are the consumers of the service in this case. This represents a very simple example of the creation of a virtual tenant network (VTN) through a process that comprises the following steps:

1. The consumer requests the creation of a new network tenant. The multitenancy allocates space to store information on the new tenant and generates a new identifier if successful.
2. The consumer iteratively requests the creation of virtual switches. The multitenancy service generates identifiers for each of the new entities.
3. The consumer iteratively requests the creation of virtual routers. The multitenancy service generates identifiers for each of the new entities.
4. The consumer iteratively requests a mapping between a physical port and a virtual port that belongs to a virtual switch or virtual router. In the latter case, an IP configuration is also required.
5. The consumer iteratively requests the creation of virtual links by routing pairs of virtual ports. To do so, the multitenancy service requests the SDN controller a route between two mapped physical ports given a set of network constraints (*net_constraints*).

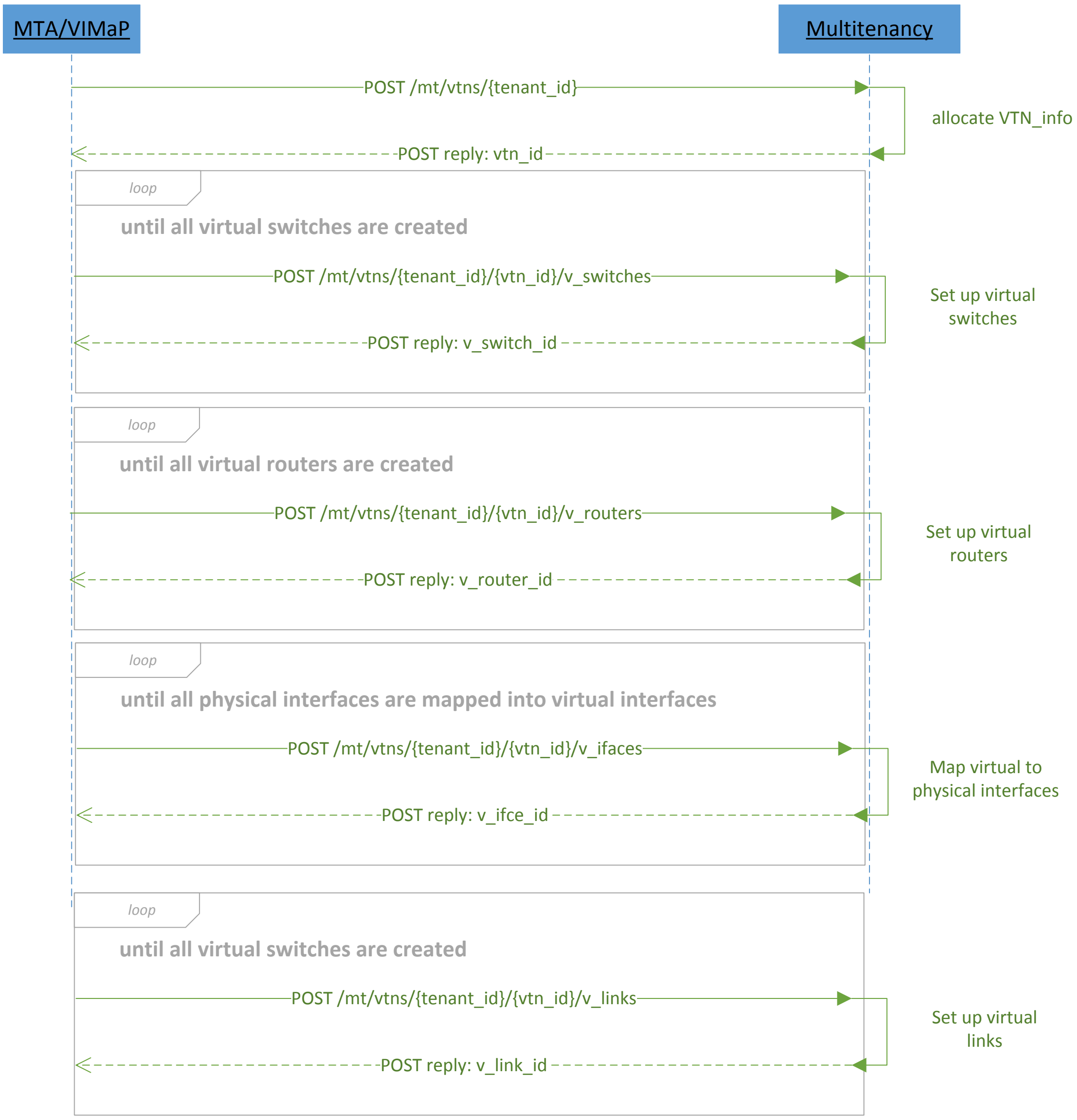

MTA/VIMaP
Multitenancy
POST /mt/vtns/{tenant_id}
allocate VTN_info
POST reply: vtn_id
loop
until all virtual switches are created
POST /mt/vtns/{tenant_id}/{vtn_id}/v_switches
Set up virtual switches
POST reply: v_switch_id
loop
until all virtual routers are created
POST /mt/vtns/{tenant_id}/{vtn_id}/v_routers
Set up virtual routers
POST reply: v_router_id
loop
until all physical interfaces are mapped into virtual interfaces
POST /mt/vtns/{tenant_id}/{vtn_id}/v_ifaces
Map virtual to physical interfaces
POST reply: v_ifce_id
loop
until all virtual switches are created
POST /mt/vtns/{tenant_id}/{vtn_id}/v_links
Set up virtual links
POST reply: v_link_id

Figure 32: Multi-tenancy service workflow example.

9. Validation and Evaluation

9.1. Methodology

This section describes the global methodology adopted for the validation of the data plane and control-plane components under development in WP3. The scope of the validation activities within WP3 is to assure the proper behaviour of the functions and algorithms that are integral part of the components constituting the whole XCI control-plane infrastructure and operating the heterogeneous resources in the data plane. In the same way, the different kinds of technology adopted to perform forwarding in the data plane will be tested to guarantee the proper functionality of the 5G-Crosshaul network substrate infrastructure. The definition of proper testing procedures for each XFE/XCI prototype and algorithm is an important base in order to facilitate the whole system integration and verification process that will take place in WP5. Indeed, the preliminary structured unit tests of each function will reduce consistently the number of integration's issues during the process of verification and evaluation of the whole 5G-Crosshaul architecture. In detail, the validation methodology adopted and formalized below is based on the definition of a set of functional objectives that are the final goals of each test performed on specific hardware/software prototypes or algorithm. Each test is structured with the aim to verify a particular functionality or expected behavior of the System Under Test (SUT), in a simplified environment which emulates the interaction with other architectural components or the data consumed by the component/algorithm itself. This approach allows identifying possible inappropriate behaviours that depend on the single component/algorithm in an isolated and controlled environment, so that potential bugs can be efficiently fixed on the single modules before the final integration phase takes place. The following test card template, reported in Table 27, represents a formalization of the validation procedure for software prototype components. Each test is divided in sequential steps with expected results. During the test execution, the status of each step will be completed with the actual result (i.e. Passed/Failed). The final objectives are achieved through the validation of each single step in the workflow, so the execution of the whole test card will be considered successful only if all the single steps will be executed with a positive result. Furthermore, the test procedure specifies the environment where executing the test, defining a reference topology made up by interfaces, data plane resources and external components, eventually emulated, in order to check the whole software functionalities and interactions.

Table 27: Test card template

Test Card #		Execution Status	
Test Name			

Objectives	
Related Use Cases	
Responsible	
Related Test Cards	
Additional Comments	

SUT and topology	
SUT	Components under test
Test environment topology	Data plane topology; control plane components
External components	External components required to perform the test

Test description		
Step #	Step description and expected results	Status
1.	**Description:** **Expected Results:**	
2.	**Description:** **Expected Results:** **Comments:**	

9.2. Data Plane Validation Plan (Nokia)

In the data plane, three different types of elements will be validated: circuit switches (XCSEs), packet switches (XPFEs), and the split ratio protocol stack.

Two types of XCSEs have been evaluated, the silicon photonics Reconfigurable Optical Add and Drop Multiplexer (ROADM) and an XCSE using the deterministic delay protocol described in [1], both developed by TEI. For each of the XCSEs one dedicated setup has been defined, on which the corresponding measurements have been performed. The test setup for the silicon photonics ROADM is shown in Figure 33(a).

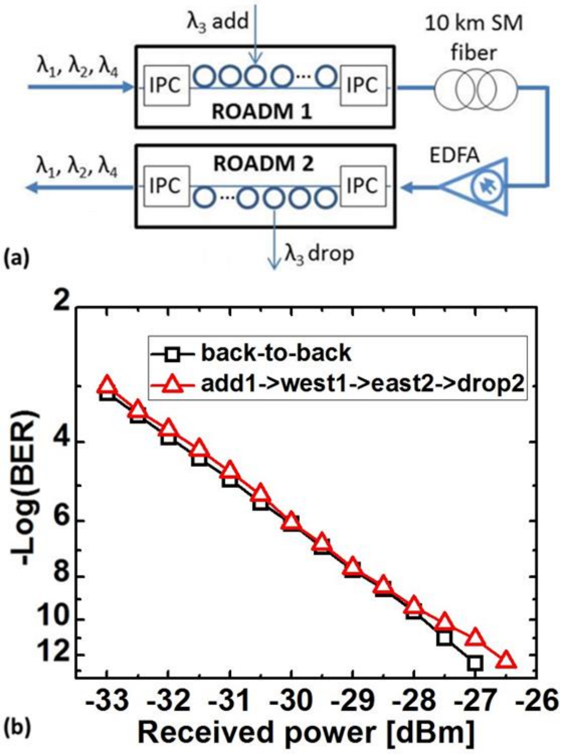


Figure 33: (a) characterization set-up. (b) BER results

We used two packaged ROADMs: ROADM-1 output bus was connected to the ROADM-2 input bus through a 10km long single-mode fibre (SMF) spool. The adjacent channels λ_1, λ_2, λ_4 were sent on the ROADM-1 input bus coming from another DWDM system. The signal λ_3 was sent to the dedicated add port of the ROADM-1 to multiplex it with the other channels on the bus. The four multiplexed signals were coupled out of the ROADM-1 and reached the ROADM-2 after propagating through the 10km fibre spool. Here λ_3 was dropped and sent to another DWDM system. Bit Error Rate (BER) measurements versus received power were performed. Figure 33(b) shows the measured BER performance of the link. We successfully obtained error free operations with a power penalty lower than 0.7dB at BER=10^{-12}.

Two XPFEs will be validated, the nodes in the Interdigital Edgelink mmWave mesh technology and the software switches provided by Nokia. In both cases the correct forwarding of frames and the correct encapsulation/decapsulation of frames will be tested. Additionally, latency/jitter measurements will be performed for the software switch. These measurements take into account different types of fronthaul (FH) and backhaul (BH) like traffic as well as the impact on flow entry modifications. The test setups consist typically of several XPFEs connected to traffic sources and sinks and being controlled by an SDN controller. As an example see Figure 34.

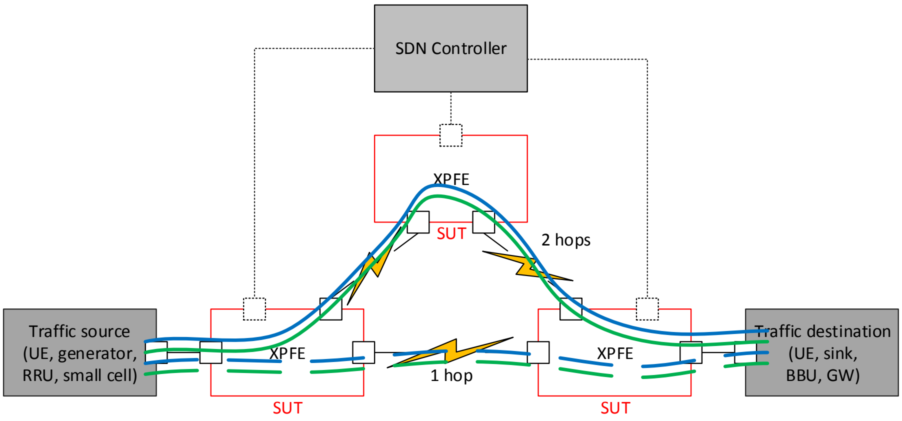

SDN Controller
XPFE
SUT
2 hops
Traffic source (UE, generator, RRU, small cell)
XPFE
SUT
1 hop
XPFE
SUT
Traffic destination (UE, sink, BBU, GW)

Figure 34 : XPFE test setup

Test Card #	*Test Name*
IDCC_XPFE_01	Forwarding of non-XCF frames
IDCC_XPFE_02	Multiplexing of non-XCF frames
IDCC_XPFE_03	Encapsulation and decapsulation of XCF frames
IDCC_XPFE_04	Validate correct forwarding of XCF frames over mmWave mesh network
IDCC_XPFE_05	Multiplexing of XCF frames
NOKIA_XPFE_01	Forwarding of XCF frames
NOKIA_XPFE_02	Encapsulation and decapsulation of non-XCF frames
NOKIA_XPFE_03	Exchange of information relevant for the Topology and Inventory Service of the XCI
NOKIA_XPFE_04	Validation of simulations against the switch behaviour
NOKIA_XPFE_05	Baselining of pure XCF forwarding and encapsulation/decapsulation functionality
NOKIA_XPFE_06	Evaluation of the C-plane load on latency and jitter
NOKIA_XPFE_07	Validate latency and jitter for heterogeneous traffic flows
NOKIA_XPFE_08	Validate latency and jitter requirements for pure CPRI traffic

The split radio stack has been evaluated both with a direct Ethernet connection among the Remote Radio Head (RRH) and Baseband Unit (BBU) as well as using mmWave technology developed by Interdigital. Figure 35 shows the setup using mmWave technology among radio remote unit (RRU) and BBU.

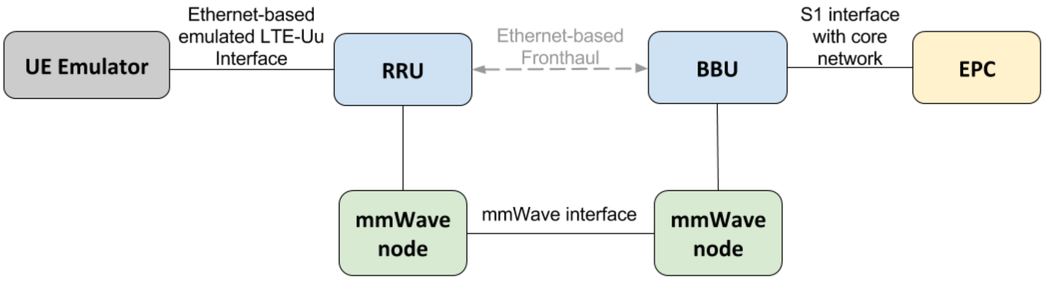


Figure 35: Setup for split radio protocol stack

The validation procedures cover setup of the S1-connection, validation of Cell-broadcast, user equipment (UE) attachment and traffic exchange.

Test Card #	*Test Name*
CND_FHBH_1	Validate S1 Setup of C-RAN eNodeB over FH/BH
CND_FHBH_2	Validate Cell Broadcast
CND_FHBH_3	UE attachment over FH and BH, downlink and uplink user plane traffic and detachment
CND_FHBH_4	UE attachment over FH and BH, downlink and uplink user plane traffic and detachment with fronthaul over mmWave
CND_FHBH_5	UE attachment over FH and BH, downlink and uplink user plane traffic and detachment using SDR board
CND_FHBH_6	UE attachment over FH and BH, downlink and uplink user plane traffic and detachment using SDR board and fronthaul over mmWave

9.3. Control Plane Validation Plan

This section presents the XCI prototype validation plan taking into consideration the different types of components that coexist within the 5G-Crosshaul Control Infrastructure, in particular dividing the argument in three subsections dealing respectively with the XCI MANO components, the SDN controllers and the algorithms running within the XCI itself. More in detail, each component/algorithm is validated on the base of the management and orchestration target resources, focusing the test procedure on the software component related use case.

9.3.1. Validation of XCI MANO Components

9.3.1.1. XCI MANO for CDN

The goal of the test with regard to the functional validation is to prove the feasibility of the virtual content delivery network (vCDN) implementation concept, as defined in [2], using the functions developed within the XCI MANO components. To that end, the main aspects that the test aims at are the following:

- Processing of a CDN service instantiation request. The NFV Orchestrator through its northbound must understand the network service request, manage the data provided in it (network service template) and interact with the VNF Manager and the VIM in order to deploy the vCDN infrastructure required.
- Starting up the VMs through the VIM and instantiate and configure the vCDN nodes through the VNF Manager.
- Enabling the proper end-to-end connectivity between the vCDN nodes through the VIM functions.
- Monitoring the status of the vCDN node containers (VMs) and specific information related to the vCDN node performance through VNF probes configured by the VNF Manager. These data have to be available at the application level.

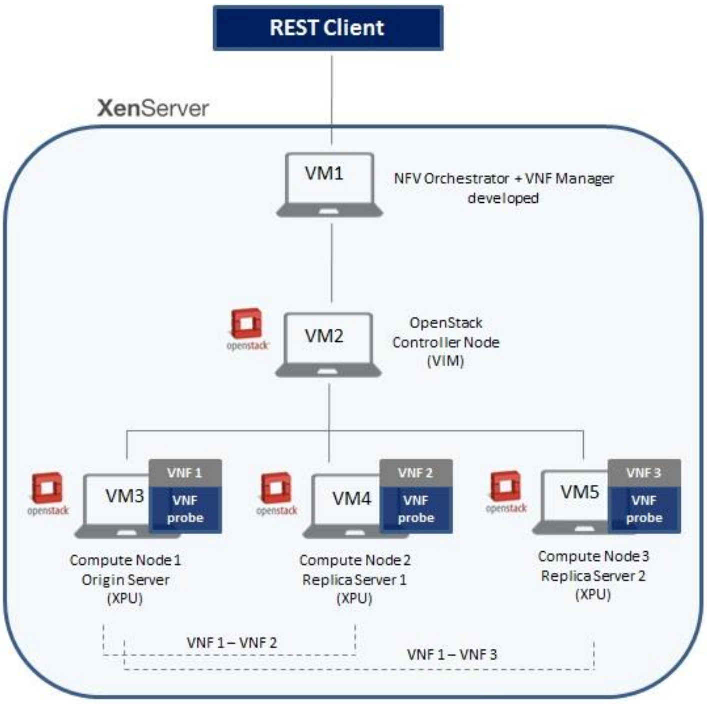

REST Client
XenServer
VM1
NFV Orchestrator + VNF Manager developed
openstack
VM2
OpenStack Controller Node (VIM)
openstack
VM3
VNF 1
VNF probe
Compute Node 1 Origin Server (XPU)
openstack
VM4
VNF 2
VNF probe
Compute Node 2 Replica Server 1 (XPU)
openstack
VM5
VNF 3
VNF probe
Compute Node 3 Replica Server 2 (XPU)
VNF 1 – VNF 2
VNF 1 – VNF 3

Figure 36: XCI MANO for CDN. Validation Environment

The validation environment where all components under test will be developed is located in a cloud environment. The testbed will be composed of a NFV Orchestrator and a VNF Manager developed based on open source software. There will also be a VIM based on OpenStack Mitaka[4] version, which will manage an infrastructure composed of a controller node and three compute nodes where it will be instantiated the vCDN nodes. The connection between the vCDN nodes will be configured through the OpenStack module Neutron functions. The vCDN nodes are based on Wowza

[4] https://www.openstack.org/software/mitaka/

Streaming Engine[5] media server software and for the simulation of the network service request will be used a REST client interacting with the NFV Orchestrator northbound.

The following test cards have been defined in order to describe the validation procedures that will be carried out for testing the components.

Table 28: XCI MANO for CDN. Testing Procedures

Test Card #	*Test Name*
ATOS_vCDN_MANO_01	Starting up the VMs and running the vCDN node VNFs
ATOS_vCDN_MANO_02	Collection and storage of VM and VNF monitoring data
ATOS_vCDN_MANO_03	Orchestration and management of a vCDN infrastructure

9.3.1.2. XCI MANO for vEPC and energy management

The validation procedures of the XCI MANO components are focused on the functional testing of the following main features:

- Monitoring of power consumption from the XPUs, performed at the VIM level based on data collected through its South Bound Interface from the compute nodes of the physical infrastructure. These data must be made available through a suitable monitoring interface exposed by the VIM component at its north bound.
- The capability to regulate the status (switch-on/off) of the compute nodes based on their working load, making use of suitable NBI at the VIM level.
- The capability to instantiate vEPC instances taking into account energy constraints, with the possibility to select and specify the optimum set of compute nodes from a power consumption perspective.

[5] https://www.wowza.com/products/streaming-engine

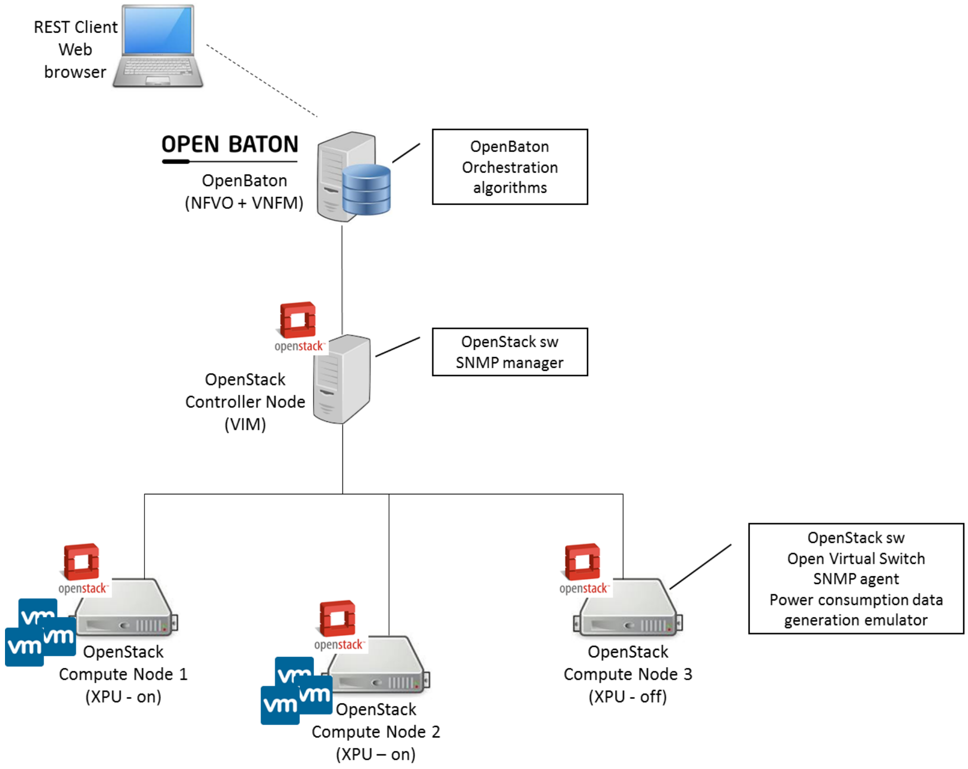


Figure 37: Validation environment

The validation environment (see Figure 37) requires the deployment of the following components of the System Under Test (SUT):

- NFV-O and VNFM(s) based on the extended OpenBaton[6] software, installed in a server;
- VIM, based on OpenStack Mitaka version, deployed with a controller node and three compute nodes. The required OpenStack modules are Nova, Neutron, Ceilometer, Glance, Keystone, and Horizon. The generation of energy monitoring data is emulated by software, based on models that take into account the number of running VMs.

The interaction with the client entities at the northbound interface (i.e. the EMMA application) is emulated through a REST client like curl[7] or Postman[8].

The functional validation will be performed through the execution of testing procedures defined in the following testcards.

Table 29: Testing procedures

Test Card #	*Test Name*
NXW_EMMA_MANO_01	Monitoring of XPU energy consumption

[6] http://openbaton.github.io/
[7] https://curl.haxx.se/
[8] https://www.getpostman.com/

NXW_EMMA_MANO_02	Regulation of power-on/power-off status of XPUs
NXW_EMMA_MANO_03	Instantiation of energy-efficient vEPC instances

9.3.1.3. Openstack-based VIMaP

Macroscopically, the goal is to validate the VIMaP component implementation and its ability to instantiate interconnected Virtual Machines, as previously defined in Section 8.5. Figure 38 provides a schematic diagram of the involved components with VIMaP development and testing.

