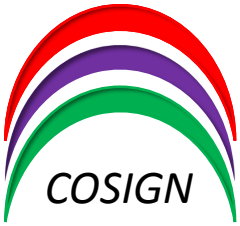

Grant Agreement No. 619572

COSIGN

Combining Optics and SDN In next Generation data centre Networks

Programme: Information and Communication Technologies

Funding scheme: Collaborative Project – Large-Scale Integrating Project

Deliverable D3.2-bis

SDN framework north-bound and south-bound interfaces specification

Due date of deliverable: 31 March 2015
Actual submission date: 31 March 2015
Resubmission date: 7 December 2015

Start date of project: January 1, 2014 Duration: 36 months

Lead contractor for this deliverable: Nextworks

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Executive Summary

This deliverable provides the specification of the north-bound and south-bound interfaces of the COSIGN control plane, which can be mapped to the Application to Controller Plane Interface and the Data plane to Controller Plane interface of the COSIGN SDN controller.

The north-bound interface provides access to the services offered by the COSIGN control plane for Data Centre Network (DCN) management and monitoring, as well as provisioning and monitoring of optical connections, optical virtual slices and virtual overlay networks. This interface enables the interaction between the DCN control and upper layer entities, like the COSIGN orchestrator and the cloud management platform that manages the whole data centre.

The south-bound interface allows the SDN controller to operate, configure and monitor the COSIGN data plane, interacting with the different type of devices defined for a COSIGN DCN. The south-bound interface is based on the OpenFlow protocol, which is extended to handle characteristics, constraints and commands related to optical devices.

The deliverable is structured as follows:

Section 1 provides an introduction to the objective of the deliverable and its positioning in COSIGN activities.

Section 2 defines the role of the COSIGN control plane in the overall system architecture, identifying the role of north-bound and south-bound interface of the control plane in the COSIGN framework.

Section 3 is dedicated to the specification of the north-bound interface for the main services of the DCN control plane.

Section 4 provides the specification of the south-bound interface, identifying the functional entities responsible for the exchange of messages between controller and data plane and specifying the OpenFlow extensions required to model configuration commands, advertising of capabilities and retrieval of monitoring data for COSIGN devices.

Executive Summary update for D3.2-bis

The document has been updated during the second half of Y2 in order to document the latest changes in WP3 architecture design and clarify the origin of the optical extensions for the OpenFlow protocol, as suggested by the reviewers during the mid-term technical review. In particular, the following modifications from the original D3.2 have been included in the document:

- Additional sections:
    - Section 2.2, describing the main changes in the COSIGN DCN Control Plane architecture design.
    - Section 3.2, describing the procedures for the coordination between overlay virtualization and provisioning of optical circuits at the underlying network, in support of the COSIGN vApp use case.
    - Section 3.3.4, specifying the north-bound APIs for the Forwarding Rule Manager service.
    - Section 4.4.4, summarizing the origin of the OpenFlow extensions used in COSIGN.
    - Section 4.4.5, summarizing the relation between the information models of COSIGN devices and the OpenFlow extensions.
- Modified sections:
    - Section 3.3.2.4, to document the changes in the software design of the Virtual Infrastructure Manager.

- Section 3.3.6.3, to document the changes in the software design of the Policy and SLA Manager.
- Section 4.4, with editing changes to improve readability.

Document Information

Status and Version:	V1.0 – bis	
Date of Issue:	December 7, 2015	
Dissemination level:		
Author(s):	**Name**	**Partner**
	Giada Landi	NXW
	Giacomo Bernini	NXW
	Cosmin Caba	DTU
	José Soler	DTU
	Yaniv Ben-Itzhak	IBM
	Amaia Legarrea	i2CAT
	Jose Aznar	i2CAT
	Matteo Biancani	IRT
	Alessandro Predieri	IRT
	Fernando Agraz	UPC
	Salvatore Spadaro	UPC
	Albert Pagès	UPC
	Jordi Perelló	UPC
	Oded Raz	TUE
	Karel Van de Plassche	PhotonX
	Bingli Guo	UNIVBRIS
	Shuping Peng	UNIVBRIS
	Reza Nejabati	UNIVBRIS
	Georgios Zervas	UNIVBRIS
	Dimitra Simeonidou	UNIVBRIS
	Chris Jackson	UNIVBRIS
M15 Submission:		
Edited by:	Giada Landi	NXW
Checked by :	Sarah Ruepp	DTU
	Plus WP Leaders	
M23 Submission:		
Edited by:	Giada Landi	NXW
Reviewed by:	Oded Raz	TUE
	José Soler	DTU
Checked by:	Sarah Ruepp	DTU
	Helene Udsen	DTU

Table of Contents

1 Introduction

The COSIGN architecture defined in WP1 has identified three main layers in the COSIGN system: the Data Centre Network (DCN) data plane, the Software Defined Networking (SDN)-based control plane for management and operation of the network infrastructure and the orchestrator for the converged management and allocation of the different resources available in the data centre. Two of the main innovations introduced by COSIGN in the data centre environment are related to the enhancement of the data plane, introducing optical technologies and the joint management of network and IT resources enabled through the cooperation between the DCN control plane and the service orchestration functions in the cloud management platform. Fundamental enablers for these innovations are the interfaces which regulate the interaction between the related layers in the COSIGN architecture. The heterogeneous data plane requires a suitable south-bound interface at the network control plane which allows to discover, manipulate, configure and monitor the new types of resources in order to exploit their capabilities, connection granularity and dynamicity. On the other hand, the interaction between network controller and orchestrator needs to rely on a powerful interface to provide access to the control plane services with abstract primitives that hide the data plane technology details but expose network programmability and monitoring with a sufficient level of detail.

The objective of this deliverable is to specify the SDN controller interfaces at the north-bound and south-bound side, in compliance with the SDN architecture defined by the Open Networking Foundation (ONF) [ONF-arch] and, in particular, with the Application to Controller Plane Interface (A-CPI) and the Data plane to Controller Plane Interface (D-CPI). The design of these interfaces has taken into account both official and de-facto standards, in order to guarantee the easy adoption of the COSIGN solution in SDN-based Data Centres environments, their interoperability with existing SDN frameworks and facilitate the exploitation of the project outcomes.

In particular, the reference protocol for the south-bound interface is OpenFlow [OF], which has been extended for its adoption in environments with optical devices or to enable additional mechanisms for advanced overlay, QoS or OAM in Ethernet switches. The specification of the south-bound interface can be found in section 4.

On the other hand, the north-bound interface is based on the REST paradigm [REST]. Even if the related ONF Working Group (WG) has not yet defined a specific standard for this interface, most of the SDN controllers available today in the market and in the open-source community adopt REST APIs to facilitate the integration with external applications and cloud platforms. Following the latest standardization efforts in IETF, the RESTConf protocol [RESTCONF] and the YANG language [RFC6020] have been considered for the specification of the north-bound interface of most of the control plane services. Since this type of API is fully supported by OpenDaylight, the SDN controller selected as baseline for the implementation of the COSIGN controller (see section 2.1), RESTConf is the default choice for all the services that will be implemented as internal modules in OpenDaylight. On the other hand, the compliance with RESTConf will be not mandatory for services which will be developed as external applications, e.g., the overlay based network virtualization. The north-bound interface, with the details of REST APIs and service parameters, is specified in section 3.

This deliverable constitutes the final result of the initial phase of WP3 activities, which have been focused on the high-level specification of the COSIGN control plane architecture. The architecture defined in deliverable D3.1 [3] and the specification of the interfaces provided in this document will be the input for the implementation activities in Task 3.2 and Task 3.3, dedicated to the software design and development of the COSIGN controller and its services.