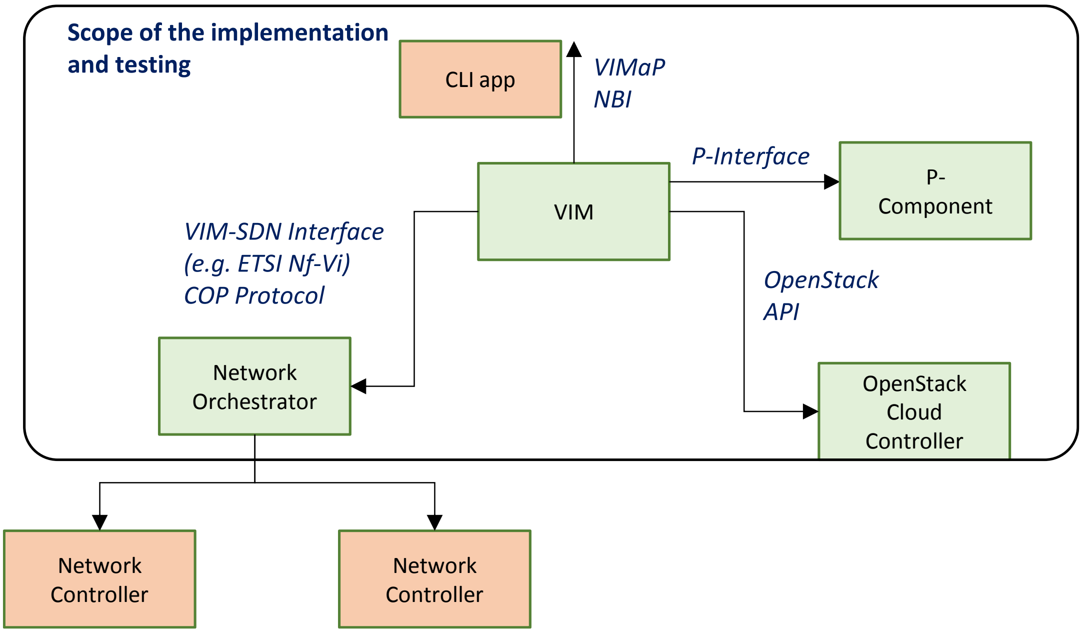

Scope of the implementation and testing
CLI app
VIMaP NBI
P-Interface
P-Component
VIM
VIM-SDN Interface (e.g. ETSI Nf-Vi) COP Protocol
OpenStack API
Network Orchestrator
OpenStack Cloud Controller
Network Controller
Network Controller

Figure 38: Diagram of the involved components with VIMaP development and testing

The main objectives of the planned tests to be carried out are the following:

- *Functional validation and experimental assessment of the VIMaP NBI interface*, that is, enabling a test application to request VIMaP services. For the testing, Command Line Interface (CLI) clients will be used. The main service will be the instantiation of interconnected Virtual Machines, following a client specified constrained graph. This is the main macroscopic objective and includes, as sub-objectives, the following ones.
- *Functional validation of the interaction with the cloud controller (OpenStack)* This objective aims at validating the VIMaP as a consumer of OpenStack services and its use to instantiate VMs in constrained locations, for the considered operating system images. For the testing, generic Linux images (medium size) will be considered.
- *Functional validation of the interaction with the SDN controller* Likewise, this objective targets the consumption of the SDN controller(s) NBIs for the provisioning of connectivity services in one (or multiple) domains. By design, a hierarchical arrangement of controllers is considered. In this sense, the VIMaP component only needs to interact with one SDN controller, which, depending on the

scenario, is either a parent controller – also known as network orchestration and implemented either following the Application-Based Network Operation (ABNO) architecture – or, in simpler deployments, a single SDN controller. In either case, it is expected that the SDN controllers implement a YANG-based API such as Control Orchestration Protocol (COP)[9], the Transport API186 defined by the ONF Transport Group [22] , or IETF TEAS topology and tunnel models [23] .

- *Functional validation of the P-interface* The P-interface will be tested to see if the P component (provided as a separate component) is able to execute the placement function independently.

These main objectives involve the following features and functionality (only considering the VIMaP component)

- The ability to query, from the OpenStack cloud controller, the status of the compute nodes, and their capabilities. This includes querying attributes of the system and retrieving the number and type of instances, etc.
- The ability to upload images into the cloud controller, which will be launched when a VM instance is allocated.
- The ability to launch VMs with a set of associated resources and to specify the image to be launched for that Virtual Machine. Likewise, the ability to query the status of the images and to terminate an image when it is no longer needed.
- The ability to query, from the SDN controller, the topology of the transport network, which needs to be augmented with the location and capabilities of the XPU nodes (OpenStack configured compute nodes) to have an enhanced Traffic Engineering Database (TED).
- The ability to query, from the SDN controller, the status of existing and/or configured flows, and their properties and attributes.
- The ability to provision, invoking the correct API from the SDN controller, one or multiple flows that will constitute the inter-VM connectivity
- Finally, the ability to instantiate and terminate interconnected overlay VMs as defined by the VIMaP NBI.

For the time being, all the VMs, which will be assumed to belong to the same IP subnet, will be instantiated within the administrative project / tenant and connectivity will be provided at layer 2 (MAC) level.

The validation environment is the CTTC testbed, which is composed of two existing internal testbeds, namely, the ADRENALINE and EXTREME testbeds [18]. It is worth noting, though, that specific integration, and regression tests may be carried out with only a subset of the components and function entities. The CTTC testbed includes several data plane technologies, namely, an mmWave/WiFi mesh network domain, an access and aggregation packet switched network domain and an optical circuit switched network domain, as detailed in D5.1 [18].

[9] https://github.com/ict-strauss/COP

From the point of view of the control and VIMaP aspect, software applications and components (VIMaP, ABNO, SDN controllers, Active Stateful Path Computation Element (PCE) etc.) will run on dedicated commercial off-the-shelf (COTS) hardware (Intel CPUs with GNU/Linux Operating System) and with diverse hardware configurations depending on hardware requirements for each application or component. An OpenStack deployment will include a controller node, a network node where applicable and, depending on the scenario, multiple compute nodes deployed in diverse locations.

For testing specific mock-up components or simplified settings will be used. For example, we are considering using available OpenFlow SDN controllers to be run in virtualized environments (e.g. Mininet[10]) for simplified testing.

A detailed description of the validation procedures and its associated test cards can be found in Appendix (Section 11.3). As a general description, the validation procedures involve, mainly,

- the implementation and use of interfaces between the VIMaP and OpenStack keystone, nova, glance and neutron.
- the implementation and use of interfaces between the VIMaP and the SDN controller (either a standalone controller or an ABNO orchestrator).

Test Card #	*Test Name*
CTTC-VIMaP-T001	Topology detection
CTTC-VIMaP-T002	XPU status and capability discovery
CTTC-VIMaP-T003	Instantiation of Virtual Machines in remote locations
CTTC-VIMaP-T004	Flow configuration across Single- and Multi-domain networks
CTTC-VIMaP-T005	Functional assessment of the VIMaP NBI
CTTC-VIMaP-T006	External placement computation of VMs and flows (P component)

9.3.2. Validation of XCI SDN Controllers

9.3.2.1. SDN controller for XPFE and energy management

The validation procedures of the XPFE SDN controller are focused on the functional testing of the following main features:

- The OpenFlow-based interaction between XPFEs and SDN controller to collect information about the capabilities of the network devices, to build their topology, to monitor their status and statistics and to configure flow entries (i.e. support of core XCI controller services for XPFEs).
- For XPFEs supporting power consumption monitoring, the collection of power-related parameters via SNMP protocol. For XPFEs not supporting explicit power consumption monitoring, the capability to compute power consumption parameters following a model based on traffic load.

[10] http://mininet.org/

- Aggregation of power consumption monitoring data to measure global power consumption at the physical infrastructure level and power consumption per network path or per tenant.
- The capability to regulate the status (switch-on/off) of the network nodes based on their working load, making use of suitable SBIs (e.g. SNMP or NETCONF).
- The capability to compute, instantiate and terminate tenant-specific network paths, triggered via NBI interface requests. Network path setup is managed through the suitable configuration of flow entries across the flow tables in the XPFE's pipeline via OpenFlow messages. The resulting configuration allows encapsulating traffic at the network edge, making use of the MAC-in-MAC feature as defined in the XCF format, in order to manage proper isolation between tenants.

The validation environment requires the deployment of the following components of the System Under Test (SUT):

- XCI SDN controller for the operation of XPFEs based on Lagopus software switches (see Section 9.2). The SDN controller is based on OpenDaylight Beryllium and includes the software bundles for OpenFlow plugin, core services like inventory, topology, statistics and flow management, and web GUI (i.e. the OpenDaylight DLUX service).
- Depending on the specific test case, further OpenDaylight-based service or external SDN applications, e.g. SNMP plugin, analytics service for power-consumption monitoring, path computation and provisioning service, network nodes activator.

Concerning the data-plane, Mininet will be used to emulate network topologies composed of several switches and hosts. Lagopus switches with XCF support will be used to emulate the data plane in tests related to OpenFlow-based SBI functionalities, while an SNMP Agent simulator[11] will be used to verify the collection of monitoring data via SNMP protocol. The interaction with the client entities at the northbound interface (i.e. the EMMA application or the OpenStack-based VIM for virtual network requests) is emulated through a REST client like curl or Postman.

[11] http://snmpsim.sourceforge.net/

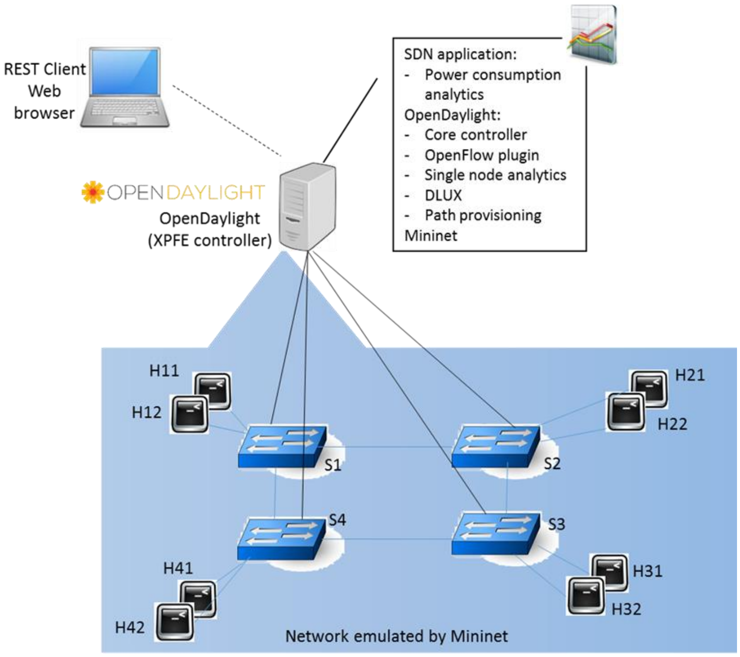


Figure 39: Validation environment with Mininet emulated data-plane

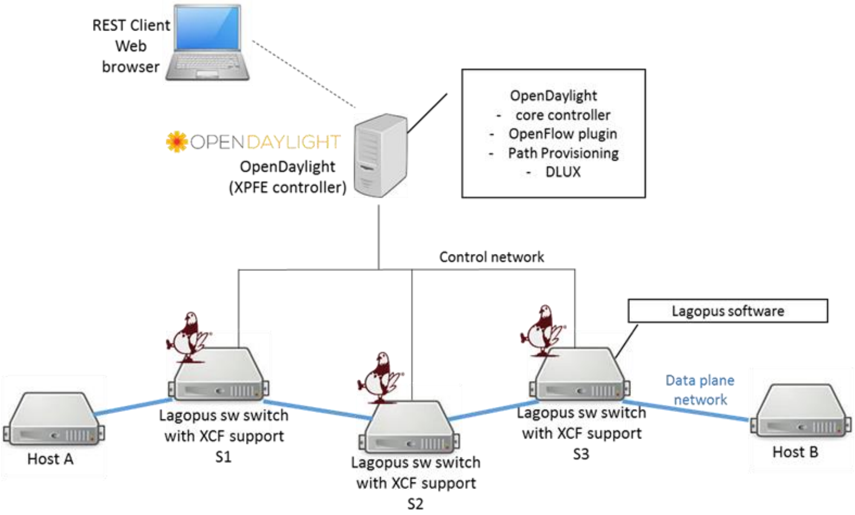


Figure 40: Validation environment with Lagopus emulated data-plane

The following test cards define the testing procedure.

Table 30: Validation procedures

Test Card #	*Test Name*
NXW_XCI_01	Core XCI controller services in XPFE's networks
NXW_XCI_02	Monitoring of XPFE power consumption via SNMP
NXW_XCI_03	Monitoring of XPFE power consumption via analytics
NXW_XCI_04	Analytics for computation of power consumption in virtual infrastructures
NXW_XCI_05	Modification of XPFE operational status
NXW_XCI_06	Setup and termination of energy-efficient paths on XPFE networks

9.3.2.2. SDN controller for mmWave and WiFi mesh technology

The validation objectives can be decomposed into three main functional goals:

- *Wireless link backhaul selection*: first goal is the intelligent interface selection out of those available in the wireless backhaul node. The SDN controller must recognize, at this stage, a basic set of capabilities of every wireless backhaul interface available in a wireless node. This also implies reaction for the detection of link failures and the selection of the appropriate wireless interface for forwarding purposes according to a wireless technology based criteria.
- *Management of wireless backhaul interfaces*: second goal is the remote management of the wireless backhaul interfaces. Instead of manually configuring the parameters associated to a wireless backhaul hardware interface, the SDN controller would embed a software interface to allow the configuration and reconfiguration of wireless interface parameters.
- *Wireless backhaul path selection*: third goal is to enable the proper end-to-end selection of paths in the wireless mesh network environment using different technologies as needed based on certain routing metrics.

A schematic illustration of the validation environment can be found in Figure 41. The validation environment requires the deployment of the following components, which are detailed in D5.1 [18]:

- From the control plane perspective, the SDN controller for mmwave/Wifi meshes.
- From the data plane perspective, forwarding nodes are equipped with IEEE 802.11ac cards and IEEE 802.11ad cards operating in the mmWave band.

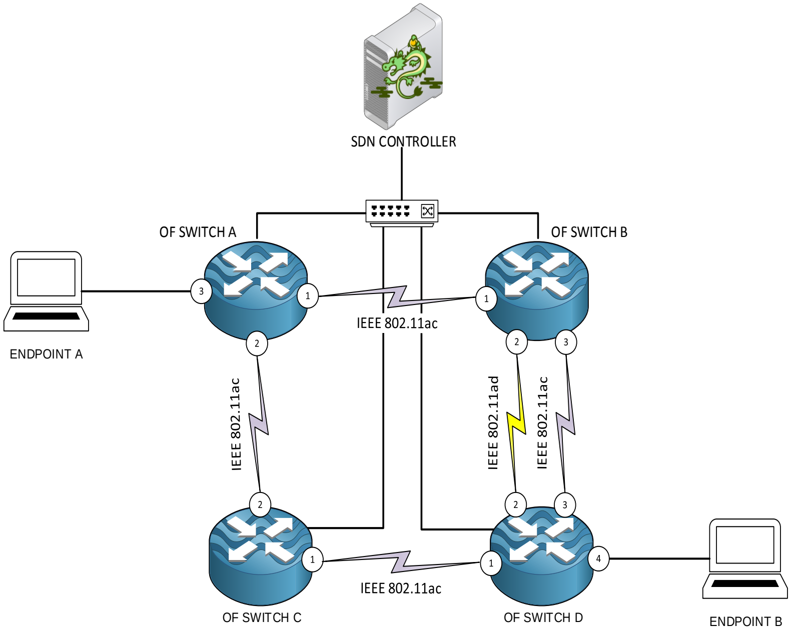


Figure 41: Validation environment for the SDN controller for mmWave/WiFi meshes

A detailed description of the validation procedure and its associated test cards are defined in Section 12.2.

Test Card #	*Test Name*
CTTC-mmWave-F01-T001	Topology detection
CTTC-mmWave-F01-T002	Interface state monitoring and reconfiguration
CTTC-mmWave-F01-T003	Interface technology detection
CTTC-mmWave-F01-T004	Route establishment according to path priority
CTTC-mmWave-F01-T005	Link failure and restoration
CTTC-mmWave-F01-T006	Failure recovery and route verification
CTTC-mmWave-F02-T001	Interface monitoring
CTTC-mmWave-F02-T002	Interface configuration
CTTC-mmWave-F03-T001	End-to-end route validation
CTTC-mmWave-F03-T002	End-to-end route validation with link priority based on technology
CTTC-mmWave-F03-T003	End-to-end route validation with link priority based on link characteristics
CTTC-mmWave-F03-T004	End-to-end route validation with link priority based link characteristics and link occupation
CTTC-mmWave-F03-T005	End-to-end route validation and link failure recovery

9.3.2.3. SDN controller for mmWave mesh technology

Three validation objectives are identified for the mmWave mesh technology, namely:

- *mmWave mesh bootstrap*: the first validation objective consists of assessing the capability of building the mmWave mesh network itself. The EdgeLink nodes will need to discover their neighbours and setup a connection with the mesh Gateway first in order to connect to the mesh controller.
- *mmWave mesh data plane configuration*: the second validation objective consists of assessing the correct mmWave mesh network configuration by the network controller related to the forwarding. the mesh controller needs to compute and configure a primary and a secondary path for forwarding the traffic within the network. The secondary path is the fall back path in case of the first path fails.
- *mmWave mesh self-healing mechanisms*: The third validation objective consists of assessing the capability of the network controller to react to events that happen in the mmWave mesh. For instance, once the primary path failed and the traffic is re-routed on the secondary path, the network controller needs to re-compute and reconfigure the primary and secondary paths in the mesh to restore the optimal configuration.

The validation environment for the SDN mmWave mesh controller for EdgeLink platform is detailed in [18] and summarized in Figure 42.

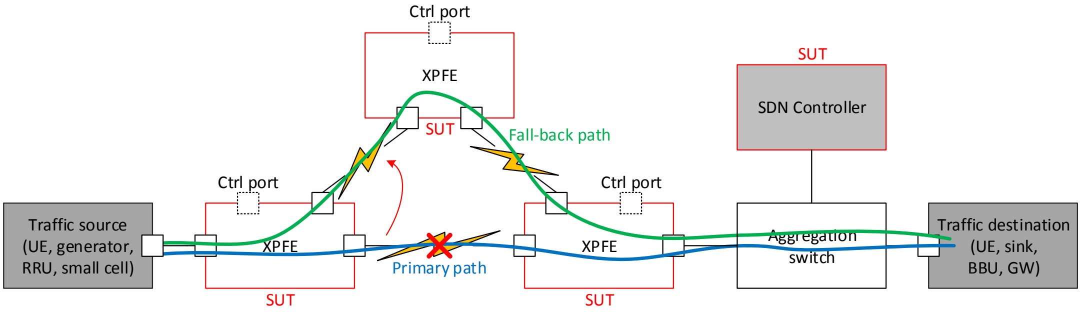


Figure 42: Validation environment for mmWave mesh controller

The validation envisages three phases: the first phase will be devoted to the assessment of the mesh stability in terms of node association; the second phase will consider the computation and configuration of primary and fall back paths in the mesh network by the network controller; finally, the third validation phase will assess the impact of the control plane on the ongoing traffic. A detailed description of the validation procedures and the associated test cards are defined in Section 12.3 and summarized in the following.

Test Card #	*Test Name*
IDCC_XCI_01	mmWave mesh stability
IDCC_XCI_02	Computation and configuration of paths within mmWave mesh network
IDCC_XCI_03	Connectivity recovery upon mmWave link failure
IDCC_XCI_04	Control plane impact on mmWave mesh network

9.3.2.4. ABNO-based hierarchical SDN controller

The main objective of this testing and validation activity is to demonstrate that the ABNO-based hierarchical SDN controller component can provide hierarchical control of underlying SDN controllers.

In the proposed implementation, recursive hierarchy is obtained through the extensive usage of resource abstraction and a YANG-based API referred to as COP, which allows the recursivity of basic SDN controller services, namely: Topology, Connectivity, Path Computation and Notification. COP was first introduced by the STRAUSS project[12], but in this activity follow up on its extensions will be proposed. Figure 43 provides a schematic diagram of the involved components with ABNO development and testing.

[12] http://www.ict-strauss.eu/

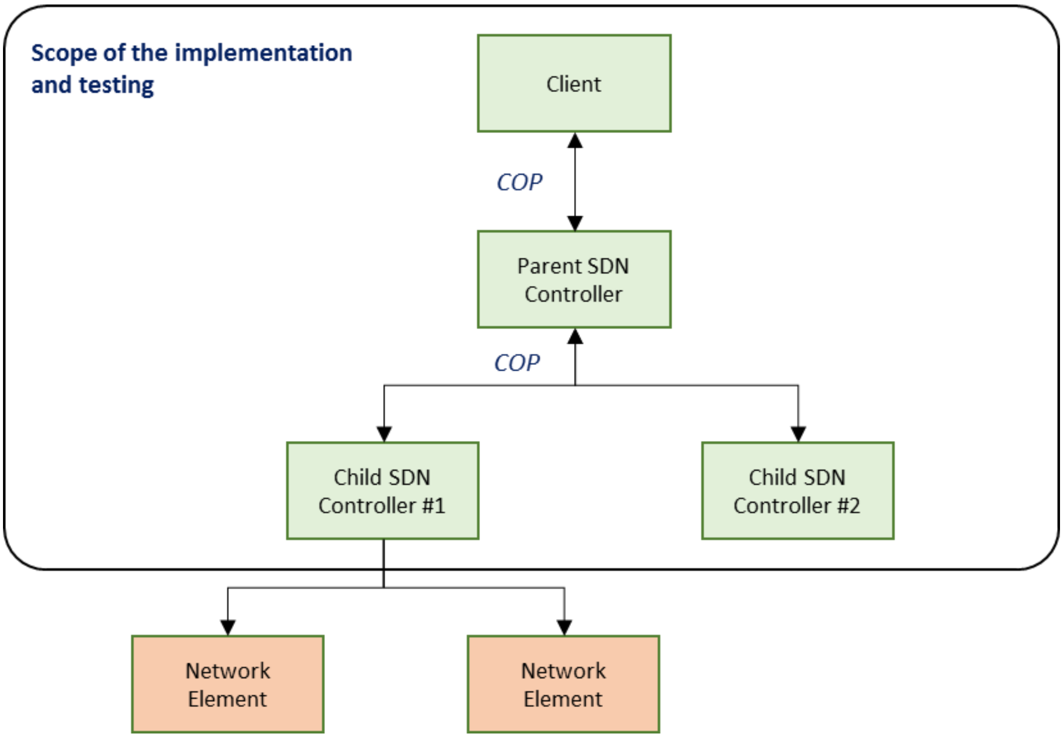


Figure 43: Diagram of the involved components with ABNO development and testing

In what follows, we describe the objectives of the planned tests:

- *Functional validation and experimental assessment of the ABNO NBI interface*, that is, enabling a client application (e.g., such as VIMaP) to request SDN services. The main service will be the instantiation of connectivity services. This is the main macroscopic objective that includes topology recovery, path computation and notification.
- *Functional validation of the interaction with the child SDN controller(s)* Likewise, this objective targets the consumption of the SDN controller(s) NBIs based on COP protocol for the provisioning of connectivity services in one (or multiple) domains. By design, a hierarchical arrangement of controllers is considered. In this sense, the parent SDN controller needs to interact with underlying SDN controllers, which are expected to implement the COP protocol as defined.

These main objectives involve the following features and functionalities:

- The ability to query the topology of the transport network.
- The ability to query and update the status of existing and/or configured flows, and their properties and attributes.
- The capability to perform constrained-based path computation queries and produce their response.
- Finally, the ability to asynchronously send notification events to the client.

The validation environment is the CTTC testbed [18], where components defined in the Sections 9.3.1.3 and 9.3.2.2 of this document are extended to add the ABNO orchestration layer. As a general description, the validation procedures involve, mainly:

- the implementation and use of interfaces between the ABNO and the client.
- the implementation and use of interfaces between the ABNO and the SDN controller(s).

Test Card #	*Test Name*
CTTC-ABNO-T001	Topology recovery
CTTC-ABNO-T002	Connectivity Service provisioning across Multi- domain networks
CTTC-ABNO-T003	Functional assessment of the SDN notifications

A more detailed description of the validation procedures and its associated test cards are defined in Appendix II (Section 12.4).

9.4. Mapping of test-cases, 5G-Crosshaul objectives and 5GPPP KPIs

The 5G-Crosshaul project defines eight main objectives, which drive the execution of the activities across the different work packages. Each objective is associated to a number of 5GPPP KPIs and their evaluation (from a functional or non-functional perspective). Table 31 analyses how the prototype test cases and analytical algorithms verification procedures and results defined in this document match the relevant project objectives and 5GPPP KPIs, identifying the WP3 contribution to the validation of the whole project. It should be noted that, in several cases, the architecture and prototypes built in WP3 constitute the functional enablers for the upper layer SDN applications (WP4 scope), which actually implement the logic to meet the KPIs. This is reported as “functional enabler for <application>” in the table below.

Table 31 – Mapping between test-cases, objectives and KPIs

Objective	**5GPPP KPI impact**	**Feature**	**Test Case / Algorithm**
Obj.1: Design of the Crosshaul Control Infrastructure (XCI)	Increase the number of connected devices per area at least by a factor of 10.	Multi-tenancy support at XPFEs and XCI SDN controllers (Functional enabler for Multi-Tenancy Appl.)	NOKIA_XPFE_01 NOKIA_XPFE_02 NXW_XCI_01 NXW_XCI_04
		Network clustering for hierarchical SDN control	CTTC-ABNO-T001 CTTC-ABNO-T002 CTTC-ABNO-T003
	Energy efficiency improvement by at least a factor of 3.	Energy monitoring at XFEs and XPUs. Management of status changes in XFEs and XPUs. Computation of energy efficient	NOKIA_XPFE_03 NXW_EMMA_MANO_01 NXW_EMMA_MANO_02 NXW_EMMA_MANO_03 NXW_XCI_02 NXW_XCI_03 NXW_XCI_04

		network paths. (Functional enabler for Energy Monitoring and Management Appl.)	NXW_XCI_05 NXW_XCI_06 Power consumption computation (Section 6.2)
Obj.2: Specify the XCI's northbound (NBI) and southbound (SBI) interfaces	Enable the introduction and provisioning of new 5G Crosshaul services in the order of magnitude of hours	Abstract network information model for 5G Crosshaul technologies (Functional)	NOKIA_XPFE_03 NXW_XCI_01-02-05 CTTC-mmWave-F01-T001-002-003 CTTC-mmWave-F02-T001-002 IDCC_XCI_01-02 CTTC-ABNO-T001-003
		XCI SBI actions (Functional)	NOKIA_XPFE_0* NXW_XCI_01-05 CTTC-mmWave-F01-T002 CTTC-mmWave-F02-T002 IDCC_XCI_02
		XCI NBI actions (Functional)	ATOS_vCDN_MANO_01-03 NXW_EMMA_MANO_02-03 CTTC-VIMaP-T005 NXW_XCI_05-06 CTTC-mmWave-F03-T001 IDCC_XCI_02
		XCI automated functions for service provisioning	ATOS_vCDN_MANO_03 NXW_EMMA_MANO_03 CTTC-VIMaP-T003-006 NXW_XCI_06 CTTC-mmWave-F03-T00* IDCC_XCI_02-03 CTTC-ABNO-T002 Network optimization (section 6.1)
Obj.3: Unify the 5G Crosshaul data plane	CAPEX and OPEX savings due to the unified data plane (25%) and multi-tenancy.	Support for multi-tenancy in the unified data plane. (Functional)	NOKIA_XPFE_01 NOKIA_XPFE_02
		Design the XFE	IDCC_XPFE_0* NOKIA_XPFE_0*
Obj.6: Design scalable algorithms for efficient 5G Crosshaul resource orchestration	Increase of total Xhaul network throughput by > 20% by means of resource optimization alone compared to current	Novel 5G-capable routing and traffic engineering algorithms	Network optimization (Section6.1)

	operators' practice.		
Obj.7: Design essential 5G Crosshaul integrated (control & planning) applications	Reduce energy consumption in the 5G Crosshaul by 30% through energy management.	Control of optimal scheduling of equipment sleep cycles, routing and function placement (Functional enabler for Energy Monitoring and Management Appl.)	NOKIA_XPFE_03 NXW_EMMA_MANO_01 NXW_EMMA_MANO_02 NXW_EMMA_MANO_03 NXW_XCI_02 NXW_XCI_03 NXW_XCI_04 NXW_XCI_05 NXW_XCI_06 Power consumption computation (section 6.2)
Obj.8: 5G Crosshaul key concept validation and proof of concept	Orchestration of 5G Crosshaul resources based on traffic load variation	Resource orchestration for CDN (Functional enabler for CDN Management Appl.)	ATOS_vCDN_MANO_01-03
	Self-healing mechanisms for unexpected link failures	Network path recovery	CTTC-mmWave-F03-T005 IDCC_XCI_03

10. Conclusion

This document provided a detailed description of the XCI design and its main elements, which is in line with the 5G-Crosshaul System Architecture defined in WP1. It includes a preliminary selection of the software building blocks inside the XCI, along with a discussion of different deployment models of the XCI. Furthermore, it includes the set of software frameworks and platforms that have been selected for the XCI implementation.

This document also described the work during year 1 regarding XCI interfaces. In particular, we provided the initial specification of the NBI that is exposed by several XCI services towards the 5G-Crosshaul applications. Specifically, we have provided an initial API design for each NBI service, the most relevant information of their data model, and a workflow to illustrate the use of each service by a generic 5G-Crosshaul application or by an internal module inside the XCI. With respect to the SBI, this document merely provided a brief summary that links with WP2 and WP3-WP2 joint work, which has been fully reported in D2.1[1] for the sake of self-completeness.

Finally, this document presented the initial methodology defined to validate the functionalities of the XPFEs and XCI prototypes under development in WP3 as well as to evaluate the performance of the related analytics algorithms for modelling computational and network optimization. The initial procedures defined in this document constitute an important step towards the development of stable prototypes to WP5. Note that the aforementioned specification of the XCI procedures will be continuously improved based on the feedback resulting of the development and testing of each of the modules in WP3, and their subsequent integration in WP5.

11. Appendix I – Validation procedures of XCI MANO Components

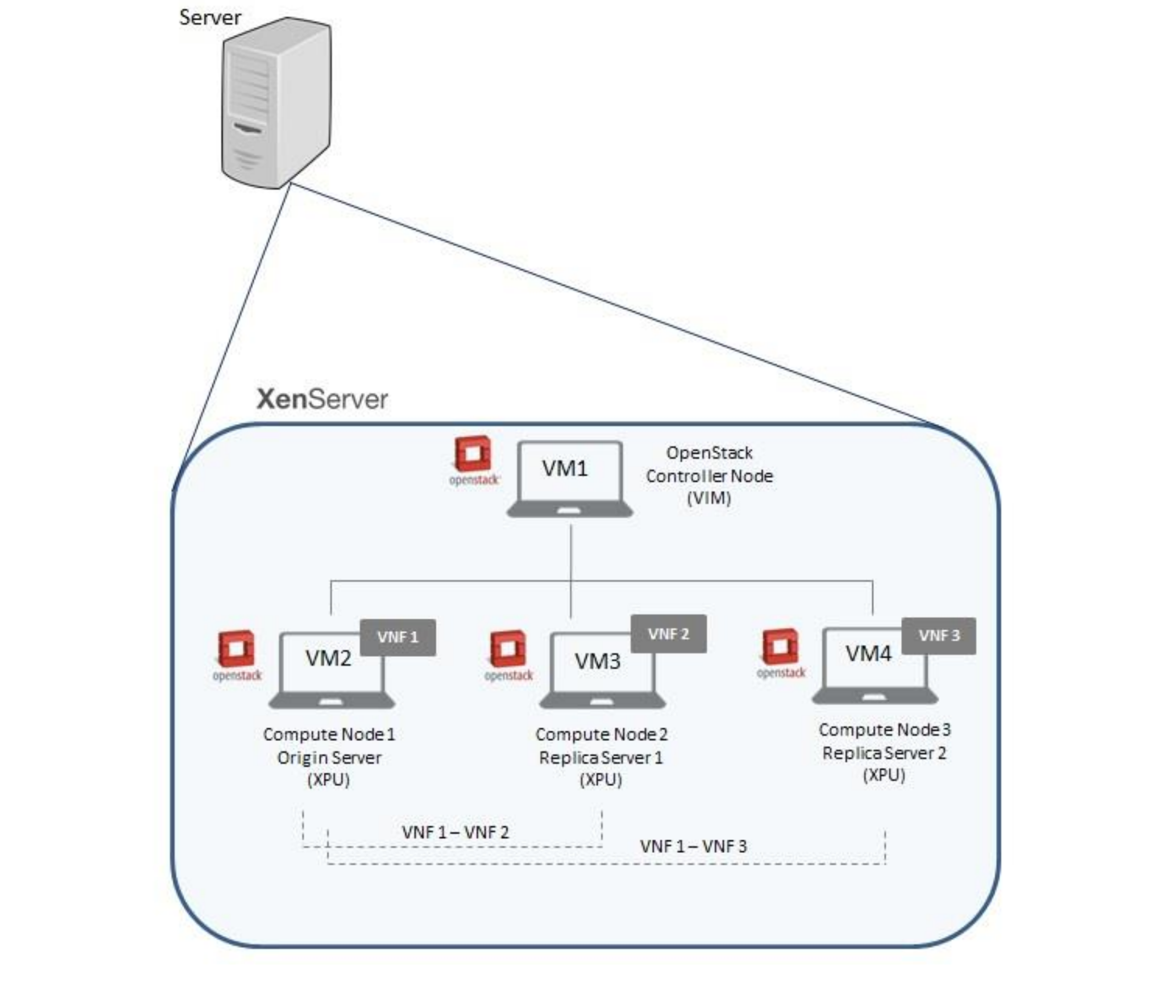
11.1. XCI MANO for CDN

This section contains the specification of the different tests to validate the XCI MANO procedures for the CDN. The test procedures follow sequential steps. First, the instantiation of VMs to run the VNFs on vCDN nodes is validated. Second, the statistics collection of the performance of the vCDN deployment is tested. Finally, the validation of the previous steps serves for the evaluation of the proper orchestration and management of the vCDN infrastructure.

11.1.1. Starting up the VMs and running the vCDN node VNFs

Test Card #	ATOS_vCDN_MANO_01	Execution Status	Testing
Test Name	Starting up the VMs and running the vCDN node VNFs		
Objectives	Starting up the VMs through the VIM. Running the vCDN node VNFs on these VMs. Enabling the proper end-to-end connectivity between the vCDN nodes through the VIM functions.		
Related Use Cases	Media Distribution: vCDN		
Responsible	ATOS		

SUT and topology	
SUT	VIM – OpenStack Mitaka version VNFs – vCDN nodes based on Wowza Streaming Engine media server software
Test environment topology	

 Server XenServer VM1 OpenStack Controller Node (VIM) openstack VM2 VNF 1 Compute Node 1 Origin Server (XPU) VM3 VNF 2 Compute Node 2 Replica Server 1 (XPU) VM4 VNF 3 Compute Node 3 Replica Server 2 (XPU) VNF 1 – VNF 2 VNF 1 – VNF 3	
External components	Interaction with the VIM through Horizon, which is the OpenStack´s dashboard and provides a web-based user interface to OpenStack services.

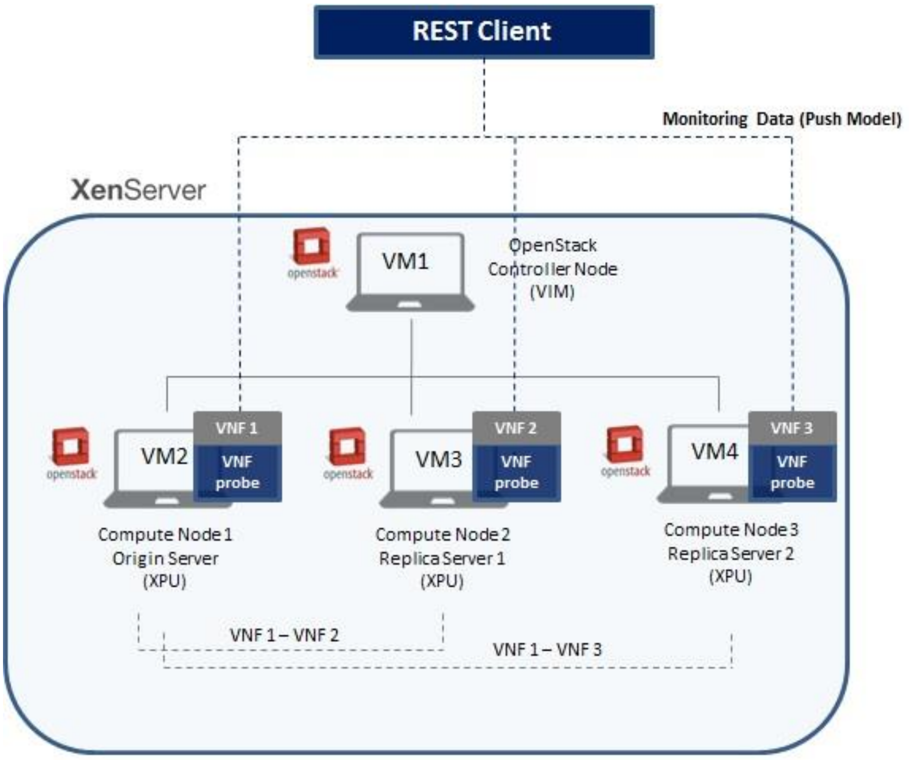
Test description		
Step #	Step description and expected results	Status
1.	**Description:** Initial configuration of OpenStack, set up of controller node and compute nodes. **Expected Results:** OpenStack is running on the controller and compute nodes properly.	
2.	**Description:** Instantiation of the vCDN node VNFs on the compute nodes defined in	

	the previous step. **Expected Results:** vCDN node VNFs are correctly configured and provide the data required (video files).	
3.	**Description:** Establishment of the proper connectivity between the vCDN nodes through the VIM functions. **Expected Results:** Video files are properly sent from Origin vCDN node to Replica vCDN nodes.	

11.1.2. Collection and storage of VM and VNF monitoring data

Test Card #	ATOS_vCDN_MANO_02	Execution Status	Planned
Test Name	Collection and storage of VM and VNF monitoring data		
Objectives	Check the mechanisms to collect the monitoring data required for the vCDN performance.		
Related Use Cases	Media Distribution: vCDN		
Responsible	ATOS		
Related Test Cards	N.A.		

SUT and topology	
SUT	VIM – OpenStack Mitaka version VNFs – vCDN nodes based on Wowza Streaming Engine media server software, supporting VNF probes for monitoring.
Test environment topology	

 REST Client Monitoring Data (Push Model) XenServer openstack VM1 OpenStack Controller Node (VIM) VNF 1 VM2 VNF probe Compute Node 1 Origin Server (XPU) VNF 2 VM3 VNF probe Compute Node 2 Replica Server 1 (XPU) VNF 3 VM4 VNF probe Compute Node 3 Replica Server 2 (XPU) VNF 1 – VNF 2 VNF 1 – VNF 3	
External components	- Interaction with the VIM through Horizon, which is the OpenStack´s dashboard and provides a web-based user interface to OpenStack services. - REST client to check the monitoring data provided by the VNF probes.

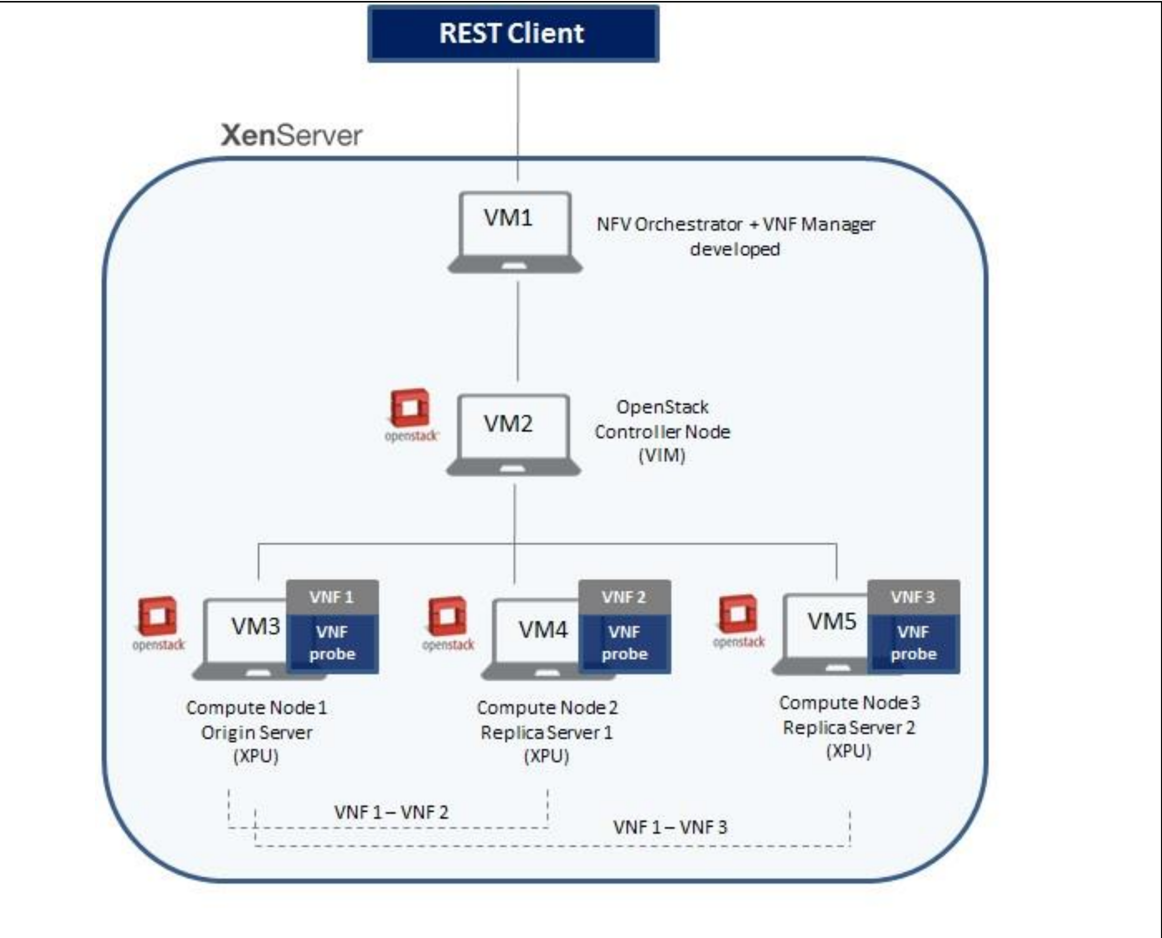
Test description		
Step #	Step description and expected results	Status
1.	**Description:** Run OpenStack and the vCDN node VNFs. **Expected Results:** OpenStack is running on the controller and compute nodes properly. vCDN node VNFs running on the compute nodes. VNF probes (specific software installed on the VNFs) are configured and running on the vCDN node VNFs.	

2.	**Description:** Using the REST client to check the data collected from the VFN probes. **Expected Results:** Monitoring data specified are properly gathered. The VNF probes have to provide the required data: status (up, down), CPU load (%), memory load (%) and number of connected users (n).	

11.1.3. Orchestration and management of a vCDN infrastructure

Test Card #	ATOS_vCDN_MANO_03	Execution Status	Planned
Test Name	Orchestration and management of a vCDN infrastructure.		
Objectives	Manage and process a CDN service instantiation request. Check the whole process to deploy a vCDN infrastructure.		
Related Use Cases	Media Distribution: vCDN		
Responsible	ATOS		
Related Test Cards	N.A.		

SUT and topology	
SUT	VIM – OpenStack Mitaka version VNFs – vCDN nodes based on Wowza Streaming Engine media server software, supporting VNF probes for monitoring. VNF Manager developed – Does the life cycle management of vCDN node VNFs. NFV Orchestrator developed – Manages the CDN service, Network Service Descriptors (NSD), vCDN node VNF Descriptors (VNFD) and vCDN VNF-FG Descriptors (VNF-FGD).
Test environment topology	

 REST Client XenServer VM1 NFV Orchestrator + VNF Manager developed openstack VM2 OpenStack Controller Node (VIM) VM3 VNF 1 VNF probe VM4 VNF 2 VNF probe VM5 VNF 3 VNF probe Compute Node 1 Origin Server (XPU) Compute Node 2 Replica Server 1 (XPU) Compute Node 3 Replica Server 2 (XPU) VNF 1 – VNF 2 VNF 1 – VNF 3	
External components	• REST client to interact with the NFV Orchestrator.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** Start the NFV Orchestrator, the VNF Manager and OpenStack, controller node and compute nodes. Configure the interaction between the NFV-O, VNFM and OpenStack. Load the vCDN node VNFDs and the NSD in the catalogues. Load the vCDN node VNF images in OpenStack. **Expected Results:** OpenStack is running on the controller and the compute nodes properly.	

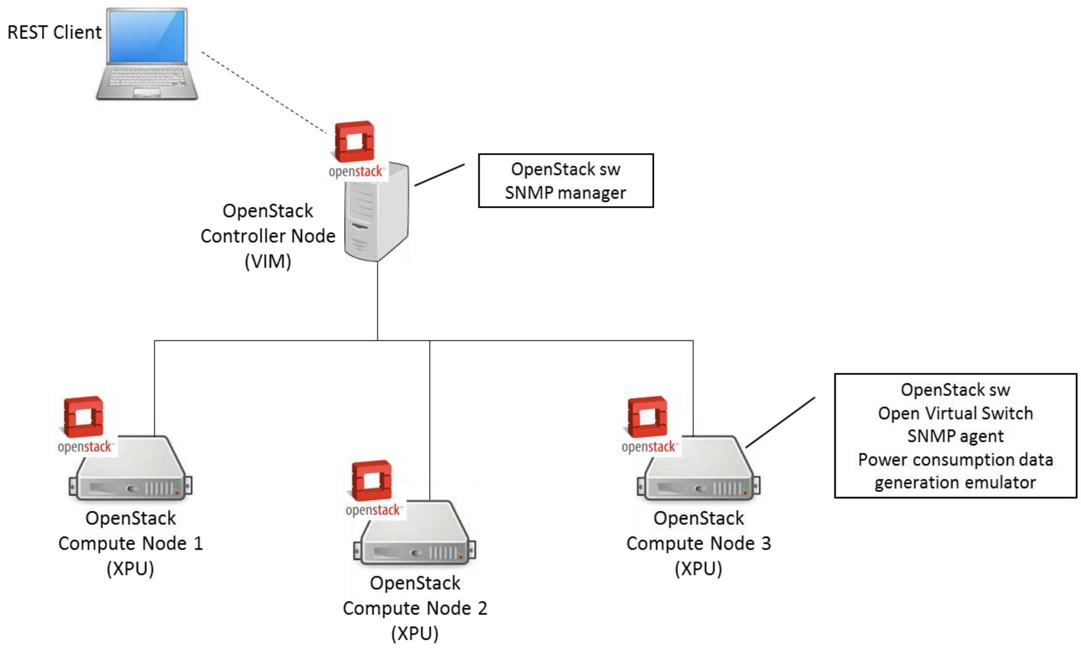
	NFV-O and VNFM are running and working correctly. vCDN node VNFDs and NSD available in the NFV-O.	
2.	**Description:** Request the CDN service instantiation through the REST Client to the NFV-O. **Expected Results:** The NFV Orchestrator through its northbound must understand the network service request, manage the data provided in it (network service template) and interact with the VNF Manager and the VIM in order to deploy the vCDN infrastructure required. The NFV-O takes the vCDN node VNFDs and NSD from the catalog. The VIM has to instantiate the vCDN node VNFs on the compute nodes required. The VNF Manager has to configure the vCDN node VNFs instantiated by the VIM. The vCDN node VNFs are running with the correct configuration.	

11.2. XCI MANO for vEPC and energy management

This section specifies additional details for the testing procedures and components of the MANO for vEPC and energy management described in Section 9.3.1.2.