1.1 COSIGN Reference Material

1.1.1 Reference Documents

[1]	COSIGN – Deliverable D1.1 – Requirements for Next Generation intra-Data Centre Networks Design
[2]	COSIGN – Deliverable D1.3 – Comparative analysis of control plane alternatives
[3]	COSIGN – Deliverable D3.1 – SDN framework functional architecture
[4]	COSIGN – Deliverable D4.1 – COSIGN orchestrator requirements and high level architecture
[5]	COSIGN – Deliverable D4.2 – COSIGN orchestrator low level architecture and prototype design

1.1.2 Acronyms and Abbreviations

Most frequently used acronyms in the Deliverable are listed below. Additional acronyms can be specified and used throughout the text.

AAA	Authentication, Authorization and Accounting
A-CPI	Application-Controller Plane Interface
API	Application Programming Interface
BGP	Border Gateway Protocol
BW	Bandwidth
C2C	Controller to Controller
CLI	Command Line Interface
CRUD	Create Read Update Delete
DC	Data Centre
DCN	Data Centre Network
D-CPI	Data-Controller Plane Interface
FL	Floodlight
GCO	Global Concurrent Optimization
HA	High Availability
HAL	Hardware Abstraction Layer
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
MD-SAL	Model Driven Service Abstraction Layer
NaaS	Network as a Service
NAT	Network Address Translation
NE	Network Element
NFV	Network Function Virtualization
NVGRE	Network Virtualization using Generic Routing Encapsulation
OF	OpenFlow
OF-DPA	Open Flow Data Plane Abstraction
ONF	Open Networking Foundation
OS	Operating System
OSGi	Open Service Gateway initiative
OVSDB	Open vSwitch Database management protocol
P2MP	Point to Multi-Point
PCE	Path Computation Element
PCEP	Path Computation Element communication Protocol
REST	Representation State Transfer
SAL	Service Abstraction Layer
SDM	Space Division Multiplexing
SDN	Software Defined Networking
SLA	Service Level Agreement

SNMP	Simple Network Management Protocol
SPF	Shortest Path First
SSL	Secure Sockets Layer
STT	Stateless Transport Tunnelling
TDM	Time Division Multiplexing
TTP	Table Type Parameter
VDC	Virtual Data Centre
VI	Virtual Infrastructure
VIO	Virtual Infrastructure Operator
VLAN	Virtual LAN
VM	Virtual Machine
VNF	Virtual Network Function
VPN	Virtual Private Network
VXLAN	Virtual Extensible LAN
XC	Cross connection
WDM	Wavelength Division Multiplexing

1.2 Document History

Version	Date	Authors	Comment
00		See the list of authors	
04	20th Feb. 2015		
05	20th March 2015		
10	31st March 2015		
10-bis	7th December 2015		

2 DCN Control Plane in COSIGN architecture

In an architecture design, the control plane is typically situated in between the data plane (defined by COSIGN WP2) and the orchestration plane (defined by COSIGN WP4); the control plane in COSIGN will not be an exception in that sense. A thorough examination of control plane alternatives was carried out and explained in D3.1 [3] where the high level architecture design was defined as well as the functional blocks to provide the committed service requirements to the users of the infrastructure. It has been agreed by the consortium that the COSIGN control plane will be implemented through OpenDaylight (see section 2.1). The cross-layer interaction is enabled by the north-bound interfaces to the orchestration and management plane and by the south-bound interfaces to the DCN data plane. According to ONF, the north-bound and south-bound interfaces can be referred as Application to Controller Plane Interface (A-CPI) and Data plane to Controller Plane Interface (D-CPI) respectively.

The network programmability offered by the SDN control plane solutions is fundamental to the efficient integration between network and cloud environments, where the DCN connectivity needs to be tightly coupled with the evolution of the Virtual Machine (VM) lifecycles with high degrees of automation.

The SDN controller, through the D-CPI, sends instructions to agents that represent the abstraction of the data plane devices or to network elements which provide functional and logical capabilities to allow the SDN controller to manage these network entities. This model is known as the controller-agent model. The controller is a functional component that represents the client's resources and capabilities. The lowest layer of abstraction is composed by the data plane resources, which are physical entities (e.g., soft switches) whereas a higher level of abstraction would include virtual resources that would have the functionalities and behaviour of the NE (network element) but virtualized and represented by the D-CPIs.

The communication between the SDN controller and the data plane at the D-CPI interface is based on technology agnostic protocols like OpenFlow (OF) [OF], an open standard protocol to execute software/user-defined flow based routing, control and management. Heterogeneous data plane devices need to be configured by the control plane in a consistent way. This requires the exchange of D-CPI messages, in polling, synchronous or asynchronous mode, for a set of basic functions:

- Collection of device attributes from the data plane to the SDN controller, where they are stored and elaborated at the Switch Manager and the Topology Manager. OF protocol extensions may be required to describe particular characteristics of some optical devices.
- Monitoring of network status, to collect and record events like link failures or status changes, which can be exposed at the north-bound API of the SDN controller.
- Configuration of the data plane optical devices, through new OF actions (e.g., to create a cross-connection in an optical node).

The north-bound interfaces, which run between the control plane and the orchestration layer, have been defined from a high level point of view in D4.1 [4], including all the mechanisms used by the Orchestration layer, from Cloud Management Platform, to request services on the DCN or network infrastructure information collection. The interface is typically based on REST APIs that support CRUD (Create Read Update Delete) operations on network related resources. The type of requests that will be received by the control plane from the orchestration plane can be categorized as one of these three:

- requests for customized virtual networks;
- requests for dynamic optical connectivity within the DC;
- DC network information and status collection.

Further information and descriptions can be found in D4.1 [4].

The COSIGN consortium has defined a list of service requirements that will be implemented along the project execution. Those service requirements that are related with the control plane layer are listed in

Table 1, where the interfaces impacted by each requirement have been identified (NB for north-bound interface and SB for south-bound interface).

Table 1 – Mapping between requirements and interfaces

Number	Service	Description	Interface
SERV-01	Seamless Network Resources Provisioning	Network resources should be provisioned to applications flexibly and transparently; underlying IT infrastructure should be configured automatically, to achieve fast (in minutes) application/service deployment.	NB
SERV-02	Advanced Network Services Provisioning	The DC should be able to provision advanced network services on demand, allowing dynamic provisioning and configuration, and including service chaining and orchestration for complex services.	NB / SB
SERV-03	Service level monitoring	In order to provide service layer monitoring, the monitoring mechanisms must be deployed at the physical infrastructure layer and forwarded to the upper service layer. The number of monitoring alarms forwarded to the service layer will be lower than the number of total alarms managed at the resource level and the alarms' information must be abstracted. The DC should facilitate application level monitoring required to decide whether service levels agreements are satisfied.	NB / SB
SERV-04	Adapting to application/service dynamicity	Resources allocated for a service or an application should be dynamically re-provisioned according to the changes in demand and/or in requirements of the application/service.	NB
SERV-05	Flexibility	The DC needs to be flexible enough to accommodate different types and sources of data without sacrificing performance or latency and foresee any requirement for dynamically increasing or decreasing capacity (upscaling, downscaling) according to predefined policies.	NB
SERV-06	QoS provisioning	The DC should provide mechanisms to satisfy application's QoS requirements (e.g., BW, delay, and resiliency) and policies set to ensure the agreed service level. The global SLAs must be translated into network QoS by the SW layers.	NB
SERV-07	Real Time Control	Control of network resources should allow real time detection and reaction to changes in the state of the infrastructure resources to cope with performance constraints. This will help reducing the impact of failures and unexpected behaviors. HW in the data plane should report faults to higher level– timely failure detection and reliable failure reporting.	SB
SERV-08	Big data support	Applications making use of big data should be supported by future DCs, having an impact on capacity, latency, access, security, cost and flexibility and requiring Big data analytics.	NB
SERV-09	Secure data management	Secure use of end-to-end data within applications as well as secured access to remote data sources.	NB