11.2.1. Monitoring of XPU energy consumption

Test Card #	NXW_EMMA_MANO_01	Execution Status	Planned
Test Name	Monitoring of XPU energy consumption		
Objectives	Verify the mechanisms to collect XPU monitoring data at the VIM and expose them through the VIM's NBI.		
Related Use Cases	Dense urban information society (however energy monitoring is a transversal feature that can be applicable to other use cases, including MEC and CDN).		
Responsible	NXW		
Related Test Cards	N/A (not applicable)		

SUT and topology	
SUT	OpenStack based VIM extended for power consumption monitoring.
Test environment topology  REST Client openstack OpenStack Controller Node (VIM) OpenStack sw SNMP manager openstack OpenStack Compute Node 1 (XPU) openstack OpenStack Compute Node 2 (XPU) openstack OpenStack Compute Node 3 (XPU) OpenStack sw Open Virtual Switch SNMP agent Power consumption data generation emulator	
External components	- REST client to interact with the VIM - Network infrastructure for data plane (between compute nodes) and control plane (between compute nodes and controller nodes). This network infrastructure is statically configured, since out of scope for this test.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** start the software components in compute and controller nodes. **Expected Results:** OpenStack software up and running in compute and controller nodes. SNMP agents and manager up and running in compute nodes and controller node respectively. The SNMP manager sends periodical GET requests to the SNMP agents. Emulators of the power consumption data generator up and running in	

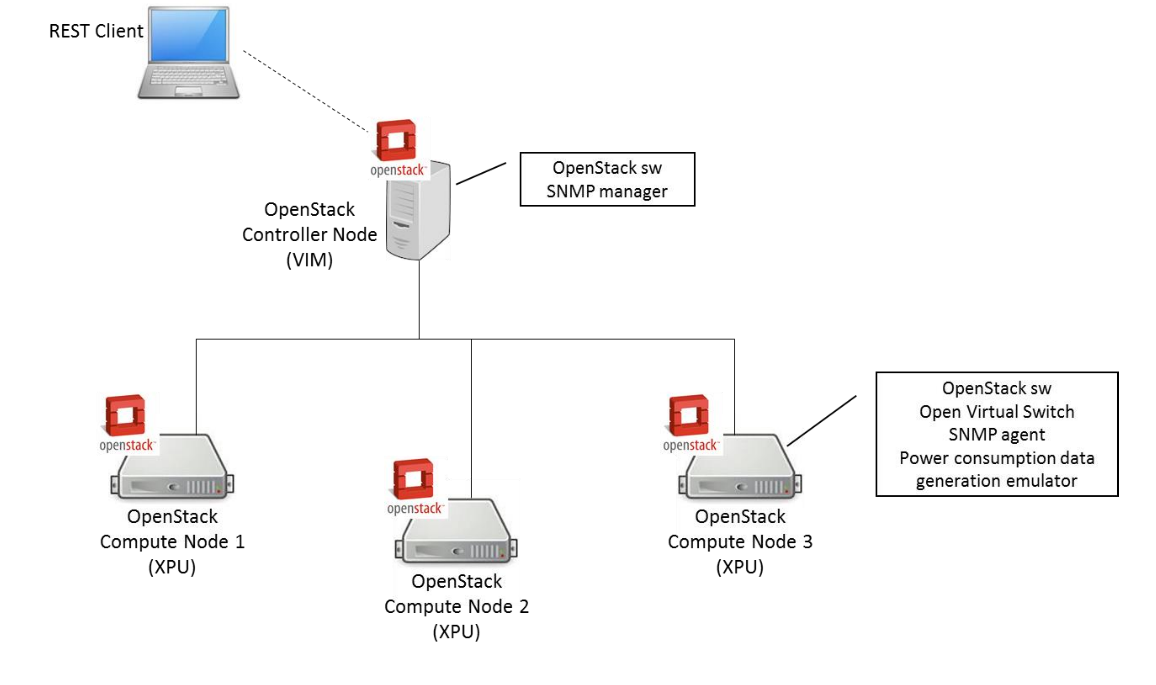
	compute nodes. Emulated data are periodically generated.	
2.	**Description:** using the Wireshark[13] tool verify the exchange of SNMP messages between VIM and XPUs. **Expected Results:** SNMP messages with power consumption data are properly exchanged through the VIM SBI.	
3.	**Description:** using the REST client retrieve power consumption information from the VIM NBI. This data include real-time power consumption values per single servers (their aggregation per VNF or Network Service level is a task of upper layer analytics at the EMMA application and at the NFV-O). **Expected Results:** power consumption data are correctly retrieved.	

11.2.2. Regulation of power-on/power-off status of XPUs

Test Card #	NXW_EMMA_MANO_02	Execution Status	Planned
Test Name	Regulation of power-on/power-off status of XPUs		
Objectives	Verify the mechanisms to switch-on and switch-off XPUs through the VIM.		
Related Use Cases	Dense urban information society (however energy monitoring is a transversal feature, which can be applicable to other use cases, including MEC and CDN).		
Responsible	NXW		
Related Test Cards	N/A (not applicable)		

SUT and topology	
SUT	OpenStack based VIM supporting power-on and power-off actions for the hosts.
Test environment topology	

[13] https://www.wireshark.org/

 REST Client openstack OpenStack Controller Node (VIM) OpenStack sw SNMP manager openstack OpenStack Compute Node 1 (XPU) openstack OpenStack Compute Node 2 (XPU) openstack OpenStack Compute Node 3 (XPU) OpenStack sw Open Virtual Switch SNMP agent Power consumption data generation emulator	
External components	• REST client to interact with the VIM • Network infrastructure for data plane (between compute nodes) and control plane (between compute nodes and controller nodes). This network infrastructure is statically configured, since out of scope for this test.

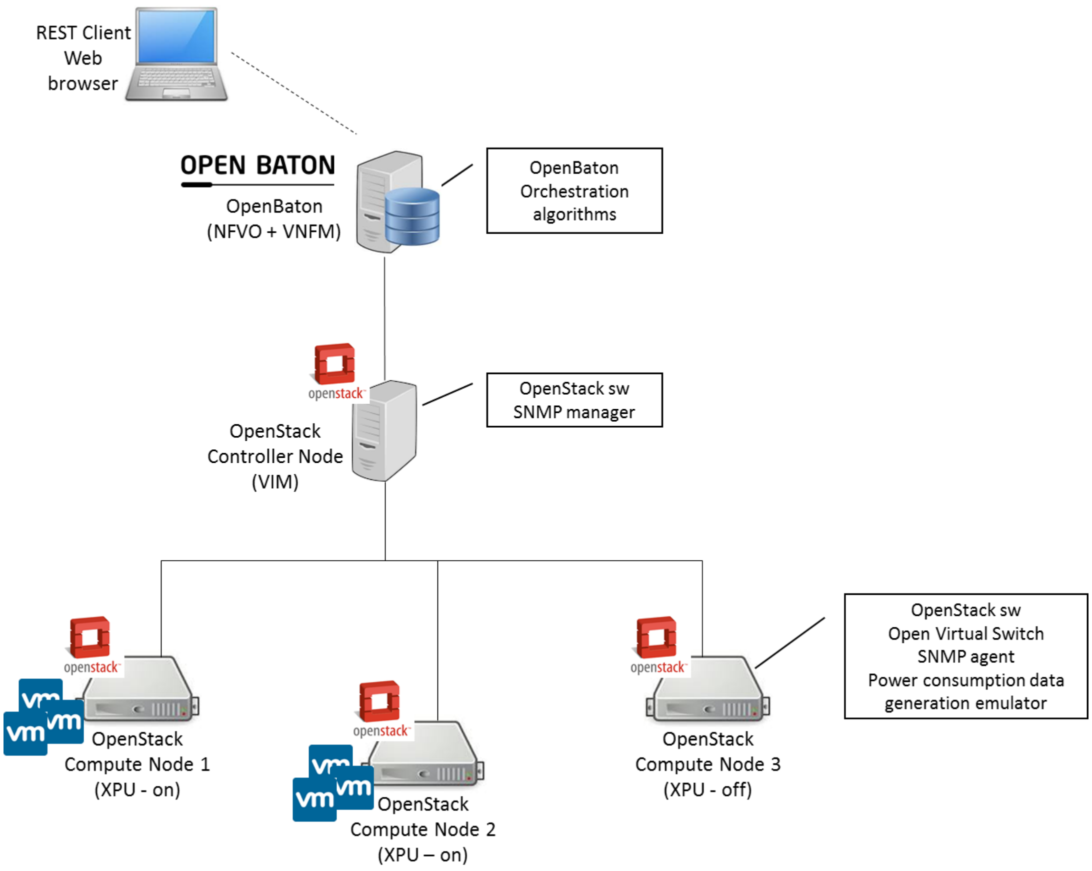
Test description		
Step #	Step description and expected results	Status
1.	**Description:** start the software components in compute and controller nodes. **Expected Results:** OpenStack software up and running in compute and controller nodes. SNMP agents and manager up and running in compute nodes and controller node respectively. The SNMP manager sends periodical GET requests to the SNMP agents. Emulators of the power consumption data generator up and running in compute nodes. Emulated data are periodically generated.	
2.	**Description:** Use the REST client to retrieve the list and status of the compute nodes and send a command to switch off compute node 1.	

	Expected Results: Compute node 1 switched off. **Note:** in a global workflow, the upper layer EMMA application will initiate the action of switching off the hosts, although here the user (REST client) manually triggers it through an explicit REST request.	
3.	**Description:** using the REST client, retrieve the list and status of the hosts and sends a command to switch on compute node 1. **Expected Results:** compute node 1 switched on. **Note:** in a global workflow, the upper layer EMMA application and NFV-O, when VMs need to be allocated in hosts that were previously off, will initiate the action of switching off the hosts, although here the user (REST client) manually triggers it through an explicit REST request.	

11.2.3. Instantiation of energy-efficient vEPC instances

Test Card #	NXW_EMMA_MANO_03	Execution Status	Planned
Test Name	Instantiation of energy-efficient vEPC instances		
Objectives	Verify the mechanisms to instantiate a vEPC instance selecting the more suitable set of compute nodes to reduce the global power consumption. This test will also verify the time needed to provision a vEPC instance, including both cases where compute nodes are already available, up and running as well as cases where the required compute nodes need to be switched on. This allows to evaluate the additional delay required to switch on the nodes that is introduced by the energy-efficient approach.		
Related Use Cases	Dense urban information society (however energy monitoring is a transversal feature which can be applicable to other use cases, including MEC and CDN).		
Responsible	NXW		
Related Test Cards	NXW_EMMA_MANO_02		

SUT and topology	
SUT	OpenStack based VIM supporting power consumption monitoring and power-on/power-off actions for the hosts. Images of vEPC VMs are pre-loaded in OpenStack. OpenBaton based NFV-O with algorithms for energy efficient provisioning of

<table>
<tr><td></td><td>Network Services. VNF and Network Service descriptors (VNFD and NSD) for the vEPC must be loaded in the NFV-O catalogues.
OpenBaton based VNFM for the management of the vEPC VNFs lifecycle.</td></tr>
<tr><td colspan="2">Test environment topology


REST Client Web browser
OPEN BATON
OpenBaton (NFVO + VNFM)
OpenBaton Orchestration algorithms
openstack
OpenStack Controller Node (VIM)
OpenStack sw SNMP manager
vm vm vm
OpenStack Compute Node 1 (XPU - on)
OpenStack Compute Node 2 (XPU – on)
OpenStack Compute Node 3 (XPU - off)
OpenStack sw Open Virtual Switch SNMP agent Power consumption data generation emulator

The initial status of the infrastructure includes two compute nodes up and running and a compute node in shutdown mode. This allows verifying that the algorithms initially select active XPUs, while the third compute node is chosen only if no more resources are available on the previous ones.
The resources made available for OpenStack on Compute Node 1 and Compute Node 2 are enough to support only three instances of the vEPC service.</td></tr>
<tr><td>External components</td><td>• REST client to interact with the VIM
• Network infrastructure for data plane (between compute nodes) and control plane (between compute nodes and controller nodes). This network infrastructure is statically configured, since out of scope for this test.</td></tr>
</table>

Test description		
Step #	Step description and expected results	Status
1.	**Description:** start the software components in NFV-O-VNFM and VIM compute and controller nodes. Configure the NFV-O to use the OpenStack VIM (via NFV-O GUI). Load the vEPC VNFDs and NSD in the NFV-O (via NFV-O GUI). **Expected Results:** OpenStack software up and running in compute and controller nodes. vEPC VM images already available in OpenStack. SNMP agents and manager up and running in compute nodes and controller node respectively. The SNMP manager sends periodical GET requests to the SNMP agents. Emulators of the power consumption data generator up and running in compute nodes. Emulated data are periodically generated. OpenBaton software up and running in the NFV-O andVNFM machines. OpenBaton configured to interact with the OpenStack VIM. VNFD and NSD for vEPC available in the NFV-O catalogues.	
2.	**Description:** using the REST client, request the creation of 3 vEPC service instances to the NFV-O. Measure the time required to provision each service, including details about the delay contribution related to deployment and configuration of each single VM. **Expected Results:** the NFV-O algorithms select compute nodes 1 and 2 as target location for the VMs. The NFV-O, through the VNFM, requests the allocation of the NFVI resources to the VIM, specifying the location of the VMs in compute nodes 1 and 2. The VIM creates the virtual resources as specified by the NFV-O. The VMs of the vEPC are instantiated on the compute nodes 1 and 2. Compute node 3 is still in shutdown mode. The vEPC VMs are up and running, with the correct configuration. The VMs within each service instance are able to interact each other. Collection of KPIs related to provisioning time (VMs deployment and VMs configuration).	
3.	**Description:** using the REST client, request the creation of an additional vEPC service instance to the NFV-O. Measure the time required to	

	provision the service, as in the previous step, and including also the time required to switch on the additional compute node. **Expected Results:** the algorithms at the NFV-O select compute node 3 as target location for the VMs. The NFV-O requests the VIM, through its NBI, to switch on compute node 3 (see the procedure in test case NXW_EMMA_MANO_02). Compute node 3 is properly started. The NFV-O, through the VNFM, request the allocation of the NFVI resources to the VIM, specifying the location of the VMs in compute node 3. The VIM creates the virtual resources as specified by the NFV-O. The VMs of the vEPC are instantiated on the compute node 3. The vEPC VMs are up and running, with the correct configuration. The VMs are able to interact each other. Collection of KPIs related to provisioning time (host switching on, VMs deployment and VMs configuration).	

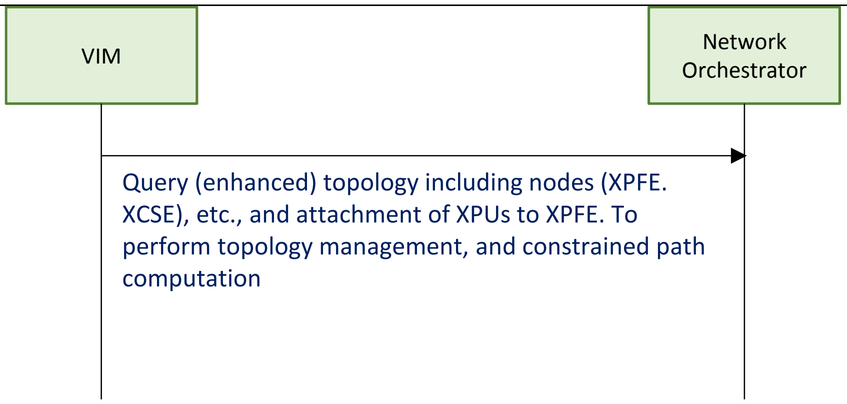
11.3. OpenStack-based VIMaP

This section specifies the testing procedures to validate the different operations and components of the OpenStack based VIMaP described in Section 9.3.1.3. The VIMaP operations are namely: topology detection, XPU status and capability discovery, VMs instantiation, and flow provisioning under different network domains. The last two defined procedures focus on testing the offered NBI interface to interact with the application plane and the integration of the P-component, which performs constrained VM placement.

11.3.1. Topology detection

Test Card #	CTTC-VIMaP-T001	Execution Status	Planned
Test Name	Topology detection		
Objectives	To be able to retrieve the transport network topology from the SDN controller or ABNO orchestrator, following YANG-based API such as COP, topology management service. Note that the ONF Transport API [22] or IETF TEAS topology [23] are other potential alternatives. To validate that the topology is correct and can be used for path computation and resource reservation processes.		

Related Use Cases	All use cases defined in D1.1 [3].
Responsible	CTTC
Related Test Cards	N/A (not applicable).
Additional Comments	This test involves a software module that requests the topology from the SDN controller and is able to store it in an internal database.

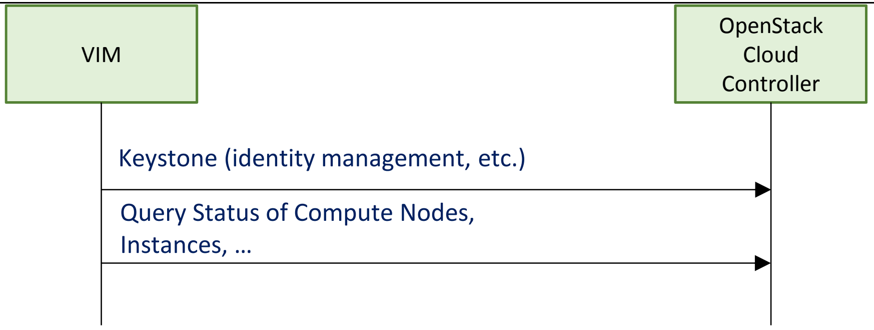
SUT and topology	
SUT	VIMaP and SDN controller (or ABNO orchestrator for multi-domain). Both applications interconnected in a LAN network (IP reachability).
Test environment topology	 VIM Network Orchestrator Query (enhanced) topology including nodes (XPFE. XCSE), etc., and attachment of XPUs to XPFE. To perform topology management, and constrained path computation
External components	Basic control plane infrastructure to enable IP reachability between functional entities. An (emulated or real) network topology that is retrieved by the VIMaP application and exported by the SDN controller for testing purposes.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure a transport network topology in one or multiple domains, with one SDN controller or multiple ones allocated in a hierarchy. **Expected Results:** SDN controller normal operation. It should be possible to consume SDN controller or ABNO orchestrator API by	

	means of COP protocol, and retrieve topological information.	
2.	**Description:** execute VIMaP application and provide the IP address and port of the services exported by the SDN controller or ABNO orchestrator. Trigger an event that results in the VIMaP requesting the topology using the COP protocol (the event can typically later be mapped to a client request). Measure the time required by the VIMaP application to retrieve the network topology. **Expected Results:** the VIMaP stores topological information that can be exported, e.g., to an external file or other entities. The VIMaP uses that topology information, e.g., for the purposes of path computation. The topology is correct and as intended. Collection of topology retrieve time related operations. The comparison of VIMaP topology recovery execution time between different network deployments provides an idea of the degree of scalability.	

11.3.2. XPU status and capability discovery

Test Card #	CTTC-VIMaP-T002	Execution Status	Planned
Test Name	XPU status and capability discovery.		
Objectives	To be able to retrieve the resources controlled by the OpenStack controller and, in particular, the number of compute nodes and availability zones, their capacities, the number of instantiated Virtual Machines and the consumed resources. Validate that the resource view is correct and can be used for VM allocation purposes. This includes the ability to consume Keystone (identity), Glance (software images) and Nova (computing) services.		
Related Use Cases	All use cases defined in D1.1 [3].		
Responsible	CTTC		
Related Test Cards	N/A (not applicable)		
Additional Comments	This test involves a software module that requests the aforementioned services from the cloud controller.		

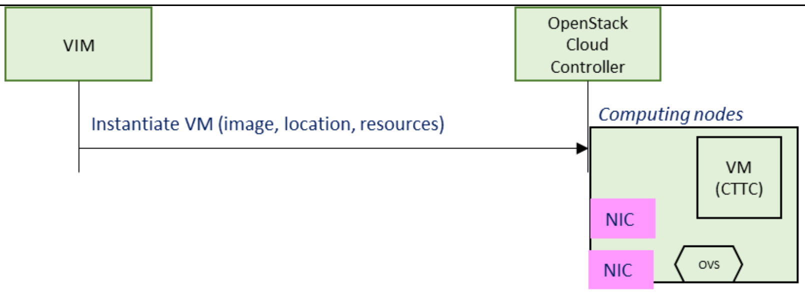
SUT and topology	
SUT	VIMaP and OpenStack cloud controller. Both applications interconnected in a LAN network (IP reachability).
Test environment topology	 VIM OpenStack Cloud Controller Keystone (identity management, etc.) Query Status of Compute Nodes, Instances, …
External components	Basic control plane infrastructure to enable IP reachability between functional entities. OpenStack deployment, with multiple compute nodes in different availability zones.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure an OpenStack cloud management infrastructure with multiple compute nodes, with emphasis on physical locations that emulate placement of XPUs, retrieving information to integrate into VIMaP databases. **Expected Results:** Operative OpenStack infrastructure	
2.	**Description:** launch manually instances from a set of predefined images in one or more compute nodes. - Single tenant / project (admin) - Same GNU/Linux image - Same IP subnet (prefix) **Expected Results:** running instances as intended.	
3.	**Description:** VIMaP retrieves the status of the compute nodes and the running instances, being able to deduce which nodes can be used for a service. Measure the time required by the VIMaP application to retrieve the status of compute nodes under different compute nodes deployment. **Expected Results:** the retrieved status is correct, and corresponds to the dynamic status of the OpenStack infrastructure. Collection of XPU status time releated operations to assess different	

	XPU deployments.	

11.3.3. Instantiation of Virtual Machines in remote locations

Test Card #	CTTC-VIMaP-T003	Execution Status	Planned
Test Name	Instantiation of VMs in remote locations.		
Objectives	The VIMaP application is able to allocate the set of Virtual Machines provided in a request directly over the OpenStack Infrastructure. This includes the ability to consume keystone (identity), glance (software images) and nova (computing) services.		
Related Use Cases	All use cases defined in D1.1 [3].		
Responsible	CTTC		
Related Test Cards	N/A (not applicable)		
Additional Comments	This test involves a software module that requests the aforementioned services from the cloud controller.		

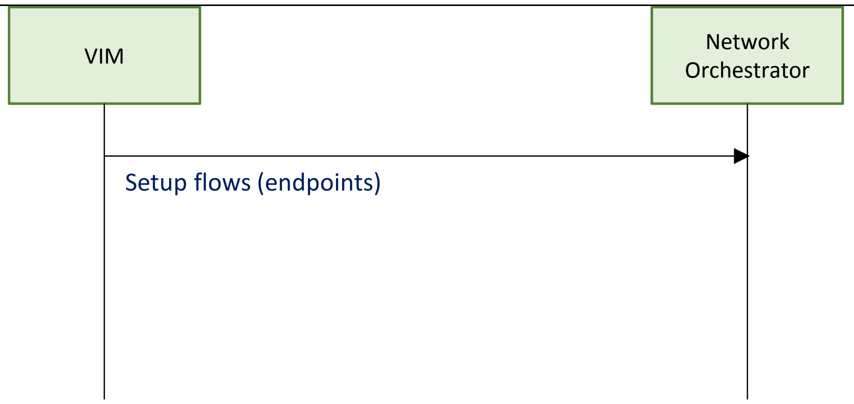
SUT and topology	
SUT	VIMaP and OpenStack cloud controller. Both applications interconnected in a LAN network (IP reachability).
Test environment topology	 VIM OpenStack Cloud Controller Instantiate VM (image, location, resources) Computing nodes VM (CTTC) NIC NIC OVS
External components	Basic control plane infrastructure to enable IP reachability between functional entities. OpenStack deployment, with multiple compute nodes in different availability zones.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure an OpenStack cloud management infrastructure with multiple compute nodes. **Expected Results:** Operative OpenStack infrastructure.	
2.	**Description:** VIMaP is able to launch instances from a set of predefined images in one or more compute nodes. Measure the time required to serve different VM provisioning requests. - Single tenant / project (admin) - Same GNU/Linux image - Same IP subnet (prefix) **Expected Results:** running instances as intended.	
3.	**Description:** VIMaP retrieves the status of the compute nodes and the running instances, being able to deduce which nodes can be used for a service. **Expected Results:** the retrieved status is correct, and corresponds to the dynamic status of the OpenStack infrastructure. Collection of XPU status time related operations to assess different XPU deployments. The combination of this XPU status time operations with the VM provisioning time provides information for further assessment of the complete cycle. NOTE: VMs may also appear in the topological database that is later on retrieved from the SDN controller (as endpoints for the connectivity services).	

11.3.4. Flow configuration across Single- and Multi-domain networks

Test Card 4	CTTC-VIMaP-T004	Execution Status	Planned
Test Name	Flow configuration across Single- and Multi-domain networks		
Objectives	The VIMaP is able to setup flows using a YANG-based API such as COP call and connection control services from the SDN controller or ABNO orchestrator.		
Related Use Cases	All use cases defined in D1.1 [3].		
Responsible	CTTC		

Related Test Cards	N/A (not applicable).
Additional Comments	This test involves a software module that provisions flows across the abstracted topology.