SERV-10	Security management	Provide a way of managing security mechanisms for a coordinated physical, network, data and user security for different stakeholders (user, service provider, cloud provider).	NB
SERV-11	Self-service management interfaces	Management interfaces should allow the configuration of self-service policies, notifications, information retrieval and monitoring and the configuration of the elements of the service, independently of other services running in the system.	NB
SERV-12	Multi-tenant Isolation	Each application or service deployed in the cloud should be sufficiently isolated from the rest of the workloads, in terms of data, management and performance. DC networks isolation techniques strongly rely on virtualization mechanisms and abstraction tools.	NB / SB
SERV-13	Service Traffic Profile	Each service or application to be deployed in the DC should provide information about its expected traffic profile and connection requirements, so that the DC network can be configured to offer the required connectivity at service runtime. Traffic profile information and requirements should include the following parameters: max packet loss, minimum guaranteed bandwidth, peak rate, jitter and latency, type of connectivity (point-to-multi-point, point-to-point, asymmetric/symmetric connectivity), recovery strategies, redundancy. Since VMs may be instantiated on the same DC or distributed among different DCs, the service traffic profile may impact in the intra-DC connectivity or also the inter-DC connectivity.	NB
SERV-14	Generalized information model	The specification of cloud applications and services should be based on a generalized information model, able to describe a wide variety of applications in a powerful, user-friendly and abstracted manner, completely technology-independent. The information model must be able to support complex topologies and interactions between the logical entities composing the service.	NB / SB
SERV-15	Service connectivity disjunction	Capability to specify two fully disjointed connection in order to provide redundancy. The accomplishment of this requirement relies on data plane limitations.	NB / SB
SERV-16	VDC optical BW control (or control of physical network aspects of the IaaS service provided)	VDC operator should be able to control the optical BW (e.g., enable access to the optical wavelength or slot). One step beyond is also to provide access to the deployed connections' BW so that VIOs (Virtual Infrastructure Operator) have control about the available capacity in the DC.	NB / SB
SERV-17	Service Calibration	Permit users to customize their services.	NB / SB
SERV-18	Intelligent/selective physical failure	Depending severity of the failures of the physical layer, they should be hidden or noticed to the IaaS	NB / SB

	awareness	service consumer.	
SERV-19	Equipment Inventory Services Tool	Provide a services catalogue of the whole suite of DC network services, capabilities of each service and their functionalities. Service customer is this way aware of the whole network connectivity services that can be managed in the DC.	NB

Sections 3 and 4 of this document will focus on the description of the north-bound and south-bound respectively, specifying the APIs and OF extensions that will be implemented in COSIGN from a low level point of view.

2.1 OpenDaylight as reference platform for COSIGN controller

During the first year of the project, WP1 and WP3 have analyzed different SDN controllers from the double perspective of supported functions and possible business impacts (WP1) and software architecture and code maturity and exploitability (WP3). Taking into account the outcomes of this analysis, the COSIGN consortium has decided to adopt OpenDaylight as reference platform for the implementation of the COSIGN controller.

The OpenDaylight choice is motivated by technical and business reasons. From a technical point of view, OpenDaylight provides a comprehensive and flexible SDN platform where new south-bound drivers and interfaces, internal core functions, north-bound APIs and high-level SDN applications can be implemented and integrated as new plugins. This modularity is fundamental for the COSIGN controller, which needs to implement extensions at the south-bound protocols and core services (e.g., to support optical constraints in OpenFlow protocol and topology services), but also develop new functions (e.g., the on-demand provisioning of optical connections or virtual optical slices) strongly integrated with the other controller services. This task is facilitated by the software architecture of the OpenDaylight platform, which is built around the concept of Model Driven Service Abstraction Layer (MD-SAL). All the controller functions are abstracted through the specification of YANG [RFC6020] models and implemented through OpenDaylight plugins which interact each other through an efficient set of core mechanisms implemented by the MD-SAL itself (e.g., exchange of information through a flexible data store, asynchronous execution of remote procedure calls, subscriptions and notifications). The platform integrates powerful tools (i.e., the YANG tools) which automate the generation of large part of the new plugins' code starting from their YANG model. This covers in particular the data model objects, the java-based and RESTCONF APIs at the north-bound of each service and the configuration and integration of the plugins with the overall OpenDaylight platform, including the usage of the MD-SAL core services. Thus, the usage of OpenDaylight and its MD-SAL allows COSIGN developers to focus on the real added value of the COSIGN prototype: the definition of open and standard-based interfaces for the DCN optical network control plane services and the implementation of their internal logic and features. Beyond this, OpenDaylight already offers some basic core services for L2-L3 technologies that can be used as a reference to implement the services defined for COSIGN, which are based on optical data plane devices. Where possible, these software modules will be re-used and extended to maximize the efficiency of the development activities. For example, this is the case for the OpenFlow library and plugin, the topology manager, the inventory manager and the statistics manager integrated in OpenDaylight Helium and Lithium version.

OpenDaylight community is an excellent candidate for disseminating COSIGN outcomes in the open source community and in the SDN market. Several industrial companies with a strong role in the ICT sector are actively contributing to the OpenDaylight project. IBM which is a founding member of ODL and a partner of the COSIGN consortium will work towards establishing the required communication channels and consider presenting the COSIGN project in OpenDaylight events (e.g., the OpenDaylight Summit). Moreover, the fact that OpenDaylight is adopted in several SDN-based cloud environments based on OpenStack deployments increases the chances for a real impact also in the OpenStack cloud community.

2.2 Evolution of COSIGN DCN Control Plane architecture in Y2

The COSIGN control plane architecture has evolved from the specification in Deliverable D3.1 [3] introducing new functional entities and re-designing some software components. The main changes are summarized in Figure 1.

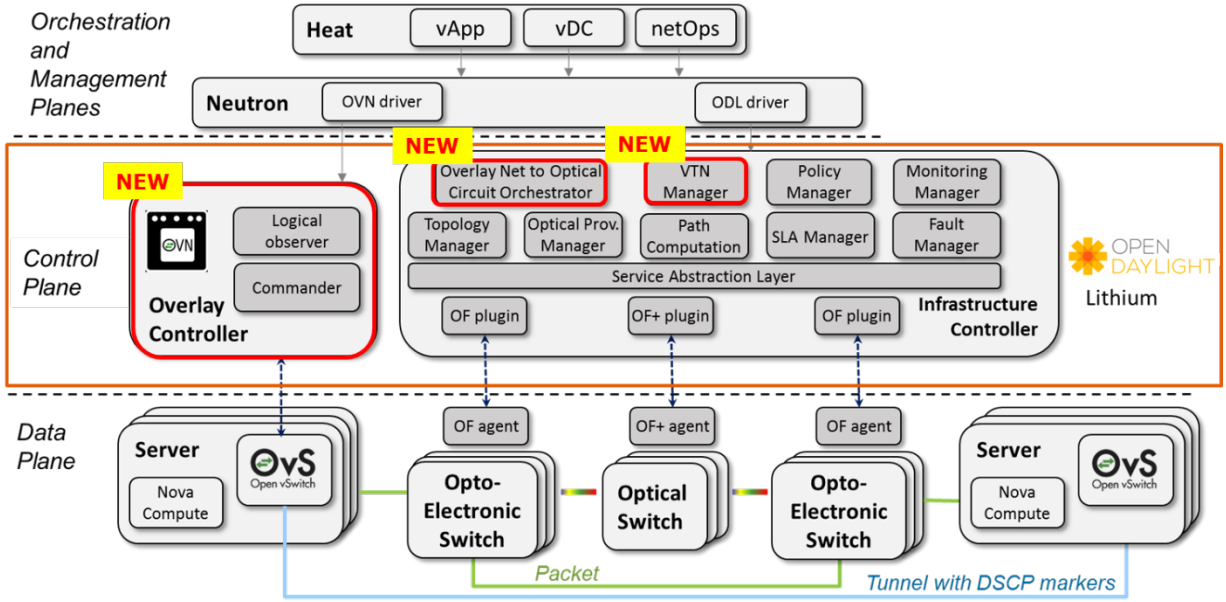

Figure 1 – COSIGN DCN Network Control Plane architecture

In particular, the following changes have been introduced:

- Splitting between an Infrastructure Controller, based on OpenDaylight, and an Overlay Controller, based on OVN. The Infrastructure Controller is in charge of the configuration and monitoring of the DCN devices and implements the logic for the provisioning and management of the underlying intra-DC connectivity to optimize the allocation of the physical resources. The Overlay Controller, on the other hand, is responsible for the provisioning of overlay virtual networks by configuring the OpenvSwitch entities at the network edges. The two controllers cooperate through a dedicated SDN application which maps the DSCP tagged flows on the dynamic optical circuits established on the underlying network. This approach allows efficient coordination of the overlay virtualization with the operation and management of the underlying optical DCN. Further details about this procedure and the associated workflow are provided in section 3.2.
- At the software architecture level, the consortium has decided to adopt the latest OpenDaylight release (i.e., Lithium instead of Helium) as starting point for the development of the Infrastructure Controller. The main benefits are related to a more extensive usage of the MD-SAL approach across the variety of OpenDaylight plugins and the availability of more stable additional plugins which can be used as baseline for specific COSIGN components (e.g., the Topology Processing Framework).
- The software design of some COSIGN components has been updated, in particular:
    - The overlay based network virtualization, previously designed as extension of the OpenDOVE application is now based on OVN and is implemented in the Overlay Controller, as described above.
    - The planning and provisioning of virtual network slices in support of the Virtual Data Centre use case, previously designed as OpenVirtex extension is now based on the OpenDaylight Virtual Tenant Network (VTN) application, properly extended to support the virtualization of optical network elements. This approach allows a more efficient integration with the other COSIGN components.

- The Policy and SLA Manager is implemented as a new OpenDaylight plugin developed from scratch, without relying on pre-existing components, as documented on section 3.3.6.3.

3 DCN Control Plane north-bound interface

This section describes the north-bound interface of the COSIGN DCN Control Plane. This interface is enabling the interaction with external architecture layer and applications, like the orchestrator and the cloud management platform, which can access and use the main services provided by the COSIGN system for the management, control and virtualization of the data centre network.

From an architectural point of view, the north-bound interface corresponds to the A-CPI defined in the SDN architecture specified by the Open Networking Foundation. In COSIGN the services exposed at the north-bound interface can be implemented internally in the SDN controller (e.g., the provisioning of optical connections is provided by a software module which runs in the controller platform) or can be provided by external applications, as in the case of the provisioning of overlay virtual networks. However, in both cases, the north-bound interface is designed as a REST API, where services are modelled through resources that can be retrieved and manipulated through CRUD actions. This approach allows an easy integration between the COSIGN DCN Control Plane and other entities of the DC cloud platform, which can interact with DCN management and operational services through a simple REST client.

This section initially discusses the current standardization efforts in the area of the north-bound interfaces and related protocols in SDN environments, mainly active in ONF and IETF. Then, the REST APIs of the main services supported by the COSIGN DCN Control Plane are presented. For each service, the specification includes the list of APIs, the associated input and output parameters and the possible error codes. Finally, for the COSIGN services which can be developed through extensions and enhancements of software modules already existing in the OpenDaylight platform, the matching with the original APIs is discussed analyzing the improvements required to meet the COSIGN requirements.

3.1 Reference technologies and standards

The Open Networking Foundation (ONF) has created a North Bound Interface Working Group (NBI-WG) in 2013, with the objective of defining and standardizing a set of SDN Controller North-bound API Interfaces (NBIs) to enable applications' portability across controllers. The NBI-WG Charter [ONF-NBI] identifies different levels of NBIs, depending on the type of services or resources they expose. As shown in Figure 2 a set of "controller APIs" provides access to basic functions or network services which constitute a sort of "developer toolkit" APIs for SDN software developer. They cover common capabilities and functions of SDN controller platforms and they are typically protocol specific. On top of them, the NBI Application APIs are exposed by specific and customized SDN applications built over the controller platform: they are protocol independent and use-case or domain specific.

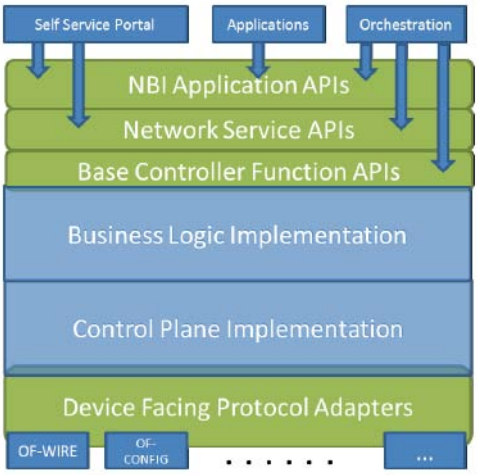

Figure 2 – Different levels of north-bound interfaces [ONF-NBI]

Applying these concepts to the COSIGN system, we can identify *Base Controller Function APIs* for some basic services of the COSIGN controller (e.g., topology and inventory, monitoring and fault manager). On the other hand, the *Network Service APIs* can be mapped to the interfaces offered by the controller's internal components which implement some services offered by the COSIGN control plane (e.g., the Optical Provisioning Manager or the Virtual Infrastructure Manager) to other architectural layers. Finally the NBI Application APIs are provided in COSIGN by the control plane network applications developed over the controller. An example is the application for the overlay-based network virtualization (refer to [3] for the list of functional components defined in the COSIGN control plane).

The initial objective of the ONF NBI-WG is the definition of an information model for the most relevant NBIs and their proposed encoding in a programming language neutral manner. However, at the moment, the WG has not yet released any official document and the variety of SDN controllers available so far is implementing its own north-bound APIs without following any standard information model. Most of the controller platforms, including OpenDaylight, implement two paradigms of north-bound APIs for each service:

- Internal interfaces (e.g., java interfaces) to be used by software components developed as internal modules of the platform itself and acting as consumers of the service.
- REST APIs, based on the HTTP protocol, to allow external applications to perform CRUD operations over the resources exposed by the controller.

In particular, OpenDaylight supports a special kind of REST APIs, based on the RESTCONF protocol [RESTCONF] and specified through YANG [RFC6020] information models.

YANG is an IETF standard, specified within the NETCONF Data Modelling Language Working Group (Netmod WG), for modelling network element configuration in the Netconf protocol. In OpenDaylight, YANG is the modelling language used in the Model Driven Service Abstraction Layer (MD-SAL) to specify the interfaces of services and plugins. In particular, it is used to model:

- Configuration and Operational Data Tree, i.e., the data structures which represent the configurable parameters and the state of components and systems. Each portion of a sub-tree is uniquely identified in the configuration or operational space through an instance identifier.
- Remote Procedure Calls (RPCs), used for calls and invocations that cross the boundaries between different modules. They are implemented by modules which act as Service Providers and are invoked by Service Consumers.
- Notifications, for asynchronous events published by Service Providers for the subscribed listeners.