SUT and topology	
SUT	VIMaP and SDN controller or ABNO orchestrator. Both applications to be interconnected in a LAN network (IP reachability).
Test environment topology	
External components	Basic control plane infrastructure to enable IP reachability between functional entities. An emulated or real network topology retrieved by the VIMaP application and exported by the SDN controller for testing purposes.

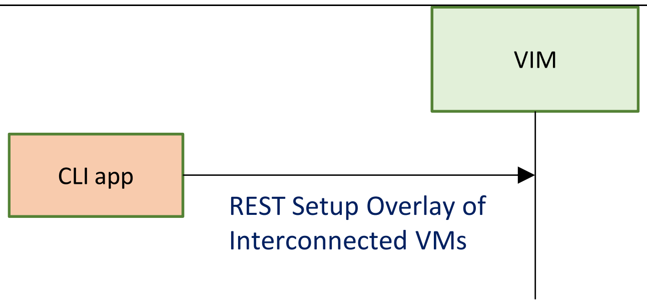
Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure a transport network topology in one or multiple domains, with one SDN controller or multiple ones allocated in a hierarchy. **Expected Results:** SDN controller normal operation. It should be possible to consume SDN controller or ABNO orchestrator API and retrieve topological information.	
2.	**Description:** Execute VIMaP application and provide the IP address and port of the services exported by the SDN controller or ABNO orchestrator.	

	Trigger an event that results in the VIMaP provisioning of flows, between provided end-points. Measure the time required to provision the flow instance in the underlying network resources. **Expected Results:** the endpoints VM are able to communicate as intended (e.g. from VMa to VMb in unidirectional or bi-directional settings) Collection of flow-provisioning time related operations under different transport network topology (one or multiple domains). **Comments:** This test is basic for the inter-VM connectivity provisioning.	

11.3.5. Functional assessment of the VIMaP NBI

Test Card 5	CTTC-VIMaP-T005	Execution Status	Planned
Test Name	Functional assessment of the VIMaP NBI.		
Objectives	The VIMaP exports a North Bound Interface consumed by applications requesting the allocation of VM instances and the provisioning of connectivity services in the 5G-Crosshaul network.		
Related Use Cases	All use cases defined in D1.1 [3].		
Responsible	CTTC		
Related Test Cards	N/A (not applicable)		
Additional Comments	This test involves a request / response protocol between a client and the VIMaP.		

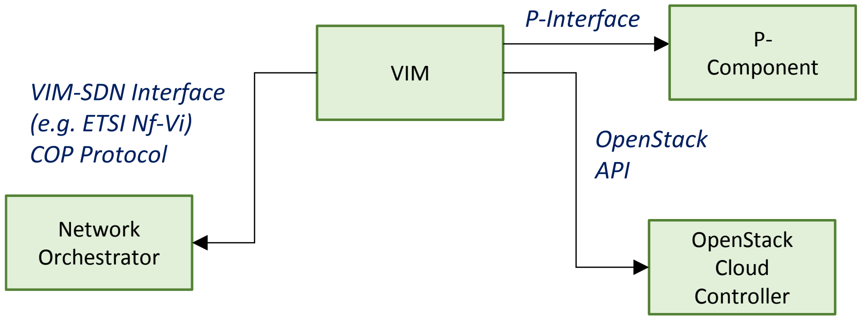
SUT and topology	
SUT	VIMaP and client application (Once integrated, also the SDN controller or ABNO orchestrator, cloud controller, optional P component). Functional components interconnected in a LAN network (IP reachability).

Test environment topology		
External components	Basic control plane infrastructure to enable IP reachability between functional entities. An (emulated or real) network topology that is retrieved by the VIMaP application and exported by the SDN controller for testing purposes. One or more compute nodes to allocate instances.	
Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure a transport network topology in one or multiple domains, with one SDN controller or multiple ones allocated in a hierarchy. **(As previous tests)**	
2.	**Description:** Configure an OpenStack deployment with multiple compute nodes. **(As previous tests)**	
2.	**Description:** Execute VIMaP application and provide the IP address and port of the services exported by the SDN controller or ABNO orchestrator. Provide the endpoints for the cloud controller services. **(As previous tests)**	
3.	**Description:** Use a client mock-up application that consumes the VIMaP NBI for the provisioning of an overlay of interconnected VMs across the Crosshaul infrastructure deployed in CTTC testbed. Measure the time required to serve different VM provisioning requests. **Expected Results:** Service provisioned as requested under different transport network deployments. Collection of KPIs related to VM provisioning time (VMs deployment and VMs configuration) of different requests under different network	

	deployment conditions (SDN hierarchy deployment and network domains). **Comments:** This test means a complete integration between all the involved components (except the P-component, which is optional and detailed in the next test card).	

11.3.6. External placement computation of VMs and flows

Test Card 6	CTTC-VIMaP-T006	Execution Status	Planned
Test Name	External placement computation of VMs and flows (P component).		
Objectives	The VIMaP is able to delegate the placement of the VMs to a separate component via the P-interface.		
Related Use Cases	All use cases defined in D1.1 [3].		
Responsible	CTTC (in charge of the VIM part), TID (in charge of the P part).		
Related Test Cards	N/A (not applicable).		
Additional Comments	This test involves a request / response protocol between the VIM and the P-component to perform constrained VM placement.		

SUT and topology	
SUT	VIMaP and SDN controller or ABNO orchestrator, cloud controller, and P component. Functional components interconnected in a LAN network (IP reachability).
Test environment topology	 P-Interface P-Component VIM VIM-SDN Interface (e.g. ETSI Nf-Vi) COP Protocol OpenStack API Network Orchestrator OpenStack Cloud Controller

External components	Control plane infrastructure to enable IP reachability between functional entities. An emulated or real network topology retrieved by the VIMaP application and exported by the SDN controller for testing purposes. One or more compute nodes to allocate instances.	
Test description		
Step #	Step description and expected results.	Status
1.	**Description:** configure a transport network topology in one or multiple domains, with one SDN controller or multiple ones allocated in a hierarchy. **Expected Results:** it should be possible to consume an SDN controller or ABNO orchestrator API and retrieve topological information.	
2.	**Description:** execute VIMaP application and provide the IP address and port of the services exported by the SDN controller or ABNO orchestrator. Provide the endpoints for the cloud controller services. Trigger an event that results in the VIMaP requesting the P-component the VMs to place and the flows to provision. Measure time required by the P-component for the VM placement and flow provisioning. **Expected Results:** the P-component replies with the compute nodes to use for each VM and the flows to set up. Collection of VMs and flow-provisioning setup time of the P-component under different transport network topologies.	
3.	**Description:** the VIMaP proceeds to instantiate the VMs and the flows as given by the P-Component. Measure time required by the ViMAP to serve the P-Component request (VMs instantiation and flow provisioning). **Expected Results:** service provisioned as requested. Collection of VIMaP time related operations to VM instantiation and flow provisioning. The combination of the P-component request processing time with the VIMaP VM instantiation and flow provisioning times provides information for further assessment of the complete cycle. **Comments:** this test means a complete integration between all the involved components. The actual use in a multi-domain real transport network is orthogonal for VIMaP operation.	

12. Appendix II – Validation procedures of XCI SDN Controllers

This section presents the validation procedures related to the XCI SDN functionality. It is divided in four subsections considering different functions each of the treats. The first subsection considers the SDN control of XPFEs and the energy management functionalities considering the power consumption of the switching elements in the network. In the second subsection, we present the validation of the SDN controller used for the general operation of mmWave and WiFi links. The third subsection focuses on the specific SDN controller application for mmWave mesh technology. Finally, the forth subsection highlights the hierarchy of SDN controllers and how to retrieve the network topology from the underlying controller.

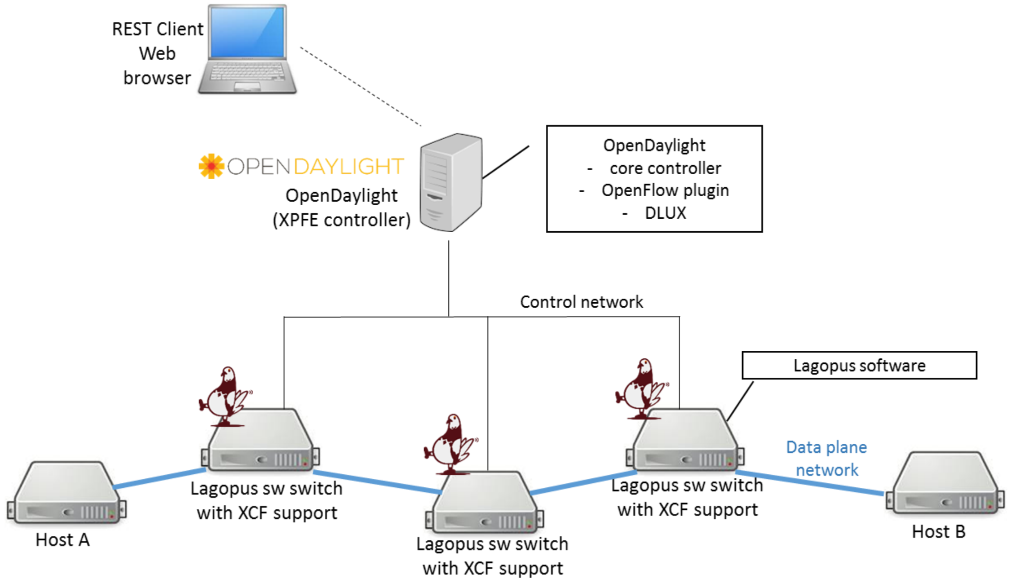
12.1. SDN Controller for XPFE and energy management

This subsection treats the issue of energy management, considering the reduction of power consumption in the data plane elements of the network. The section also considers the exchange of information between the XPFE and the SDN controllers in the XCI, which is given through OpenFlow messages.

12.1.1. Core XCI controller services in XPFE's networks

Test Card #	NXW_XCI_01	Execution Status	Planned
Test Name	Core XCI controller services in XPFE's networks		
Objectives	Verify the mechanisms for basic operation of XPFE nodes based on software switches: • Connection between XPFE node and SDN controller • Exchange of XPFE capabilities and building of the topology • Configuration of flow entries in XPFE nodes for encapsulation, decapsulation and forwarding of XCF frames.		
Related Use Cases	Dense urban information society		
Responsible	NXW		
Related Test Cards	NOKIA_XPFE_01, NOKIA_XPFE_02, NOKIA_XPFE_03		

SUT and topology

SUT	OpenDaylight controller with the following software modules enabled: • Core controller services • L2 switch • OpenFlow plugin • DLUX (web GUI)
Test environment topology  REST Client Web browser OPENDAYLIGHT OpenDaylight (XPFE controller) OpenDaylight - core controller - OpenFlow plugin - DLUX Control network Lagopus software Data plane network Lagopus sw switch with XCF support Lagopus sw switch with XCF support Lagopus sw switch with XCF support Host A Host B	
External components	• REST client to interact with the controller via REST API + web browser to visualize the controller web GUI. • Network infrastructure with XPFEs in the data plane. This network infrastructure is based on Lagopus software switches.

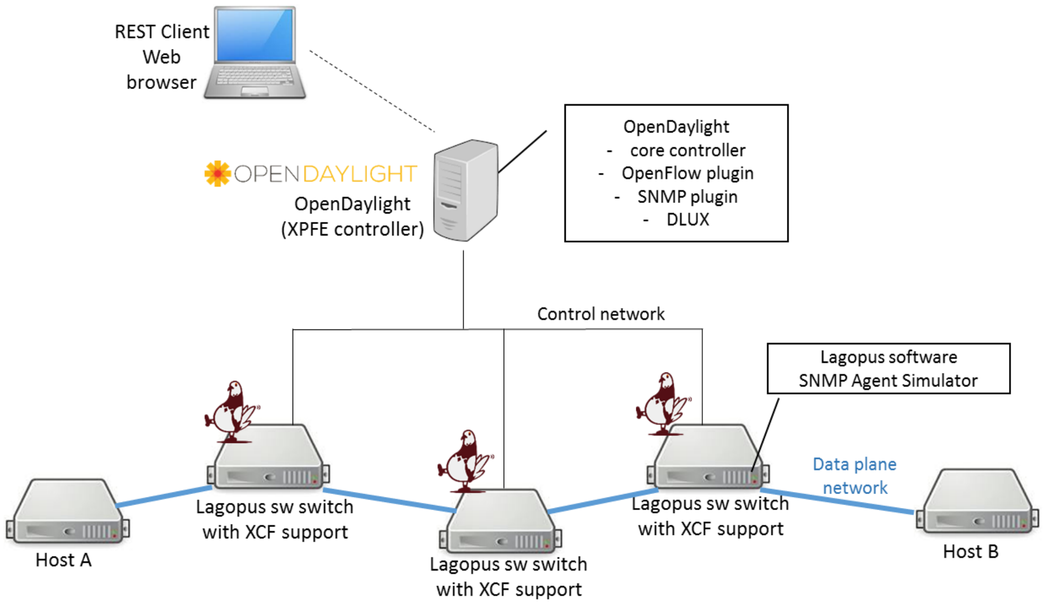
Test description		
Step #	Step description and expected results	Status
1.	**Description:** start OpenDaylight in the SDN controller and activate the required software modules enabling the related karaf features. Start the Lagopus software in the data plane nodes. **Expected Results:** OpenDaylight software up and running in the SDN controller. Required OpenDaylight software bundles in active status.	
2.	**Description:** start the Lagopus software in the data plane nodes. Using the Wireshark tool verify the exchange of OpenFlow messages between software switches and SDN controller.	

	Expected Results: software switches connected to the controller. Initial exchange of OpenFlow messages between switches and SDN controller properly exchanged.	
3.	**Description:** using OpenDaylight web interface visualize the network topology and the nodes in the inventory page. **Expected Results:** the three network nodes are available in the network topology and in the list of nodes in the inventory.	
4.	**Description:** using the controller REST APIs create flow entries in each node in order to create a connection between host A and host B. In particular, the first node should be configured with PUSH_PBB and PUSH_VLAN actions, the second node with forwarding actions and the third node with POP_PBB and POP_VLAN actions. The actual splitting of these commands across the pipeline of flow tables in XPFE nodes is still under discussion in T3.1. Using OpenDaylight web interface visualize the flows installed in the network nodes. Using the Wireshark tool verify the exchange of OpenFlow messages between software switches and SDN controller. **Expected Results:** the FlowMod OpenFlow messages, corresponding to the commands sent via the NBI, are exchanged between controller and software switches. OpenDaylight web interface reports the new flows. The hosts are able to interact.	
5.	**Description:** generate traffic data between the hosts and, using the Wireshark tool, verify the exchange of statistics OpenFlow messages between network nodes and controller. Making use of the REST APIs at the controller, verify the statistics collected and stored at the controller. **Expected Results:** OpenFlow messages about statistics exchanged between controller and network nodes. Statistics data available at the controller.	

12.1.2. Monitoring of XPFE power consumption via SNMP

Test Card #	NXW_XCI_02	Execution Status	Planned
Test Name	Monitoring of XPFE power consumption via SNMP		
Objectives	Verify the mechanisms to collect XPFEs power consumption monitoring data, store them at the core services level (inventory and topology) and expose them through REST APIs.		

Related Use Cases	Dense urban information society
Responsible	NXW
Related Test Cards	N.A.
Additional Comments	SNMP agents on network nodes simulated via SNMP Agent Simulator[14]. The reference SNMP MIBs are IANAPowerStateSet-MIB, ENERGY-OBJECT-MIB and POWER-ATTRIBUTES-MIB specified in [19].

SUT and topology	
SUT	OpenDaylight controller with the following software modules enabled: • Core controller services • L2 switch • OpenFlow plugin • SNMP plugin • DLUX (web GUI)
Test environment topology  REST Client Web browser OPENDAYLIGHT OpenDaylight (XPFE controller) OpenDaylight - core controller - OpenFlow plugin - SNMP plugin - DLUX Control network Lagopus software SNMP Agent Simulator Data plane network Host A Lagopus sw switch with XCF support Lagopus sw switch with XCF support Lagopus sw switch with XCF support Host B	
External components	• REST client to interact with the controller via REST API + web browser to visualize the controller web GUI. • Network infrastructure with XPFEs in the data plane. This network infrastructure is based on machines with Lagopus software switches

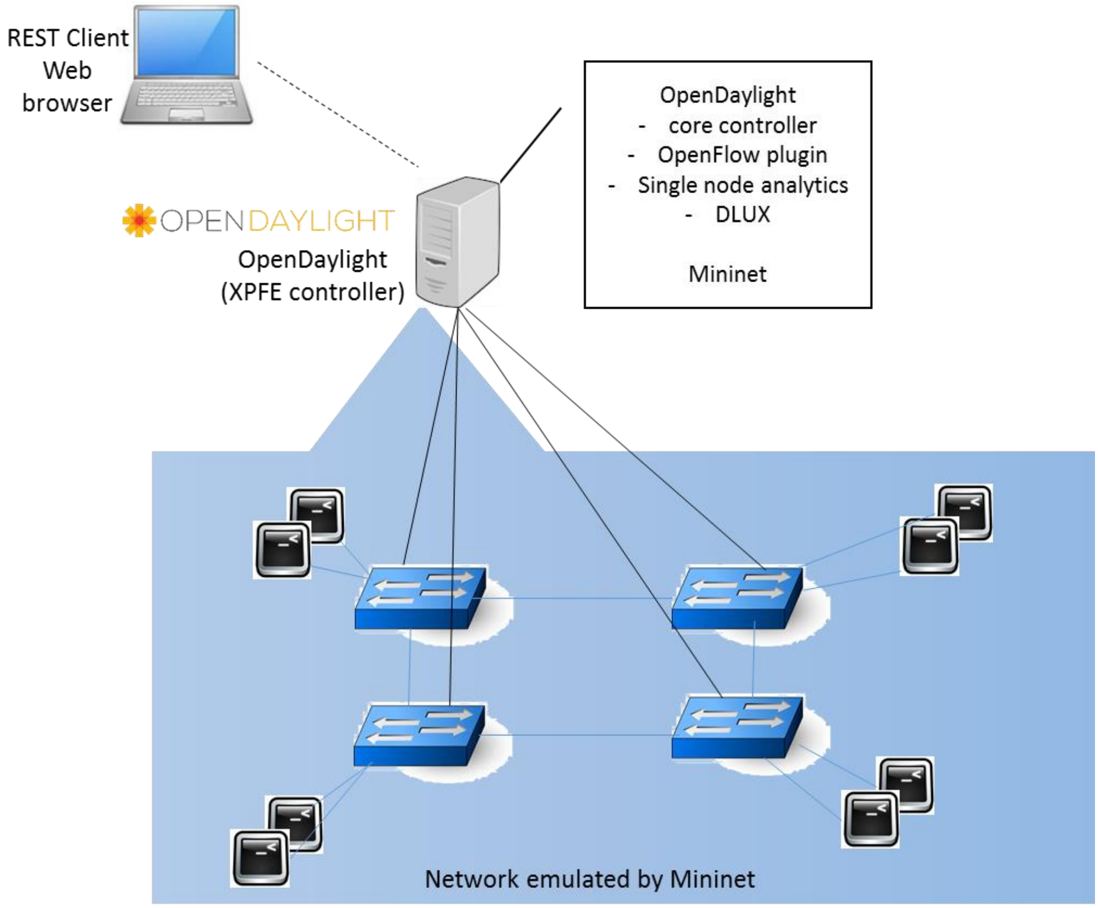
[14] http://snmpsim.sourceforge.net/

	where the SNMP Agent Simulator is also installed.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** start OpenDaylight in the SDN controller and activate the required software modules enabling the related karaf features. In the data plane nodes, start Lagopus software and SNMP Agent Simulator, configured with fake power consumption data. **Expected Results:** OpenDaylight software up and running in the SDN controller node with SNMP manager, extended inventory and topology. SNMP agents up and running in network nodes. Emulated data are periodically generated following the fake data loaded in the configuration. The SNMP manager sends periodical GET requests to the SNMP agents.	
2.	**Description:** using the Wireshark tool verify the exchange of SNMP messages between SDN controller and XPFEs. **Expected Results:** SNMP messages with power consumption data are properly exchanged through the SDN controller SBI.	
3.	**Description:** using the exposed REST APIs, retrieve power consumption information from the extended topology module. **Expected Results:** power consumption data are correctly retrieved.	

12.1.3. Monitoring of XPFE power consumption via analytics

Test Card #	NXW_XCI_03	Execution Status	Planned
Test Name	Monitoring of XPFE power consumption via analytics		
Objectives	Verify the mechanisms to produce power consumption parameters, based on traffic load information, for XPFE nodes that do not support natively power consumption monitoring. The algorithms used to produce this kind of data are described in Section 6.2.		
Related Use Cases	Dense urban information society		
Responsible	NXW, POLITO		
Related Test Cards	N.A.		
Additional	This feature must be enabled to activate power consumption monitoring in nodes		

Comments	without explicit power monitoring support. Monitoring data generated based on an analytical model that takes into account the status of the node, the status of its ports and the traffic load.
SUT and topology	
SUT	OpenDaylight controller with the following software modules enabled: - Core controller services - L2 switch - OpenFlow plugin - Analytics for physical nodes power monitoring - DLUX (web GUI)
Test environment topology	 REST Client Web browser OPENDAYLIGHT OpenDaylight (XPFE controller) OpenDaylight - core controller - OpenFlow plugin - Single node analytics - DLUX Mininet Network emulated by Mininet
External components	- REST client to interact with the controller via REST API + web browser to visualize the controller web GUI. - Network infrastructure emulated by Mininet (see picture above).

Test description

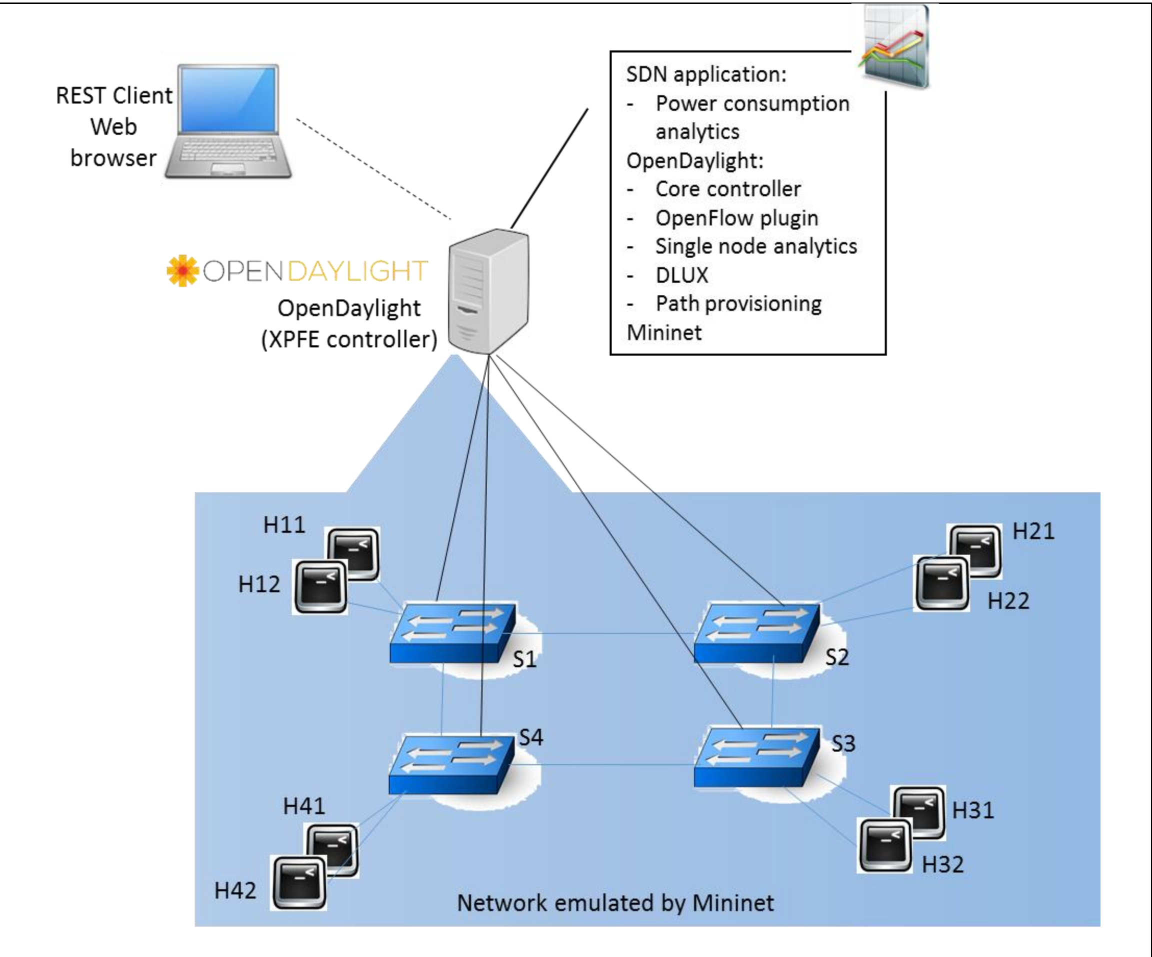
Step #	Step description and expected results	Status
1.	**Description:** start OpenDaylight in the SDN controller and activate the required software modules enabling the related karaf features. Start Mininet with preconfigured topology. On Mininet console, perform a *pingall* command to allow the controller to detect the hosts. Using OpenDaylight web interface visualize the network topology and the nodes in the inventory page. **Expected Results:** OpenDaylight software up and running in the SDN controller node. The controller and the emulated network nodes exchange OpenFlow messages. The controller computes the correct topology, which is shown in the web GUI.	
2.	**Description:** using REST APIs, retrieve power consumption information from the topology. Visualize analytics log to verify: • the collection of information about network nodes, ports and statistics from the proper OpenDaylight services through periodical polling; • the computation of power consumption values and their storage in the topology information. **Expected Results:** power consumption values available in the node elements retrieved from the topology service. Analytics log without errors.	
3.	**Description:** emulate traffic between the hosts using iPerf and verify that power consumption values are increased. **Expected Results:** power consumption values available in the node elements retrieved from the topology service are higher than the ones collected in step 2.	

12.1.4. Analytics for computation of power consumption in virtual infrastructures

Test Card #	NXW_XCI_04	Execution Status	Planned
Test Name	Analytics for computation of power consumption in virtual infrastructures		
Objectives	Verify the mechanisms to compute the power consumption related to the whole physical network and to virtual infrastructures assigned to a given tenant. The algorithms used to compute power consumption for physical networks and		

	virtual network slices are described in Section 6.2.
Related Use Cases	Dense urban information society
Responsible	NXW, POLITO
Related Test Cards	NXW_XCI_01, NXW_XCI_03, NXW_XCI_06
Additional Comments	N.A.

SUT and topology	
SUT	OpenDaylight controller with the following software modules enabled: • Core controller services, L2 switch and OpenFlow plugin • Analytics for physical nodes power monitoring • Path provisioning • DLUX (web GUI) SDN application for power consumption analytics.
Test environment topology	

	
External components	• REST client to interact with the controller via REST API + web browser to visualize the controller web GUI. • Network infrastructure emulated by Mininet (see picture above).

Test description		
Step #	Step description and expected results	Status
	Description: start OpenDaylight in the SDN controller and activate the required software modules enabling the related karaf features. Start Mininet with preconfigured topology. On Mininet console, perform a *pingall* command to allow the controller to detect the hosts. Using OpenDaylight web interface visualize the network topology and the nodes in the inventory page. **Expected Results:** OpenDaylight software up and running in the SDN controller node. The controller and the emulated network nodes exchange OpenFlow	

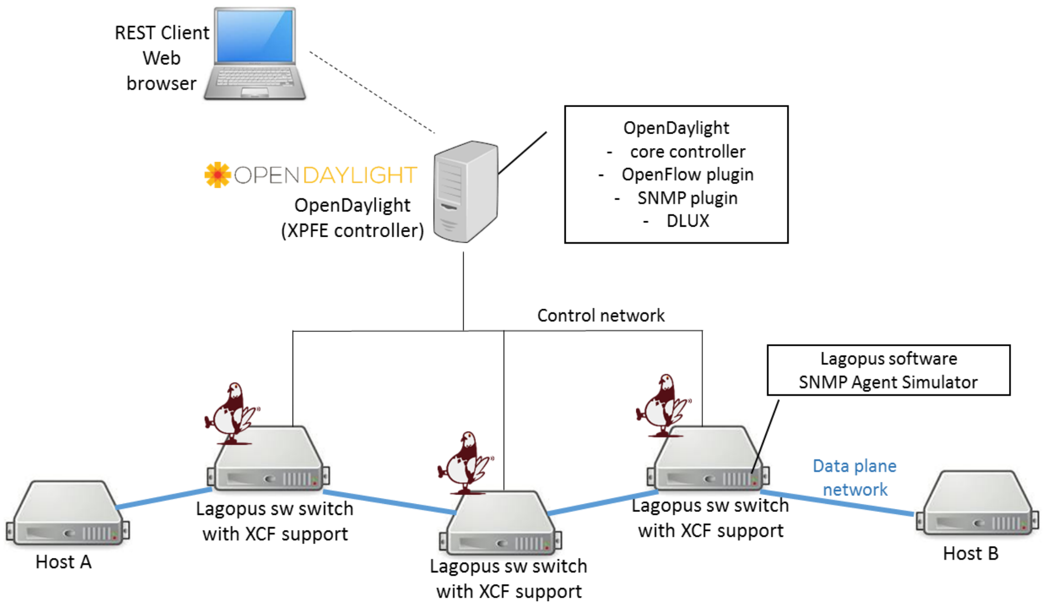
	messages. The controller computes the correct topology, which is shown in the web GUI.	
2.	**Description:** using the REST APIs of the Path Provisioning service, create network paths associated to different tenants. Use the controller GUI to visualize the paths and the flows. **Expected Results:** the network paths are correctly created and the flows are available in the network nodes (see NXW_XCI_06 for further details about path provisioning).	
3.	**Description:** emulate traffic between the hosts using iPerf[15] to simulate traffic associated to the created paths. Verify power consumption values for the network nodes in the topology. **Expected Results:** power consumption values available in the node elements retrieved from the topology service.	
4.	**Description:** use the REST APIs of the power consumption analytics application to retrieve: - The power consumption of the global physical infrastructure - The power consumption associated to resources consumed by each tenant Use the web GUI of the power consumption analytics applications to visualize the graphs of the power consumption data. **Expected Results:** retrieval of valid power consumption data for physical infrastructure and virtual infrastructures (i.e. composed by the network paths associated to a given tenant).	

12.1.5. Modification of XPFE operational status

Test Card #	NXW_XCI_05	Execution Status	Planned
Test Name	Modification of XPFE operational status		
Objectives	Verify the mechanisms to change the operational status of XPFE devices via NBI of the SDN controller.		
Related Use Cases	Dense urban information society		
Responsible	NXW		
Related Test	N.A.		

[15] https://iperf.fr/

Cards	
Additional Comments	SNMP agents on network nodes simulated via SNMP Agent Simulator[16]. The reference SNMP MIBs are IANAPowerStateSet-MIB, ENERGY-OBJECT-MIB and POWER-ATTRIBUTES-MIB specified in [19]. The change of the status is performed through a SET on the eoPowerAdminState MIB object.

SUT and topology	
SUT	OpenDaylight controller with the following software modules enabled: • Core controller services • L2 switch • OpenFlow plugin • SNMP plugin • DLUX (web GUI)
Test environment topology  REST Client Web browser OPENDAYLIGHT OpenDaylight (XPFE controller) OpenDaylight - core controller - OpenFlow plugin - SNMP plugin - DLUX Control network Lagopus software SNMP Agent Simulator Data plane network Host A Lagopus sw switch with XCF support Lagopus sw switch with XCF support Lagopus sw switch with XCF support Host B	
External components	• REST client to interact with the controller via REST API + web browser to visualize the controller web GUI. • Network infrastructure with XPFEs in the data plane. This network infrastructure is based on machines with Lagopus software switches where the SNMP Agent Simulator is also installed.

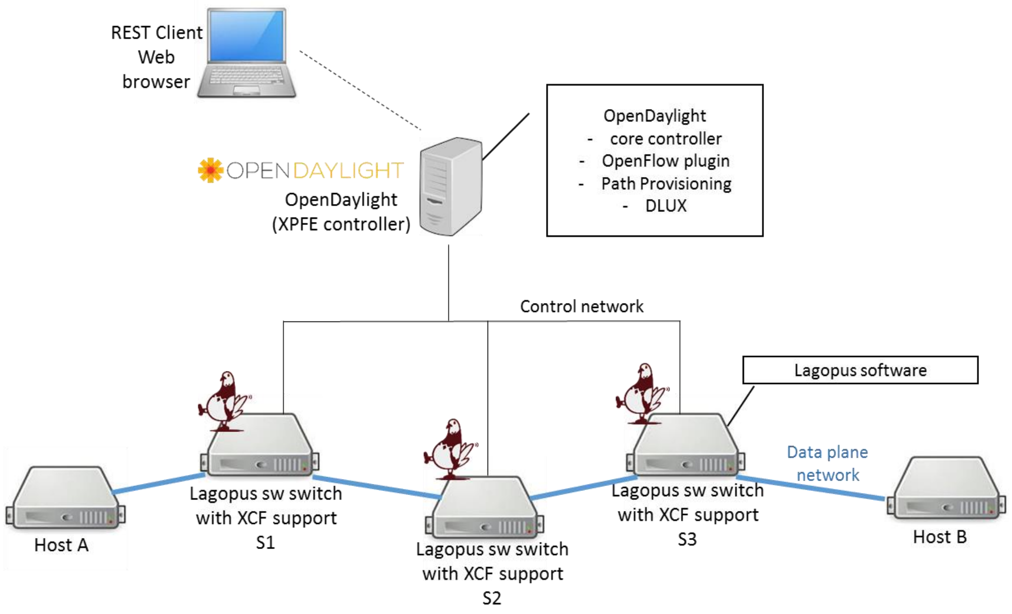
[16] http://snmpsim.sourceforge.net/

Test description		
Step #	Step description and expected results	Status
1.	**Description:** start OpenDaylight in the SDN controller and activate the required software modules enabling the related karaf features. In the data plane nodes, start Lagopus software and the SNMP Agent Simulator. **Expected Results:** OpenDaylight software up and running in the SDN controller node with SNMP manager, extended inventory and topology. SNMP agents up and running in network nodes.	
2.	**Description:** using Wireshark verify the exchange of SNMP messages from the controller to the network nodes. Using the REST APIs, retrieve the list and status of the network nodes by the extended Topology. **Expected Results:** the status of all the nodes is active. SNMP messages to get the power status are properly exchanged through the SDN controller SBI. The messages are GET on the *eoPowerOperState* MIB object.	
3.	**Description:** using the REST APIs, sends a command to put a network node in sleeping mode. Using Wireshark verify the exchange of SNMP messages from the controller to the network nodes. **Expected Results:** the controller sends an SNMP message with a SET on the *eoPowerAdminState* MIB object and value *emanSleep* (1028).	

12.1.6. Setup and termination of energy-efficient paths on XPFE networks

Test Card #	NXW_XCI_06	Execution Status	Planned
Test Name	Setup and termination of energy-efficient paths on XPFE networks		
Objectives	Verify the mechanisms to setup network paths associated to a given tenant minimizing the energy consumption of the whole power consumption at the physical infrastructure. Verify the mechanisms to terminate an existing network path.		
Related Use Cases	Dense urban information society		
Responsible	NXW, POLITO		

Related Test Cards	NXW_XCI_061
Additional Comments	This test card focuses only on the functional aspects of the procedures, in terms of interaction between software components and generation of the suitable commands to configure the XPFE forwarding behaviour. The performance evaluation of the path computation algorithm is out of scope, since it is covered through simulation analysis described in D4.1 [2]. In this test, we assume that the path returned by the computation algorithm is the optimal one.

SUT and topology	
SUT	OpenDaylight controller with the following software modules enabled: • Core controller services • L2 switch • OpenFlow plugin • Path provisioning • Path computation • DLUX (web GUI)
Test environment topology	 REST Client Web browser OPENDAYLIGHT OpenDaylight (XPFE controller) OpenDaylight - core controller - OpenFlow plugin - Path Provisioning - DLUX Control network Lagopus software Data plane network Host A Lagopus sw switch with XCF support S1 Lagopus sw switch with XCF support S2 Lagopus sw switch with XCF support S3 Host B
External components	• REST client to interact with the controller via REST API + web browser to visualize the controller web GUI. • Network infrastructure with XPFEs in the data plane. This network infrastructure is based on machines with Lagopus software switches.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** start OpenDaylight in the SDN controller and activate the required software modules enabling the related karaf features. In the data plane nodes, start the Lagopus. **Expected Results:** OpenDaylight software up and running in the SDN controller node. Network topology correctly loaded in the controller and visible on the web GUI.	
2.	**Description:** using the REST APIs of the Path Provisioning service create a new network path associated to *tenant 1* from host A to host B. Using Wireshark verify the exchange of OpenFlow messages from the controller to the network nodes. **Expected Results:** the Path Computation service computes a path that includes S1, S2, S3. The Path Provisioning service maps the path in suitable OpenFlow commands (specific mapping of OpenFlow Actions to XPFE flow tables still to be decided). In particular, for the host A → host B direction: - S1: PUSH_PBB + PUSH_VLAN + forwarding action - S2: forwarding action - S3: POP_PBB + POP_VLAN + forwarding action and similarly for the opposite direction (push on S3 and pop on S1). PBB and VLANs are built to refer tenant 1 as defined in the XCF format. The associated OpenFlow FlowMod messages are correctly exchanged between the controller and the network nodes. The flow entries are available in the network nodes and are visible on the controller web GUI. Using the REST API of the Path Provisioning service, the details of the path are correctly returned.	
3.	**Description:** perform again step 2 for *tenant 2.* **Expected Results:** a similar path is created for *tenant 2*. Traffic for the two tenants can be exchanged between the two hosts.	
4.	**Description:** using the REST APIs of the Path Provisioning service terminate the first network path associated to *tenant 1*. Using Wireshark verify the exchange of OpenFlow messages from the controller to the network nodes. **Expected Results:** the OpenFlow messages to remove the flow entries associated to the path are correctly exchanged between the controller and	

	the network nodes. The flow entries have been removed from the network nodes and are no more visible on the controller web GUI. Using the REST API of the Path Provisioning service, the path is returned with "status = terminated".	

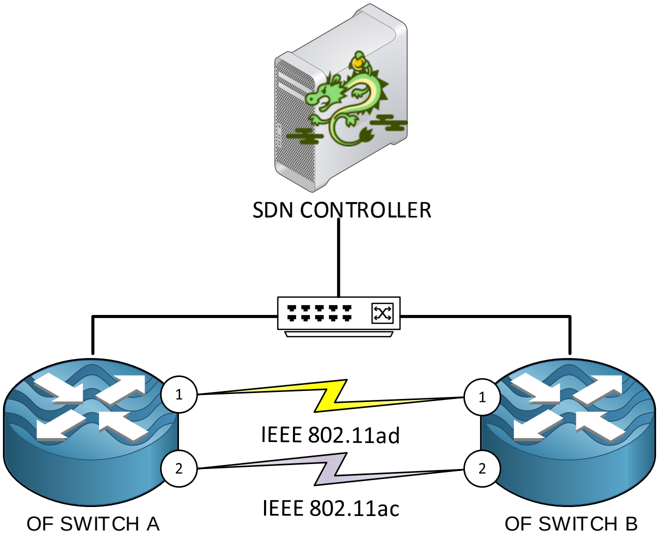
12.2. SDN Controller for mmWave and WiFi mesh technology

This section provides details on the configuration and monitoring of WiFi and mmWave elements summarized in Section 9.3.2.2.

12.2.1. Topology detection

Test Card #	CTTC-mmWave-F01-T001	Execution Status	Planned
Test Name	Topology detection		
Objectives	Detection of the switches conforming the network topology together with the different equipped interfaces		
Related Use Cases	Dense urban information society		
Responsible	CTTC		
Related Test Cards	N/A (not applicable)		

SUT and topology	
SUT	Two Backhaul nodes equipped with different wireless technologies and the mmWave SDN controller
Test environment topology	

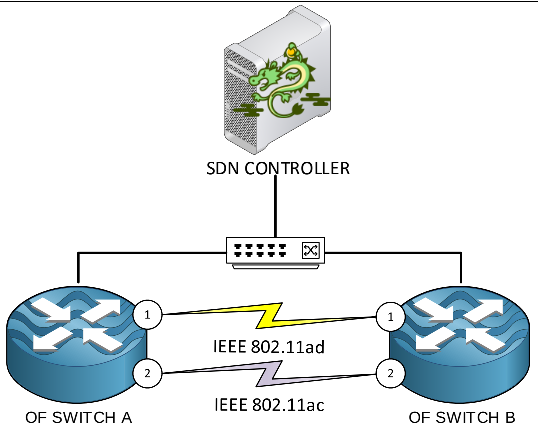
	 SDN CONTROLLER 1 1 IEEE 802.11ad 2 2 IEEE 802.11ac OF SWITCH A OF SWITCH B
External components	Ethernet switch allowing a wired control plane.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure manually two wireless backhaul links between two wireless backhaul nodes: one for IEEE 802.11ad (primary link) and the other for IEEE 802.11ac (redundant link). Start OF wireless switch at each node. **Expected Results:** link connection verified by means of ping command.	
2.	**Description:** run SDN controller and detect OF wireless switches in the network. **Expected Results:** the SDN controller reports the detection of the wireless switches and links and their properties through OpenFlow. MAC address and interface name must match the actual ones. **Comments:** from the set of properties reported by OpenFlow message, we are only interested in MAC addresses and interface name.	

12.2.2. Interface state monitoring and reconfiguration

Test Card #	CTTC-mmWave-F01-T002	Execution Status	Planned
Test Name	Interface state monitoring and reconfiguration		
Objectives	Basic validation of a software interface based on REST to interact with backhaul nodes.		

	Monitoring/configuration of the state of each of the interface (ON/OFF) present in the wireless node.
Related Use Cases	Dense urban information society
Responsible	CTTC
Related Test Cards	CTTC-mmWave-F01-T001
Additional Comments	The interface state (e.g., link up or down) could be monitored with the OpenFlow Switch specification; however, OpenFlow Switch specification does not provide the possibility to configure the interface status. Because of this, we propose to use a REST-based interface for the configuration of the wireless interfaces.

SUT and topology	
SUT	A backhaul node equipped with different wireless technologies offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller.
Test environment topology	 SDN CONTROLLER 1 1 IEEE 802.11ad 2 2 IEEE 802.11ac OF SWITCH A OF SWITCH B
External components	Ethernet switch allowing a wired control plane.

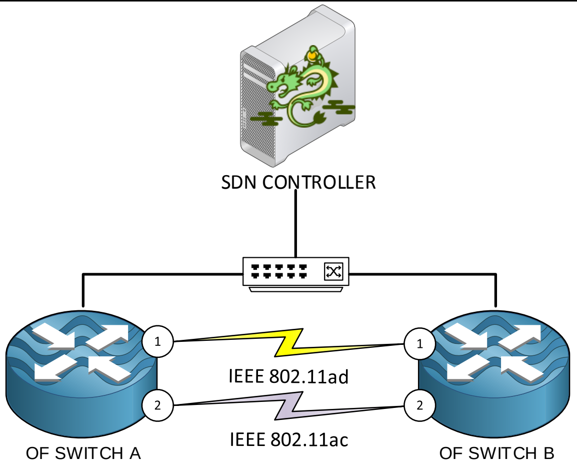
Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure manually two wireless interfaces at one backhaul node, start software switch at node and run the SDN controller. **Expected Results:** SDN controller detects the switch and provide information about ON state of switch interfaces.	
2.	**Description:** the SDN controller makes GET query of the state of each	

	of the interfaces in the switch. **Expected Results:** the query to the SBI REST API informs the SDN controller about the ON state of the different interfaces.	
3.	**Description:** the SDN controller makes a PUT query to switch off one of the interfaces of the switch and a GET query to verify that this change has been executed. **Expected Results:** one interface of the switch is deactivated. **Comments:** in addition to this, the SDN controller will receive a OF message from the switch informing about the change in the state of the interface. This event will be relevant for a subsequent test named CTTC-mmWave-F01-T005.	

12.2.3. Interface technology detection

Test Card #	CTTC-mmWave-F01-T003	Execution Status	Pending/Pass
Test Name	Interface technology detection		
Objectives	Basic validation of a software interface based on REST to interact with backhaul nodes. Monitoring of the wireless technologies present in the node.		
Related Use Cases	Dense urban information society.		
Responsible	CTTC		
Related Test Cards	CTTC-mmWave-F01-T002		
Additional Comments	Due to the current node possibilities, the technology detection is done by querying the carrier frequency of each backhaul node interface. The carrier frequency of IEEE 802.11ac technology can be in the 2.4/5 GHz band while the carrier frequency of IEEE 802.11ad is in the 60GHz band (V-Band)		

SUT and topology	
SUT	A backhaul node equipped with different wireless technologies offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller

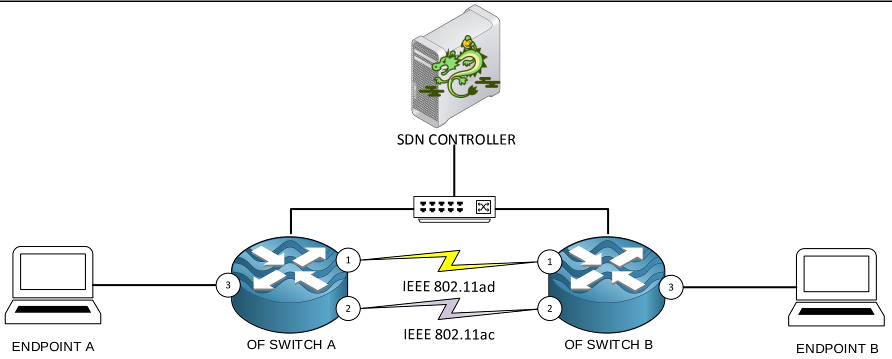
Test environment topology	 SDN CONTROLLER 1 IEEE 802.11ad 2 IEEE 802.11ac 1 2 OF SWITCH A OF SWITCH B
External components	Ethernet switch allowing a wired control plane.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure manually two wireless interfaces at one backhaul node, start software switch at node and run the SDN controller. **Expected Results:** SDN controller detects the switch and provide information about the switch interfaces.	
2.	**Description:** the SDN controller makes GET queries to gather the carrier frequency of each of the interfaces in the switch. **Expected Results:** query triggered from the SDN controller through REST to gather the carrier frequency of each wireless interface, hence discriminating the technology at each interface. **Comments:** identifier assigned to the interface depending on the technology in SDN controller DB.	

12.2.4. Route establishment according to path priority

Test Card #	CTTC-mmWave-F01-T004	Execution Status	Planned
Test Name	Route establishment according to path priority		
Objectives	Establishment of a traffic connection between endpoints using prioritized links		
Related Use Cases	Dense urban information society		
Responsible	CTTC		

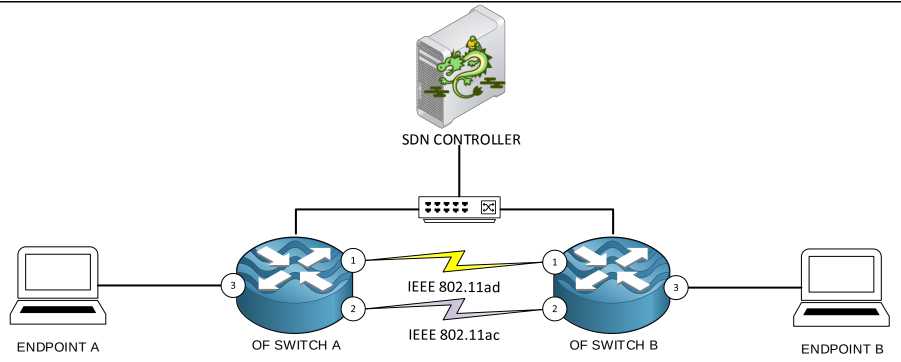
Related Test Cards	CTTC-mmWave-F01-T003
Additional Comments	mmWave (IEEE 802.11ad) interface/link is prioritized over Wi-Fi interface/link (IEEE 802.11ac)

SUT and topology	
SUT	Two backhaul nodes equipped with different wireless technologies offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller.
Test environment topology	 SDN CONTROLLER 1 3 2 IEEE 802.11ad IEEE 802.11ac 1 2 3 ENDPOINT A OF SWITCH A OF SWITCH B ENDPOINT B
External components	Ethernet switch allowing a wired control plane. Two computer/VMs connected to the switches acting as traffic generators.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure manually two wireless interfaces at two different backhaul nodes, start software switch at each node and run the SDN controller. **Expected Results:** SDN controller detects the switches and provides information about the switch interfaces/links. It detects the technologies present in the node by assigning the corresponding identifier in order to prioritize traffic through mmWave (IEEE 802.11ad) link.	
2.	**Description:** start a flow from EndpointA to EndpointB. **Expected Results:** flow provisioning through the mmWave link because this link has higher priority over the IEEE 802.11ac WiFi link. **Comments:** verification of the use of mmWave through Wireshark.	