YANG constitutes a full, formal contract language with a rich syntax and semantics to build applications on and it is characterized by hierarchical and highly modular models which can be easily re-used and extended through "augmentation" to define new services. The IETF Netmod WG has released several YANG models to define the most common network concepts: examples are the *ietf-inet-types* specified in RFC 6991 [RFC6991] or the *ietf-interfaces* specified in RFC 7223 [RFC7223]. New YANG models for the definition of SDN controller services and their north-bound APIs should re-use the existing standard models as much as possible.

RESTCONF is a REST-like protocol running over HTTP for accessing data modelled in YANG. It derives from Netconf, without replacing it, but just providing additional and simplified REST APIs, particularly suitable for resource-oriented device abstraction. RESTCONF is the protocol used at the north-bound APIs in OpenDaylight and allows external applications to access the configuration and operational data stores of the controller or invoke the RPCs for specific operations defined in the YANG data models of OpenDaylight modules. The protocol itself supports also asynchronous notifications through web sockets, even if the implementation of this feature in the latest version of OpenDaylight is still limited.

Following the traditional REST principles, RESTCONF provides CRUD operations on a data store, with data modelled as resources, identified by a URI, which can be manipulated depending on their

access rights. RESTCONF operations are translated over HTTP messages, with edit operations mapped through POST, PUT, PATCH or DELETE HTTP methods and retrieval requests based on GET or HEAD methods. Each operation is represented by a pair <method,URI> where URIs have a standard format: "/restconf/<path>?<query>". The content of the messages can be in XML [RFC6020] or JSON format [JSON-YANG].

3.2 Dynamic Optical Circuits (Network Orchestrator)

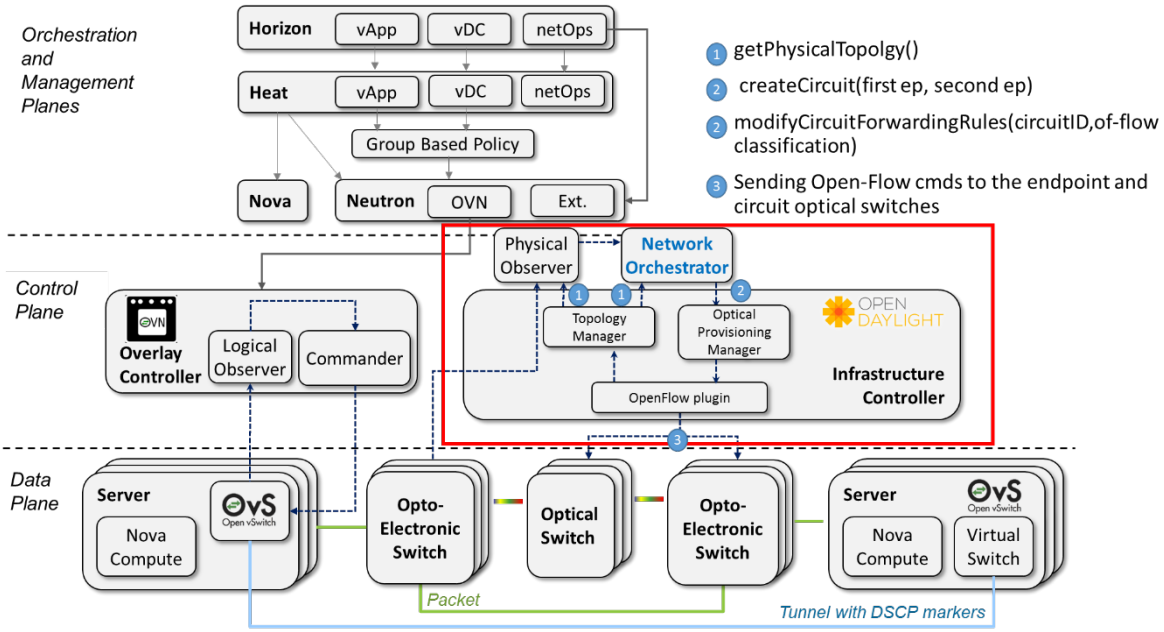

Figure 3 – APIs flow between ODL and network orchestrator

The COSIGN vApp use-case (see Deliverable D4.2 [5]) employs dynamic circuit establishment according to the traffic in the data-plane. To that end, flows are tagged by the OVS with DSCP markers. These DSCP markers identify both the QoS of the flow and the type of the flow (e.g., mice, elephant, short, long, etc ...). The flows are monitored at the optoelectronic switches which are connected directly to the optical switch, by the physical observer.

The physical observer and the network orchestrator get the physical topology from ODL (Step 1 in Figure 3). In this step, the physical observer identifies the TUe switches which are connected directly to the optical switch and the hosts which are served by each such TUe switch. The physical observer can identify and inform the network orchestrator about scenarios which require an optical circuit over the optical switches, such as aggregated mice flows, elephant flow, etc. The network orchestrator decides which circuit to establish depending on the current optical circuits, traffic demands, QoS and more. Then, the network orchestrator creates the new circuit (Step 2 in Figure 3), using the OpenDaylight APIs exposed by the Optical Provisioning Service (see section 3.3.3). Once, it gets ACK and connection ID from the Infrastructure Controller, the network orchestrator can modify the forwarding flow table of the corresponding optoelectronic switches in order to forward the desired flows through this circuit (Step 3 in Figure 3). The network orchestrator can reuse current circuits over the optical switch for different and new flows by re-modifying the forwarding rules for a given circuit over the corresponding optoelectronic switches. In response, the Infrastructure Controller sends the corresponding Open-Flow commands to the optoelectronic switches. The API for the creation and modification of forwarding rules is documented in section 3.3.4.

3.3 Specification of north-bound APIs for control plane services

In COSIGN the north-bound APIs which provide the access to the control plane services are defined as REST APIs. Moreover, since COSIGN will adopt OpenDaylight as reference platform for the development of the COSIGN controller, the north-bound interfaces of all the services developed as internal modules of the SDN controller will be compliant with the RESTCONF protocol. In terms of implementation, this means that the interface of each service or module will be developed through the definition of the associated YANG model, while its translation in java classes and interfaces and REST APIs will be auto-generated through the YANG tools available in the OpenDaylight platform.

On the other hand, the full compliance with the RESTCONF protocol is not mandatory for the control plane services developed as external applications running over the controller itself. For example, this is the case for the overlay-based network virtualization service which will expose a REST APIs not correlated to any YANG model.

The following subsections provides the high-level specification of the REST APIs for the main services exposed by the COSIGN control plane towards the other layers of the COSIGN architecture or the management platform for the administration of the data centre network. On the other hand, the specification of the APIs for all the internal components of the control plane is beyond the scope of this document.

3.3.1 Provisioning of overlay virtual networks

This service manages the overlay virtual networks of the tenants. In general, these virtual networks can be shared and/or have external accessibility.

3.3.1.1 REST APIs

Table 2 – List of APIs for provisioning of overlay virtual networks

Operation	URI	Description
List networks	List networks	Lists networks to which the specified tenant has access.
Create network	Create network	Creates a network.
Bulk create networks	Bulk create networks	Creates multiple networks in a single request.
Show network	Show network	Shows information for a specified network.
Update network	Update network	Updates a specified network.
Delete network	Delete network	Deletes a specified network and its associated resources.

Table 3 – List networks specification

List networks			
Request parameters	tenant_id	UUID	The tenant ID
Response parameters	admin_state_up	bool	The administrative state of the network, which is up (true) or down (false)
	Id	uuid	The network ID
	Name	string	The network name
	Shared	bool	Indicates whether this network is shared across all tenants. By default, only administrative users can change this value
	Status	string	The network status.
	Subnets	dictionary	The associated subnets
Successful response codes: 200			
Error response codes: unauthorized (401)			

Table 4 – Create network specification

Create network			
Request parameters	admin_state_up (optional)	bool	The administrative state of the network, which is up (true) or down (false).
	name (Optional)	string	The network name. A request body is optional: If you include it, it can specify this optional attribute.
	shared (Optional)	bool	Indicates whether this network is shared across all tenants. By default, only administrative users can change this value.
	tenant_id (Optional)	uuid	Admin-only. The UUID of the tenant that will own the network. This tenant can be different from the tenant that makes the create network request. However, only administrative users can specify a tenant ID other than their own. You cannot change this value through authorization policies.
	router:external (Optional)	bool	Indicates whether this network is externally accessible.
Response parameters	admin_state_up	bool	The administrative state of the network, which is up (true) or down (false).
	Id	uuid	The network ID.
	name	string	The network name.
	shared	bool	Indicates whether this network is shared across all tenants. By default, only administrative users can change this value.
	Status	string	The network status.
	subnets	Dictionary	The associated subnets.
	tenant_id	UUID	The tenant ID
Successful response codes: 201			
Error response codes: badRequest (400), unauthorized (401)			

Table 5 – Bulk create networks specification

Bulk create networks			
Request parameters	admin_state_up (optional)	bool	The administrative state of the network, which is up (true) or down (false).
	name (Optional)	string	The network name. A request body is optional: If you include it, it can specify this optional attribute.
	shared (Optional)	bool	Indicates whether this network is shared across all tenants. By default, only administrative users can change this value.
	tenant_id (Optional)	uuid	Admin-only. The UUID of the tenant that will own the network. This tenant can be different from the tenant that makes the create network request. However, only administrative users can specify a tenant ID other than their own. You cannot change this value through authorization policies.
	router:external (Optional)	bool	Indicates whether this network is externally accessible.
Response parameters	admin_state_up	bool	The administrative state of the network, which is up (true) or down (false).
	Id	uuid	The network ID.
	name	string	The network name.
	shared	bool	Indicates whether this network is shared across all tenants. By default, only administrative users can change this value.
	Status	string	The network status.
	subnets	Dictionary	The associated subnets.
	tenant_id	UUID	The tenant ID
Successful response codes: 201			
Error response codes: badRequest (400), unauthorized (401)			

Table 6 – Show network specification

Show network			
Request parameters	network_id	uuid	The UUID for the network of interest to you.
Response parameters	admin_state_up	bool	The administrative state of the network, which is up (true) or down (false).
	Id	uuid	The network ID.
	name	string	The network name.
	shared	bool	Indicates whether this network is shared across all tenants. By default, only administrative users can change this value.
	Status	string	The network status.
	subnets	Dictionary	The associated subnets.
	tenant_id	UUID	The tenant ID
Successful response codes: 200			
Error response codes: badRequest (400), unauthorized (401)			

Table 7 – Update network specification

Update network			
Request parameters	network_id	uuid	The UUID for the network of interest to you.
	admin_state_up (optional)	bool	The administrative state of the network, which is up (true) or down (false).
	name	string	The network name. A request body is optional: If you include it, it can specify this optional attribute.
	shared (Optional)	bool	Indicates whether this network is shared across all tenants. By default, only administrative users can change this value.
	tenant_id (Optional)	uuid	Admin-only. The UUID of the tenant that will own the network. This tenant can be different from the tenant that makes the create network request. However, only

			administrative users can specify a tenant ID other than their own. You cannot change this value through authorization policies.
	router:external (Optional)	bool	Indicates whether this network is externally accessible.
Response parameters	admin_state_up	bool	The administrative state of the network, which is up (true) or down (false).
	Id	uuid	The network ID.
	name	string	The network name.
	shared	bool	Indicates whether this network is shared across all tenants. By default, only administrative users can change this value.
	Status	string	The network status.
	subnets	Dictionary	The associated subnets.
	tenant_id	UUID	The tenant ID
	router:external	bool	Indicates whether this network is externally accessible.
Successful response codes: 200			
Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404)			

Table 8 – Delete network specification

Delete network			
Request parameters	network_id	uuid	The UUID for the network of interest to you.
Response parameters	<none>		
Successful response codes: 204			
Error response codes: unauthorized (401), itemNotFound (404), conflict (409)			

3.3.1.2 Matching with the reference SDN controller

The aforementioned REST APIs are based on OpenStack networking API v2.0. The OpenDaylight SDN controller is now integrated with OpenStack Liberty and therefore should support these REST APIs.

3.3.2 Provisioning of virtual optical slices

The virtual optical slice implements a fully automated Infrastructure as a Service (IaaS) solution that incorporates different optical devices/technologies across data centres. This section focuses on the key information model of the virtual optical slices provisioning, based on the outlined requirements (selection of virtual resource profiles and configuration selected by the customer/user). Main access point of the customer to the virtual optical slices environment is the portal through which virtual resources are requested, instantiated according to selected templates and constraints, and subsequently monitored in terms of health statistics and performances.

3.3.2.1 Information model

Figure 4 shows the information model for virtual optical slice provisioning.

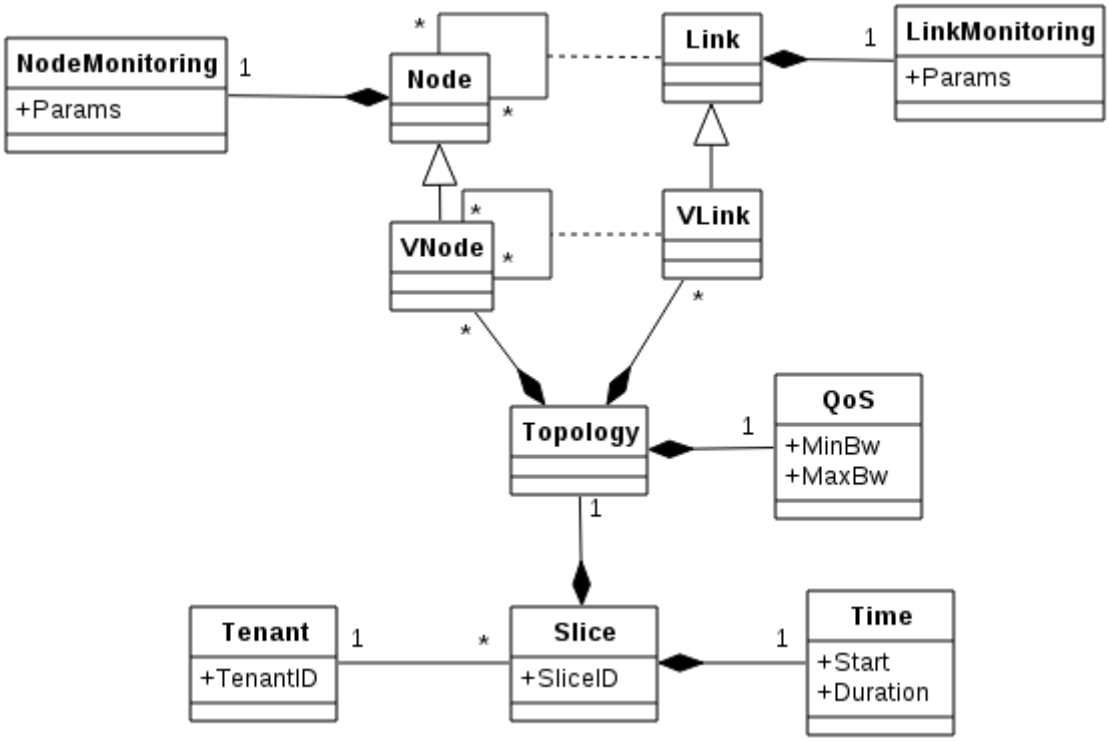

Figure 4 – Information model for virtual optical slices creation and management

3.3.2.2 REST APIs

Table 9 – List of APIs for provisioning of virtual optical slices

Operation	URI	Description
POST	POST/restconf/config/virtual-infrastructure-manager:slice	Creates a new virtual optical slice.
GET	GET/restconf/config/virtual-infrastructure-manager:slice/<TenantID>/<SliceID>	Shows the information associated to the virtual optical slice.
GET	GET/restconf/config/virtual-infrastructure-manager:slice/<TenantID>	Shows the information associated to the virtual optical slices owned by a given tenant
PUT	PUT/restconf/config/virtual-infrastructure-	Updates or modifies the configuration of