12.2.5. Link failure and restoration

Test Card #	CTTC-mmWave-F01-T005	Execution Status	Planned
Test Name	Link failure and restoration.		
Objectives	Establishment of a traffic connection between endpoints using prioritized links. Detection of link failure and restoration of connection.		
Related Use Cases	Dense urban information society.		
Responsible	CTTC		
Related Test Cards	CTTC-mmWave-F01-T002, CTTC-mmWave-F01-T003, CTTC-mmWave-F01-T004		

SUT and topology	
SUT	Two backhaul nodes equipped with different wireless technologies offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller.
Test environment topology	 SDN CONTROLLER ENDPOINT A 3 1 2 OF SWITCH A IEEE 802.11ad IEEE 802.11ac 1 2 3 OF SWITCH B ENDPOINT B
External components	Ethernet switch allowing a wired control plane. Two VMs connected to the switches acting as traffic generators.

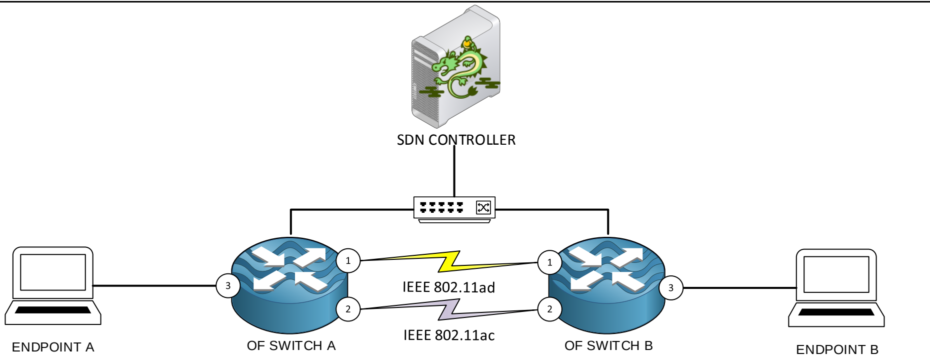
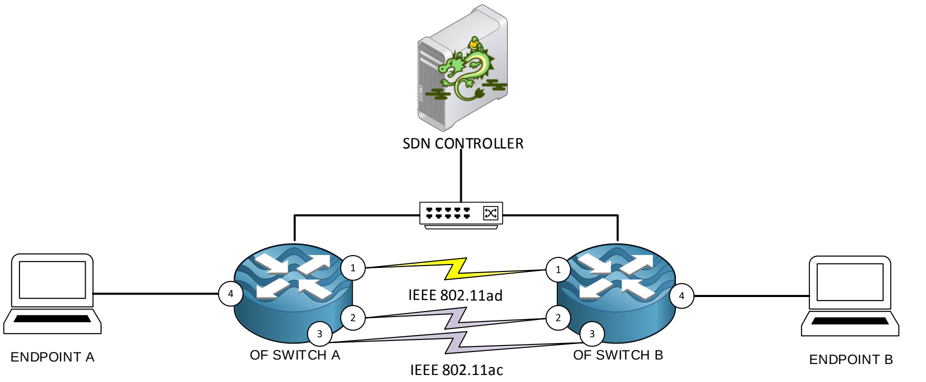
Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure manually two wireless interfaces at two different backhaul nodes, start software switch at each node and run the SDN controller. **Expected Results:** SDN controller detects the switches and provides information about the switch interfaces/links. It also detects the technologies present in the node and assigns the corresponding weights	

	prioritizing mmWave (IEEE 802.11ad) link.	
2.	**Description:** start a flow from EndpointA to EndpointB. **Expected Results:** the flow is provisioned through the mmWave link because this link has higher priority over the IEEE 802.11ac link. **Comments:** the verification of the use of mmWave link validated by means of Wireshark.	
3.	**Description:** manually switch down the mmWave interface of switch B. We have to switch down the mmWave interface of the node acting as station and not as the PBSS control point (PCP). **Expected Results:** the SDN controller detects the link failure, the flow interrupts and it is re-established through the IEEE 802.11ac link. **Comments:** the verification of the use of IEEE 802.11ac link is done by means of Wireshark software.	

12.2.6. Failure recovery and route verification

Test Card #	CTTC-mmWave-F01-T006	Execution Status	Planned
Test Name	Failure recovery and route verification		
Objectives	Establishment of a traffic connection between endpoints using prioritized links Detection of link failure and restoration of connection Recovery from link failure, route verification		
Related Use Cases	Dense urban information society		
Responsible	CTTC		
Related Test Cards	CTTC-mmWave-F01-T005		

SUT and topology	
SUT	Two backhaul nodes equipped with different wireless technologies offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller.
Test environment topology	We divide the Test environment topology in two specific different variants: a) two heterogeneous wireless backhaul interfaces

	 SDN CONTROLLER ENDPOINT A 3 1 2 OF SWITCH A IEEE 802.11ad IEEE 802.11ac 1 2 3 OF SWITCH B ENDPOINT B b) three heterogeneous wireless backhaul interfaces (i.e., an 802.11ad link and two 802.11ac links)  SDN CONTROLLER ENDPOINT A 4 1 2 3 OF SWITCH A IEEE 802.11ad IEEE 802.11ac 1 2 3 4 OF SWITCH B ENDPOINT B
External components	An Ethernet switch allowing a wired control plane and two computer/VMs connected to the switches acting as traffic generators.

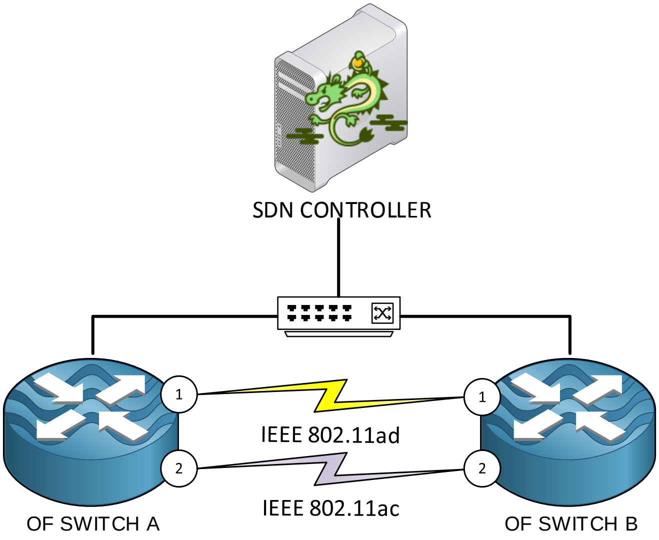
Test description		
Step #	Step description and expected results	Status
1.a	**Description:** in variant a), configure manually two wireless interfaces at two different backhaul nodes, start software switch at each node and run the SDN controller. **Expected Results:** SDN controller detects the switches and provides information about the switch interfaces/links. It also detects the technologies present in the node and assigns the corresponding weights prioritizing mmWave (IEEE 802.11ad) link.	
1.b	**Description:** in variant b), configure manually two wireless interfaces at two different backhaul nodes, start software switch at each node and run the SDN controller. **Expected Results:** SDN controller detects the switches and provides information about the switch interfaces/links. It also detects the technologies present in the node and assigns the corresponding weights prioritizing mmWave (IEEE 802.11ad) link.	

2.	**Description:** start a flow from EndpointA to EndpointB. **Expected Results:** the flow is established through the mmWave link because this link has priority over the IEEE 802.11ac link/s. **Comments:** the verification of the use of mmWave link is done by means of Wireshark software.	
3.	**Description:** manually switch down the mmWave interface of switch B. We have to switch down the mmWave interface of the node acting as station and not as the PBSS control point (PCP). **Expected Results:** the SDN controller detects the link failure, the flow interrupts and it is re-established through the IEEE 802.11ac link. **Comments:** the verification of the use of IEEE 802.11ac is done by means of Wireshark software.	
4. a)	**Description:** in variant a) switch on the mmWave interface of switch B. **Expected Results:** the SDN controller detects the link recovery and the flow is re-established through the mmWave link because this link has higher priority over the IEEE 802.11ac link. **Comments:** The verification of the use of mmWave link is done by means of Wireshark software.	
4. b)	**Description:** in variant b) we switch down the free WiFi interface of switch B. **Expected Results:** the SDN controller detects the link failure and the flow continues through the link established in step 3.	
5. b)	**Description:** in variant b) we switch up the IEEE 802.11ac interface switched off at switch B. **Expected Results:** the SDN controller detects the link recovery but the link is not re-routed through this link because this possible path presents the same weight as the current established one.	

12.2.7. Interface monitoring

Test Card #	CTTC-mmWave-F02-T001	Execution Status	Planned
Test Name	Interface monitoring		
Objectives	Get the configuration information of the interfaces of the switches in the network topology.		
Related Use Cases	Dense urban information society.		
Responsible	CTTC		

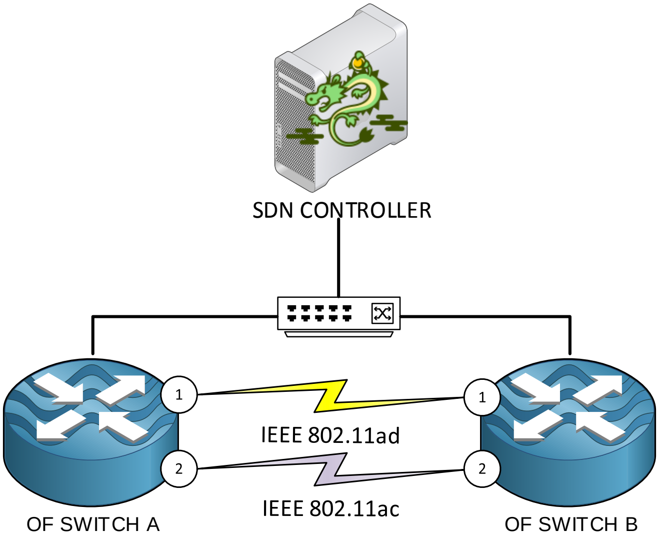
Related Test Cards	N/A (not applicable).
Additional Comments	The interface parameters information includes the interface status (ON/OFF), transmission power, transmission rate, channel and frequency. The SDN Controller will be able to get the values of these parameters as a whole or each of them independently.

SUT and topology	
SUT	One Backhaul node equipped with different wireless technologies and the mmWave SDN controller
Test environment topology	 SDN CONTROLLER 1 1 IEEE 802.11ad 2 2 IEEE 802.11ac OF SWITCH A OF SWITCH B
External components	Ethernet switch allowing a wired control plane

Test description		
Step #	Step description and expected results	Status
1.	**Description:** manually configure two wireless interfaces at one backhaul node and run the SDN controller. One interface with IEEE 802.11ac technology and the other one with IEEE 802.11ad. **Expected Results:** SDN controller detects the switch and provides information about the ON state of the switch interfaces	
2.	**Description:** trigger REST GET query to get the configuration information of each of the interfaces in the switch. **Expected Results:** the query to the SDN controller informs of the interfaces configuration; all configuration values must match the configuration set on step #1.	

12.2.8. Interface configuration

Test Card #	CTTC-mmWave-F02-T002	Execution Status	Planned
Test Name	Interface configuration.		
Objectives	Configure the interfaces of the switches in the network topology.		
Related Use Cases	Dense urban information society.		
Responsible	CTTC		
Related Test Cards	CTTC-mmWave-F02-T001		

SUT and topology	
SUT	One backhaul node equipped with different wireless technologies and the mmWave SDN controller.
Test environment topology	 SDN CONTROLLER 1 1 IEEE 802.11ad 2 2 IEEE 802.11ac OF SWITCH A OF SWITCH B
External components	Ethernet switch allowing a wired control plane

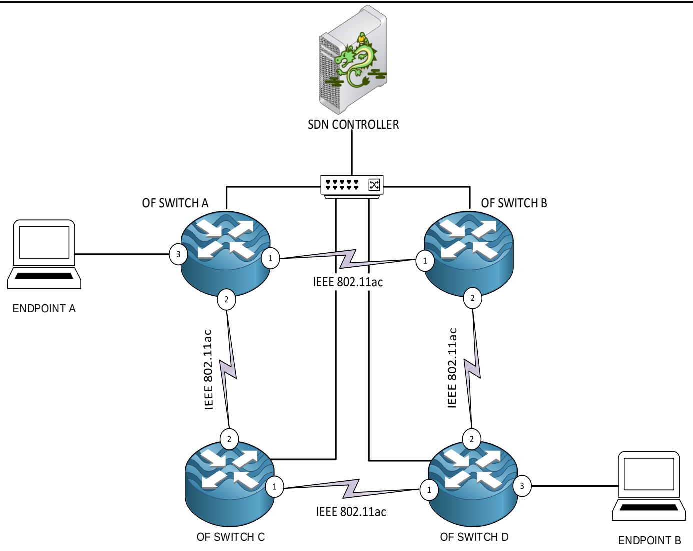
Test description		
Step #	Step description and expected results	Status
1.	**Description:** manually configure two wireless interfaces at one backhaul node and run the SDN controller. One interface with IEEE 802.11ac technology and the other one with IEEE 802.11ad. **Expected Results:** SDN controller detects the switch and provides	

	information about ON state of switch interfaces	
2.	**Description:** trigger REST PUT query to modify the configuration of each of the interfaces in the switch. **Expected Results:** manually check the interface configuration in the node All configuration values must match the configuration values in the PUT query. **Comments:** None.	

12.2.9. End-to-end route validation

Test Card #	CTTC-mmWave-F03-T001	Execution Status	Planned
Test Name	End-to-end route validation.		
Objectives	Traffic connection establishment between endpoints selecting one out of several possible end-to-end paths.		
Related Use Cases	Dense urban information society.		
Responsible	CTTC		
Related Test Cards	N.A.		
Additional Comments	In this case, all links between adjacent nodes are IEEE 802.11ac, hence all the possible paths have the same link weight.		

SUT and topology	
SUT	Four backhaul nodes equipped with several wireless interfaces offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller

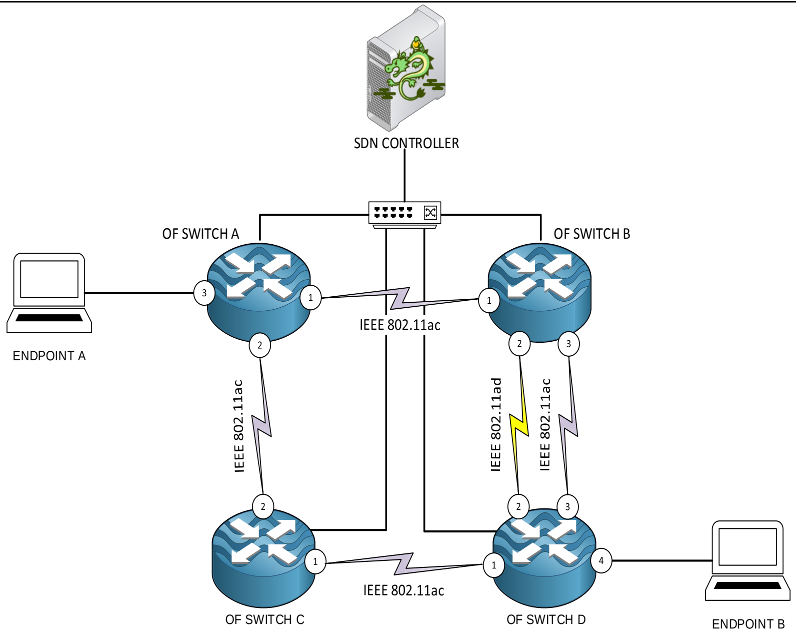
Test environment topology	 SDN CONTROLLER OF SWITCH A OF SWITCH B ENDPOINT A 3 1 2 IEEE 802.11ac 1 2 IEEE 802.11ac IEEE 802.11ac 2 2 1 1 3 IEEE 802.11ac OF SWITCH C OF SWITCH D ENDPOINT B
External components	Ethernet switch allowing a wired control plane Two computer/VMs connected to the switches acting as traffic generators
Test description	

Step #	Step description and expected results	Status
1.	**Description:** configure manually two wireless interfaces at each backhauling node to setup the network connections according to the test environment, start software switch at each node and run the SDN controller. **Expected Results:** SDN controller detects the switches and provides information about the switch interfaces/links. It also detects the technologies present in the node and assigns the corresponding weights.	
2.	**Description:** start a flow from EndpointA to EndpointB. **Expected Results:** the flow is established through one of the possible paths. In this case, the path selection is based on following a shortest path approach in terms of number of hops. In case there are several paths with an equivalent number of hops to reach the destination, a random path is chosen out of those with an equivalent number of hops to reach the destination.	

12.2.10. End-to-end route validation with link priority based on technology

Test Card #	CTTC-mmWave-F03-T002	Execution Status	Planned
Test Name	End-to-end route validation with link priority based on technology.		
Objectives	Establishment of a traffic connection between endpoints selecting the path with less accumulated weight.		

Related Use Cases	Dense urban information society.
Responsible	CTTC
Related Test Cards	N.A.
Additional Comments	In this case, there is one or several nodes interconnected with different technologies, namely IEEE 802.11ac and IEEE 802.11ad.

SUT and topology	
SUT	Four backhaul nodes equipped with several wireless interfaces offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller.
Test environment topology	 SDN CONTROLLER OF SWITCH A OF SWITCH B ENDPOINT A IEEE 802.11ac IEEE 802.11ac IEEE 802.11ad IEEE 802.11ac IEEE 802.11ac OF SWITCH C OF SWITCH D ENDPOINT B 1 2 3 4
External components	Ethernet switch allowing a wired control plane. Two VMs connected to the switches acting as traffic generators.

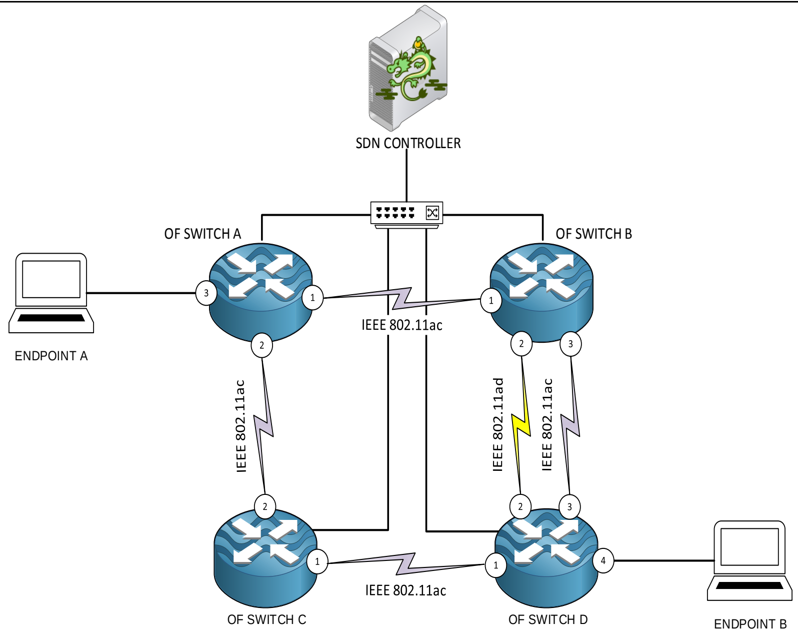
Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure manually the wireless interfaces at each backhauling node to setup the network connections according to the test environment, start software switch at each node, and run the SDN controller. **Expected Results:** SDN controller detects the switches and provides	

	information about the switch interfaces/links. It also detects the technologies present in the nodes and assigns the corresponding weights on a per technology basis.	
2.	**Description:** start a flow from EndpointA to EndpointB. **Expected Results:** the flow is established through the path with the less accumulated weight, hence using the IEEE 802.11ad connection.	

12.2.11. End-to-end route validation with link priority based on link characteristics

Test Card #	CTTC-mmWave-F03-T003	Execution Status	Planned
Test Name	End-to-end route validation with link priority based on link characteristics		
Objectives	Establishment of a traffic connection between endpoints selecting the path with less accumulated weight. Link weight is assigned in base on link characteristics: transmission rate, that is, the physical link rate reported by the hardware device.		
Related Use Cases	Dense urban information society.		
Responsible	CTTC		
Related Test Cards	N.A.		

SUT and topology	
SUT	Four backhaul nodes equipped with several wireless interfaces offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller

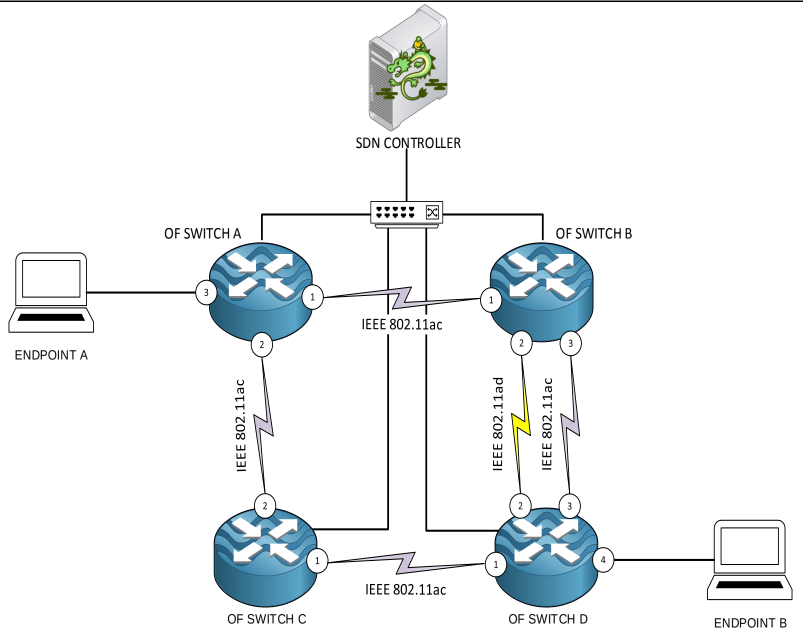
Test environment topology	 SDN CONTROLLER OF SWITCH A OF SWITCH B ENDPOINT A IEEE 802.11ac IEEE 802.11ac IEEE 802.11ad IEEE 802.11ac IEEE 802.11ac OF SWITCH C OF SWITCH D ENDPOINT B 1 2 3 4
External components	Ethernet switch allowing a wired control plane and two computer/VMs connected to the switches acting as traffic generators.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure manually the wireless interfaces at each backhauling node to setup the network connections according to the test environment, start software switch at each node and run the SDN controller. In this case, nodes between adjacent nodes are configured with different transmission rate. **Expected Results:** SDN controller detects the switches and provides information about the switch interfaces/links. It also detects the wireless link characteristics in the nodes and assigns the corresponding weights on a per link basis.	
2.	**Description:** start a flow from EndpointA to EndpointB. **Expected Results:** the flow is established through the path with the less accumulated weight, hence using links with bigger transmission rate.	