	manager:slice/<TenantID>/<SliceID>	the specified virtual optical slice.
DELETE	DELETE/restconf/config/virtual-infrastructure-manager:slice/<TenantID>/<SliceID>	Removes the virtual optical slice and releases the occupied resources.

Table 10 – Create virtual optical slice specification

POST/restconf/config/virtual-infrastructure-manager:slice			
Request parameters	TenantID	integer	Identifier of the tenant
	EndPoints	List<Node>	List of the nodes that need to be included in the slice
	QoS	QoS object	List of the parameters that set the QoS requested for the optical slice
	MonitoringInfo (optional)	Monitoring object	Object that contains the parameters that need to be monitored in the optical slice
	Time (optional)	Time object	Parameters to set the start time and the duration of the optical slice
Response parameters	SliceID	integer	Identifier given to the slice created
	Topology	Topology object	Object that contains the topology of the created optical slice (i.e. list of nodes and links)
	Status	enum	Current status of the optical slice
Successful response codes: 201 CREATED			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 11 – Show virtual optical slice specification

GET/restconf/config/virtual-infrastructure-manager:slice/<TenantID>/<SliceID>			
Request parameters	TenantID	Integer	Identifier of the tenant of the slice
	SliceID	Integer	Identifier of the slice
Response parameters	Topology	Topology object	Object that contains the topology of the optical slice (i.e., list of nodes and links) identified by SliceID

	QoS	QoS object	List of the parameters that set the QoS requested for the optical slice identified by SliceID
	MonitoringInfo (optional)	Monitoring object	Object that contains the parameters that need to be monitored in the optical slice identified by SliceID
	Time (optional)	Time object	Parameters to set the start time and the duration of the optical slice identified by SliceID
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 12 – Show tenant's virtual optical slices specification

GET/restconf/config/virtual-infrastructure-manager:slice/<TenantID>			
Request parameters	TenantID	Integer	Identifier of the tenant of the slice
Response parameters	SliceInformationList	List<SliceInformation>	Object that contains the information of the virtual optical slice: SliceId, Topology, associated QoS, Time, etc.
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 13 – Update virtual optical slice specification

PUT/restconf/config/virtual-infrastructure-manager:slice/<TenantID>/<SliceID>			
Request parameters	TenantID	Integer	Identifier of the tenant of the slice
	SliceID	Integer	Identifier of the slice
	EndPoints (optional)	List<Node>	List of the nodes that need to be included in the slice
	QoS (optional)	QoS object	List of the parameters that set the QoS requested for the optical slice identified by SliceID

	MonitoringInfo (optional)	Monitoring object	Object that contains the parameters that need to be monitored in the optical slice identified by SliceID
	Time (optional)	Time object	Parameters to set the start time and the duration of the optical slice identified by SliceID
Response parameters	Status	Enum	Current status of the optical slice
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 405 METHOD NOT ALLOWED 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 14 – Delete virtual optical slice specification

DELETE/restconf/config/virtual-infrastructure-manager:slice/<TenantID>/<SliceID>			
Request parameters	TenantID	Integer	Identifier of the tenant of the slice
	SliceID	Integer	Identifier of the slice
Response parameters	-	-	-
Successful response codes: 204 NO CONTENT			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

3.3.2.3 Message exchange sequence

This section describes the messaging associated to the virtual optical slice provisioning and management operations. The Virtual Infrastructure Manager of the COSIGN SDN controller is responsible for managing the actions performed over the virtual optical slices. In brief, the Virtual Infrastructure Manager receives these messages from an external entity (such as the orchestrator or a tenant application) through the north-bound interface, acts as a trigger to perform such operations in the controller, and manages the virtual optical infrastructure as long as it is active. Figure 5 shows the messages that are exchanged for the creation, update and deletion of virtual optical slices.

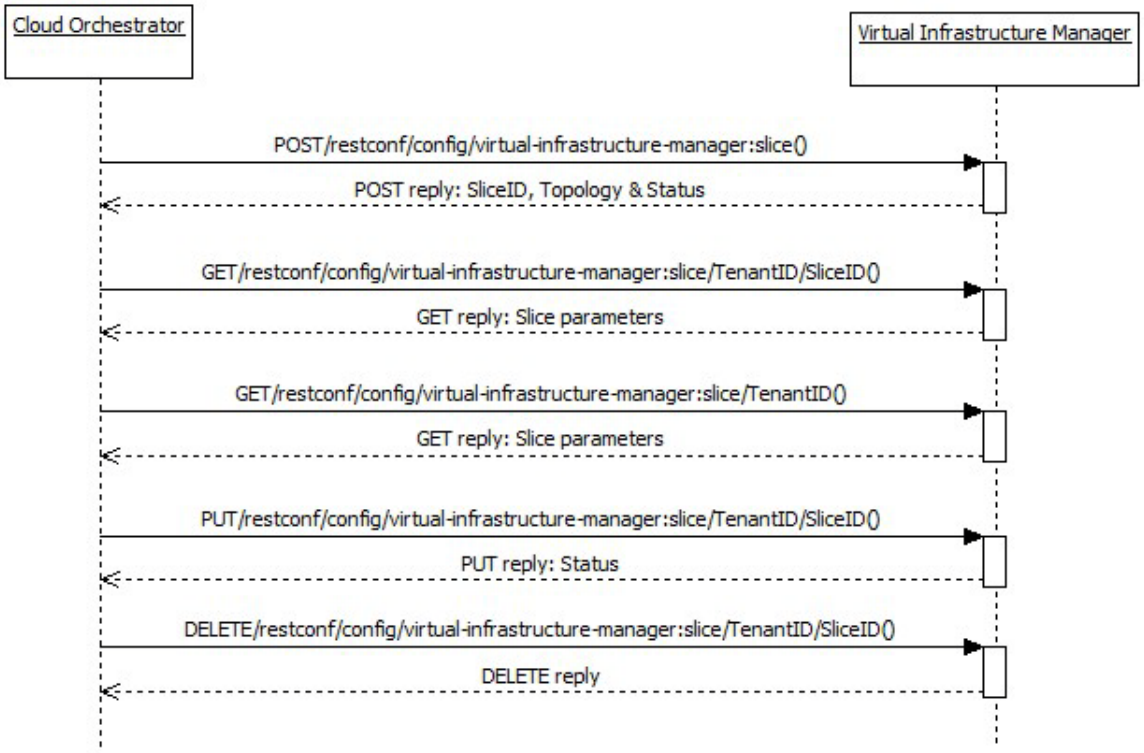

Figure 5 – Message sequence diagram for the virtual optical slice provisioning

To create a new virtual slice, a POST message is sent to the Virtual Infrastructure Manager, which starts the virtual optical infrastructure provisioning procedure as it is described in deliverable D3.1 [3]. This operation returns the identifier of the new optical slice as well as its related information (such as topology, QoS, etc.).