12.2.12. End-to-end route validation with link priority based on link characteristic and link occupation

Test Card #	CTTC-mmWave-F03-T004	Execution Status	Planned
Test Name	End-to-end route validation with link priority based link characteristics and		

	link occupation
Objectives	Periodically monitoring of link statistics by means of OpenFlow messages to derive a routing metric derived on the amount of bytes transmitted by a port. Establishment of successive traffic connections between endpoints selecting the path with less accumulated weight. Link weight is assigned in base on link characteristics: transmission rate[17] and the amount of bytes transmitted by a port, which is obtained through OpenFlow OF_PORT_STATS message.
Related Use Cases	Dense urban information society
Responsible	CTTC
Related Test Cards	CTTC-mmWave-F03-T003

SUT and topology	
SUT	Four backhaul nodes equipped with several wireless interfaces offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller
Test environment topology	 SDN CONTROLLER OF SWITCH A OF SWITCH B ENDPOINT A IEEE 802.11ac IEEE 802.11ac IEEE 802.11ad IEEE 802.11ac IEEE 802.11ac OF SWITCH C OF SWITCH D ENDPOINT B 3 1 2 1 2 3 2 1 2 3 1 4
External components	Ethernet switch allowing a wired control plane. Two computer/VMs connected to the switches acting as traffic generators

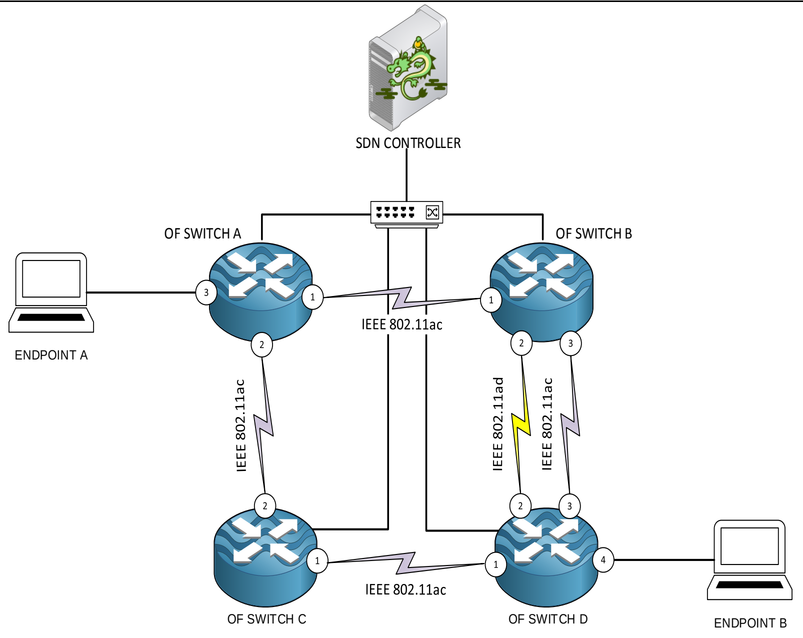
[17] We refer the transmission rate to the physical link rate reported by the hardware device.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure manually the wireless interfaces at each backhauling node to setup the network connections according to the test environment, start software switch at each node and run the SDN controller. In this case, nodes between adjacent nodes are configured with different transmission rate to force different link weights. **Expected Results:** SDN controller detects the switches and provides information about the switch interfaces/links. It also detects the wireless link characteristics in the nodes and assigns the corresponding weights on a per link basis.	
2.	**Description:** start a flow from EndpointA to EndpointB. **Expected Results:** the flow is established through the path with the less accumulated weight, hence using links with bigger transmission rate because no other flow is in the network. In this case, the path will contain the mmWave link.	
3.	**Description:** check the periodical reception and update of statistics of each port and of each switch in the network. **Expected Results:** the SDN controller receives periodical OpenFlow multipart messages from the switches reporting statistics of the port utilization. This information is stored at the SDN controller to monitor the utilization of the port during the periodical reporting time.	
4.	**Description:** start another flow from EndpointA to EndpointB with bigger required data rate than the previous flow. **Expected Results:** the new flow is established through the path with the less accumulated weight and less traffic utilization. **Comments:** The path followed by new flow will try to follow a disjoint route since the alternative path has not registered any traffic.	
5.	**Description:** start a third flow from EndpointA to EndpointB with bigger required data rate than the previous. **Expected Results:** the new flow is established through the path with the less accumulated weight weighted by the traffic utilization. **Comments:** The path followed by the new flow will use the path found in step #2 but using the IEEE 802.11ac link instead.	

12.2.13. End-to-end route validation and link failure recovery

Test Card #	CTTC-mmWave-F03-T005	Execution Status	Planned

Test Name	End-to-end route validation and link failure recovery
Objectives	Establishment of successive traffic connections between endpoints selecting the path with less accumulated weight. Link weight is assigned in base on link characteristics: transmission rate and the amount of bytes transmitted by a port, which is obtained through OpenFlow OF_PORT_STATS message. Storing state information of active flows in network. Route re-computation for all traffic flows affected by a link failure.
Related Use Cases	Dense urban information society.
Responsible	CTTC
Related Test Cards	N.A.

SUT and topology	
SUT	Four backhaul nodes equipped with several wireless interfaces offering a REST API interface for SBI monitoring/configuration and the mmWave SDN controller
Test environment topology	 SDN CONTROLLER OF SWITCH A OF SWITCH B ENDPOINT A IEEE 802.11ac IEEE 802.11ac IEEE 802.11ad IEEE 802.11ac IEEE 802.11ac OF SWITCH C OF SWITCH D ENDPOINT B 1 2 3 4
External components	Ethernet switch allowing a wired control plane. Two computer/VMs connected to the switches acting as traffic generators.

Test description

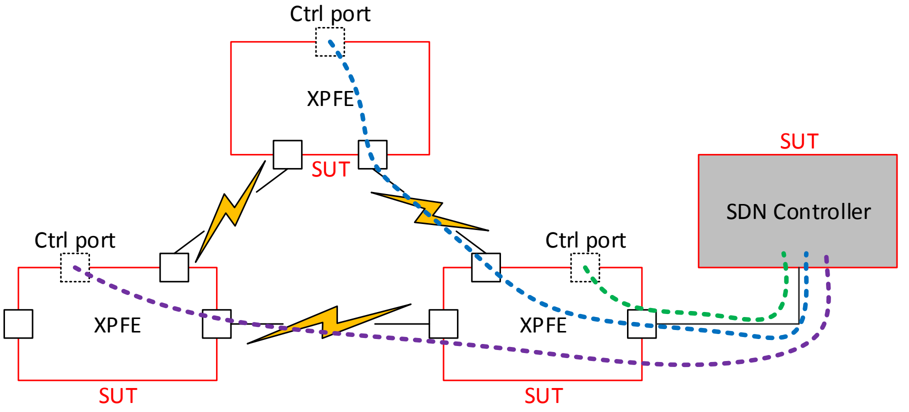
Step #	Step description and expected results	Status
1.	**Description:** configure manually the wireless interfaces at each backhauling node to setup the network connections according to the test environment, start software switch at each node and run the SDN controller. In this case, nodes can be configured with different transmission rates. **Expected Results:** SDN controller detects the switches and provides information about the switch interfaces/links. It also detects the wireless link characteristics in the nodes and assigns the corresponding weights on a per link basis.	
2.	**Description:** start a flow from EndpointA to EndpointB. **Expected Results:** the flow is established through the path with the less accumulated weight, hence using links with higher transmission rate. In this case, the path will contain the mmWave link.	
3.	**Description:** check the periodical reception and update of statistics of each port and of each switch in the network. **Expected Results:** the SDN controller receives periodical OpenFlow multipart messages from the switches reporting statistics of the port utilization. This information is stored at the SDN controller to monitor the utilization of the port during the periodical reporting time.	
4.	**Description:** start a new flow from EndpointA to EndpointB. **Expected Results:** the new flow is established through the path with the less accumulated weight and less traffic utilization.	
5.	**Description:** manually switch down one of the links used by the second defined flow. **Expected Results:** the SDN controller detects the link failure, detects the affected flow and provides a new path for this flow to re-establish the flow. **Comments:** the new path will contain the IEEE 802.11ac link of the nodes connected by means of the mmWave link. Note that tests can be extended for more complex topologies possible with the current equipment configuring the data plane.	
6.	**Description:** manually switch up the link previously switched off **Expected Results**: the SDN controller detects the link failure but do not redirect any flow because they are not affected by the link recovery.	

12.3. SDN Controller for mmWave mesh technology

This section focuses on the impact the control plane has over the data plane, to infer and retrieve the topology of the mesh network from the SDN controller with the target of establishing the primary and fallback paths. The fact of having the fallback paths improve the delivery of packets in case of link failure and avoid losses of traffic.

12.3.1. mmWave mesh stability

Test Card #	IDCC_XCI_01	Execution Status	Planned
Test Name	mmWave mesh stability		
Objectives	Validate the stability of the mmWave mesh network in terms of network nodes association and connection with the network controller through in-band signaling.		
Related Use Cases	mmWave mesh		
Responsible	IDCC		
Related Test Cards	IDCC_XCI_02, IDCC_XCI_03		

SUT and topology	
SUT	Two and three XPFEs, SDN Controller
Test environment topology	 Ctrl port XPFE SUT SUT SDN Controller Ctrl port Ctrl port XPFE XPFE SUT SUT
External components	None

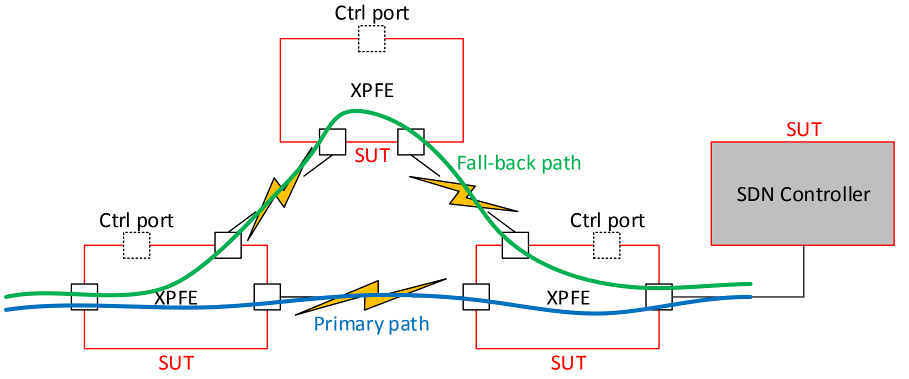
Test description

Step #	Step description and expected results	Status
1.	**Description:** align directional mmWave antennas. **Expected Results:** mmWave mesh nodes are able to associate as mesh neighbours.	
2.	**Description:** creation of in-band signalling channel. **Expected Results:** mmWave mesh nodes are able to connect to the network controller.	
3.	**Description:** creation of mmWave mesh network. **Expected Results:** mmWave mesh nodes maintains the connection with the network controller.	

12.3.2. Computation and configuration of paths within mmWave mesh network

Test Card #	IDCC_XCI_02	Execution Status	Planned
Test Name	Computation and configuration of paths within mmWave mesh network.		
Objectives	Validate the capability of the network controller to retrieve the mesh network topology, compute the primary and fallback paths, and configure the mesh nodes.		
Related Use Cases	mmWave mesh.		
Responsible	IDCC		
Related Test Cards	IDCC_XCI_02, IDCC_XCI_03		

SUT and topology	
SUT	Three XPFEs, SDN Controller

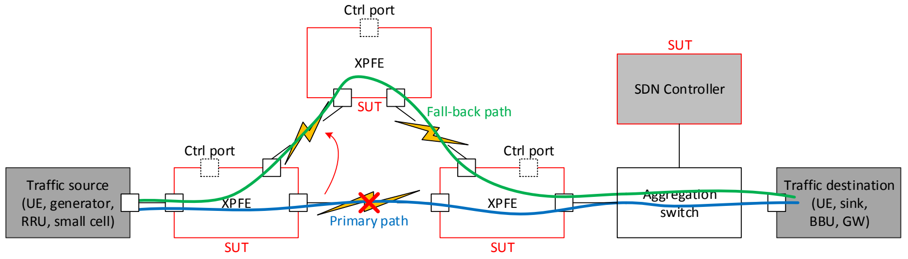
Test environment topology	 Ctrl port XPFE SUT Fall-back path SUT SDN Controller Ctrl port Ctrl port XPFE XPFE Primary path SUT SUT
External components	None

Test description		
Step #	Step description and expected results	Status
1.	**Description:** the mmWave nodes connect to the SDN Controller. **Expected Results:** the SDN Controller builds the mmWave mesh network topology.	
2.	**Description:** the SDN Controller computes the primary and fallback paths. **Expected Results:** the computed paths are the optimal ones.	
3.	**Description:** the SDN Controller configures the primary and fallback paths on the mmWave mesh nodes. **Expected Results:** correct configuration of the forwarding rules on the mmWave mesh nodes.	

12.3.3. Traffic recovery upon mmWave link failure

Test Card #	IDCC_XCI_03	Execution Status	Planned
Test Name	Connectivity recovery upon mmWave link failure.		
Objectives	Validate the capability of restoring the connectivity for both control and data planes.		
Related Use Cases	mmWave mesh.		
Responsible	IDCC		

Related Test Cards	IDCC_XCI_01, IDCC_XCI_02

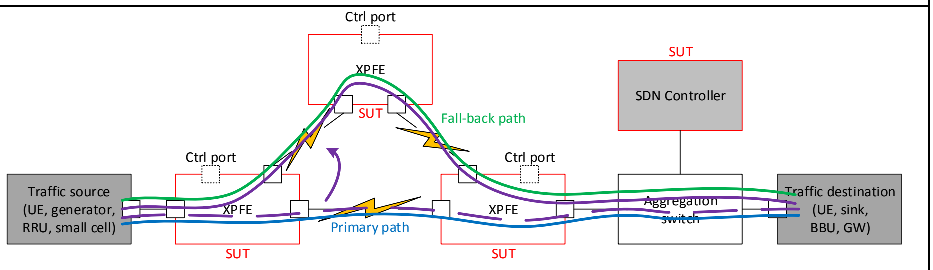
SUT and topology	
SUT	Three XPFEs, SDN Controller
Test environment topology	
External components	2 nodes (physical or logical) generating/consuming test traffic.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** a mmWave link on the primary path fails. **Expected Results:** XPFEs autonomously restore the connectivity on the fallback path.	
2.	**Description:** the SDN Controller reacts to the failure and re-computes and re-configures the primary and fallback paths. **Expected Results:** the computed paths are the optimal ones.	

12.3.4. Control plane impact on mmWave mesh network

Test Card #	IDCC_XCI_04	Execution Status	Planned
Test Name	Control plane impact on mmWave mesh network.		
Objectives	Assess the impact of the control plane on the data plane in terms of throughput, latency, jitter, packet loss.		
Related Use Cases	mmWave mesh.		
Responsible	IDCC		

Related Test Cards	IDCC_XCI_03

SUT and topology	
SUT	Three XPFEs, SDN Controller.
Test environment topology	 Ctrl port XPFE SUT Fall-back path SUT SDN Controller Ctrl port Ctrl port Traffic source (UE, generator, RRU, small cell) XPFE Primary path XPFE Aggregation switch Traffic destination (UE, sink, BBU, GW) SUT SUT
External components	2 nodes (physical or logical) generating/consuming test traffic

Test description		
Step #	Step description and expected results	Status
1.	**Description:** the SDN Controller continuously reconfigure the mesh network (e.g., moving one traffic flow from the primary path to the secondary path and vice versa) considering different traffic profiles and loads **Expected Results:** the data plane will be temporarily affected by this reconfiguration in terms of latency, jitter, and throughput	

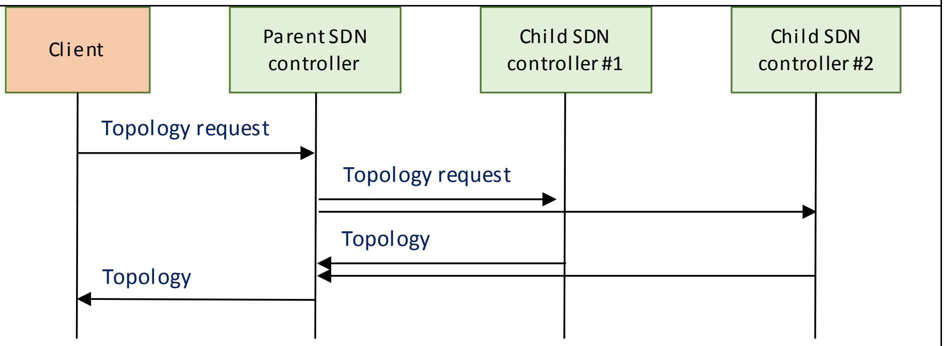
12.4. ABNO-based hierarchical SDN Controller

These tests are oriented to retrieve the multi-domain network topology from the underlying ABNO-based hierarchical SDN controller and validate the correctness of that retrieved topology, as covered in Section 9.3.2.4. This topology is used to compute the available paths and allocate the required resources. The parent SDN controller offers connectivity and connexion control services. This SDN controller exports web socket notifications that can be consumed by client applications.

12.4.1. Topology recovery

Test Card 1	CTTC-ABNO-T001	Execution Status	Planned

Test Name	Topology recovery
Objectives	To be able to retrieve the multi-domain network topology from the underlying SDN controller(s), using a YANG-based API such as COP topology service. Validate that the topology is correct, it includes the Service End Points and it can be used for path computation and resource reservation processes.
Related Use Cases	All use cases defined in D1.1 [3].
Responsible	CTTC
Related Test Cards	N/A (not applicable)
Additional Comments	This test involves a client that requests the topology from the parent SDN controller.

SUT and topology	
SUT	Parent SDN controller. All applications to be interconnected in a management network (IP reachability).
Test environment topology	 Client Parent SDN controller Child SDN controller #1 Child SDN controller #2 Topology request Topology request Topology Topology
External components	Basic management network IP reachability between functional entities. An (emulated or real) network topology retrieved by the parent SDN controller from the underlying SDN controller(s), and exported to client.

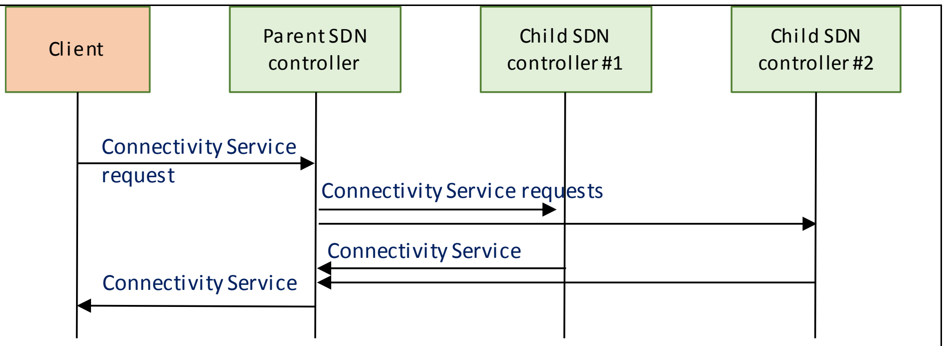
Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure a transport network topology in one or multiple domains, with one SDN controller or multiple ones allocated in a	

	hierarchy. **Expected Results:** it should be possible to consume parent SDN controller COP topology service and retrieve topological information	
2.	**Description:** Execute client request for the topology using the COP protocol. **Expected Results:** the client receives a multi-domain topology that is compliant with the expected one.	

12.4.2. Connectivity Service provisioning across Multi-domain networks

Test Card 2	CTTC-ABNO-T002	Execution Status	Planned
Test Name	Connectivity Service provisioning across Multi- domain networks		
Objectives	The parent SDN controller can provide connectivity services using a YANG based APIS such as COP call and connection control services. The multi-domain path is computed and the single-domain connectivity services are requested to underlying SDN controller(s).		
Related Use Cases	All use cases defined in D1.1 [3].		
Responsible	CTTC		
Related Test Cards	N/A (not applicable)		
Additional Comments	This test involves a client that requests connectivity services across the multi-domain topology.		

SUT and topology	
SUT	Client and parent SDN controller. Both applications to be interconnected in a management network (IP reachability).

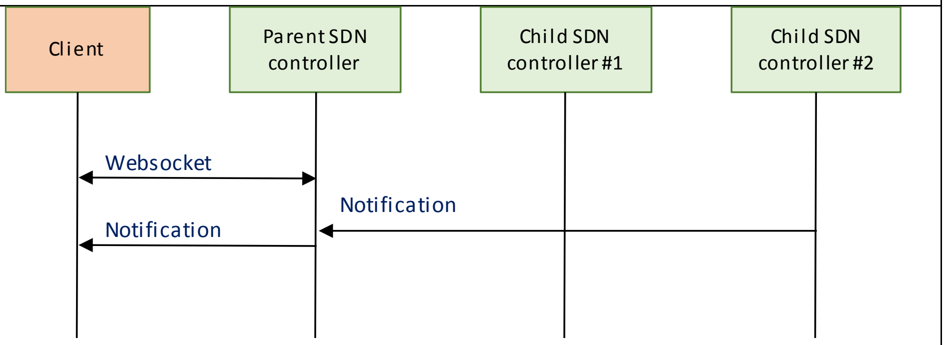
Test environment topology	 Client Parent SDN controller Child SDN controller #1 Child SDN controller #2 Connectivity Service request Connectivity Service requests Connectivity Service Connectivity Service
External components	Control plane infrastructure to enable reachability between functional entities. An (emulated or real) multi-domain network topology retrieved by the client.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure a multi-domain transport network topology, with one parent SDN controller and underlying SDN controller(s) allocated in a hierarchy. **Expected Results:** Parent SDN controller normal operation.	
2.	**Description:** Execute client application and provide the IP address and port of the services exported by the parent SDN controller. Trigger an event that results in the provisioning of a connectivity service, between provided end-points. **Expected Results:** a connectivity service established in each of the necessary underlying network domains. **Comments:** this test is basic for connectivity provisioning.	

12.4.3. Functional assessment of the SDN notifications

Test Card 3	CTTC-ABNO-T003	Execution Status	Planned
Test Name	Functional assessment of the SDN notifications.		
Objectives	The parent SDN controller exports a notification web socket that can be consumed by client applications to receive asynchronous notifications.		
Related Use Cases	All use cases defined in D1.1 [3].		
Responsible	CTTC		

Related Test Cards	N/A (not applicable)
Additional Comments	This test involves a notification protocol between a client and the parent SDN controller.

SUT and topology	
SUT	Client, parent SDN controller and underlying SDN controller(s). Functional components to be interconnected in a LAN network (IP reachability).
Test environment topology	 Client Parent SDN controller Child SDN controller #1 Child SDN controller #2 Websocket Notification Notification
External components	Basic control plane infrastructure to enable IP reachability between functional entities. An (emulated or real) network topology.

Test description		
Step #	Step description and expected results	Status
1.	**Description:** configure a transport network topology in one or multiple domains, with one SDN controller or multiple ones allocated in a hierarchy.	
2.	**Description:** execute parent SDN controller and connect it to underlying SDN controller(s) web socket notifications. **Expected Results:** Web sockets established.	
3.	**Description:** use a client application that consumes the parent SDN controller web socket for notifications. **Expected Results:** if a notification is received in the parent SDN controller, it is retransmitted to the client.	

13. Bibliography

[1] 5G-Crosshaul, Deliverable 2.1, Detailed analysis of the technologies to be integrated in the XFE based on previous internal reports from WP2 and WP3.

[2] 5G-Crosshaul, Deliverable 4.1, Initial design of the 5G-Crosshaul applications and algorithms.

[3] 5G-Crosshaul, Deliverable 1.1, Initial specification of the system architecture accounting for feedback received from WP2/WP3/WP4.

[4] ETSI, "Network Functions Virtualisation (NFV); Virtual Network Functions Architecture," December 2014. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_nfv-swa001v010101p.pdf.

[5] IETF Abstraction and Control of Transport Networks BoF. https://sites.google.com/site/actnbof/.

[6] Open Baton [Online]. Available at http://openbaton.github.io.

[7] OpenStack. Available: [Online] http://www.openstack.org.

[8] OpenDaylight [Online]. Available: http://www.opendaylight.org.

[9] Ryu controller [Online]. Available at https://osrg.github.io/ryu/.

[10] ONOS [Online]. Available: http://onosproject.org.

[11] A. Vishwanath, K. Hinton, R. Ayre, R. Tucker, "Modeling Energy consumption in High-Capacity Routers and Switches," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 8, 2014.

[12] P. Congdon, P. Mohapatra, M. Farrens, V. Akella, "Simultaneously Reducing Latency and Power Consumption in OpenFlow Switches," *IEEE/ACM Transactions on Networking*, vol. 22, no. 3, 2014.

[13] Cisco Catalyst 3750 Series Switches Data Sheet, http://www.cisco.com/c/en/us/products/collateral/switches/catalyst- 3750- series-switches/product data sheet0900aecd80371991.html.

[14] Optical Transport Protocol Extensions, Open Networking Foundation v1.0, ONF TS-022, March 2015.

[15] WebSocket protocol. https://tools.ietf.org/html/rfc6455 .

[16] RESTCONF protocol. https://tools.ietf.org/html/draft-ietf-netconf-restconf-12.

[17] OpenStack Ceilometer [Online]. Available at http://docs.openstack.org/developer/ceilometer/.

[18] 5G-Crosshaul, Deliverable 5.1, Testbed Setup and the integration and experimentation validation plan.

[19] M. Chandramouli, B. Claise, B. Schoening, J. Quittek, T. Dietz, "Monitoring and Control MIB for Power and Energy", IETF RFC 7460, March 2015

[20] ETSI NFV Use Cases [Online]. Available at:. http://www.etsi.org/deliver/etsi_gs/nfv/001_099/.../gs_nfv001v010101p.pdf.

[21] OpenFlow Switch Specification, Open Networking Foundation v1.5.1, ONF TS-025, March 2015.

[22] Open Transport ONF. https://www.opennetworking.org.

[23] IETF Teas Working Group. https://datatracker.ietf.org/wg/teas/charter/.