Two GET messages have been defined to retrieve the information related to the existing optical slices. The first one is aimed at providing the information of a specific slice (so the slice identifier has to be included in the request). The second one returns the information of all the slices owned by a specific tenant (so just the tenant identifier is required).

The characteristics of an optical slice can be modified or updated by means of a PUT message. In this case, the new parameters of the slice, such as new topology, QoS or time values, need to be included in the message.

Finally, the DELETE message is used to terminate an existing optical slice. While the terminated slice information can be kept in the controller, all the physical resources formerly associated with the slice are released and set as available, so that they can be used to fulfil forthcoming virtual optical infrastructure requests.

3.3.2.4 Matching with the reference SDN controller

The aforementioned APIs are based on OpenVirtex (OVX) API. OVX implements network virtualization as a proxy that sits between the physical network and the tenants' network operating system (e.g., SDN controller). OVX builds its view of the infrastructure in its physical network representation through topology discovery, and builds the representations of virtual networks with the aid of configuration information provided through its API. The OVX view of the infrastructure can be used as a starting point for the development of the virtual optical slice provisioning service. However, this needs to be properly adapted to support COSIGN optical devices features and configurations, e.g., mapping a virtual connection to one end-to-end optical OCS or TDM connection.

Update for D3.2-bis: After the analysis of alternative solutions for the use of OpenVirtex, which imposes the use of an additional and external software component between the Infrastructure

Controller and the physical network, the design of the Virtual Infrastructure Manager software has changed in order to enable a better integration with OpenDaylight. In particular the Virtual Infrastructure Manager will be implemented as an extension of the Virtual Tenant Network (VTN) application, which is an OpenDaylight project, and will be improved to support the virtualization of optical switches.

The main advantages of this choice are related to the simplification of the whole architecture, the more mature integration with Neutron and OpenStack that is offered by the VTN application and the fact that VTN supports both OpenFlow v1.0 and v1.3 through its integration with OpenDaylight. Both solutions would require the OCS extensions defined for the OpenFlow v1.0 in support of optical devices, but OpenVirtex would need additional extensions to support OpenFlow v1.3 to manage TUe's electronic switches. This choice does not impact the specification of the north-bound APIs of the service that are documented above.

3.3.3 Provisioning of optical connectivity

The provisioning of optical connectivity is the service in charge of establishing dynamic optical paths over the data centre network infrastructure, to provide the connectivity required to support overlay virtual networks or data centre management tasks which require the exchange of large bulks of data across the network.

The service is implemented by an internal module of the SDN controller and exposes a REST API, compliant with the RESTCONF protocol, to create, modify, retrieve and delete optical connections. The consumers of the service are the overlay-based network virtualization application, the cloud orchestrator or a management platform for the administration of the whole data centre.

3.3.3.1 Information model

Figure 6 shows the information model for an optical network connection.

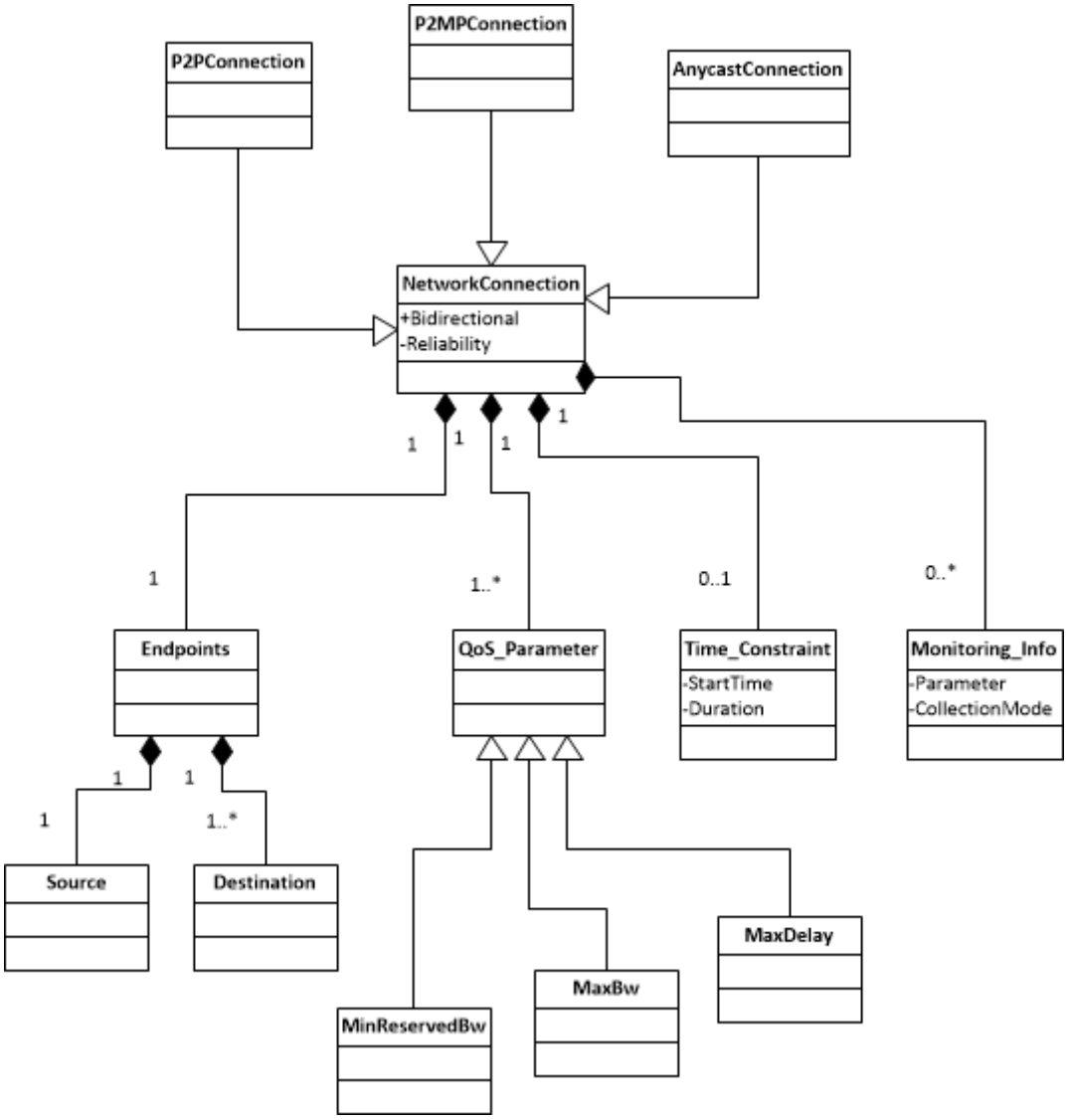

Figure 6 – Information model for optical connections

3.3.3.2 REST APIs

Table 15 – List of APIs for provisioning of optical connectivity

Operation	URI	Description
POST	/restconf/config/optical-provisioning-manager:connection	Creates a new optical connection
PUT	/restconf/config/optical-provisioning-manager:connection/connectionID	Modifies some parameters of an established optical connection with identifier "connectionID"
GET	/restconf/config/optical-provisioning-	Retrieves the details of an established optical

	manager:connection/connectionID	connection with identifier “connectionID”
GET	/restconf/config/optical-provisioning-manager:connections	Lists all the established optical connections
DELETE	/restconf/config/optical-provisioning-manager:connection/connectionID	Tears-down and removes the optical connection with identifier “connectionID”

Table 16 – POST /restconf/config/optical-provisioning-manager:connection specification

POST /restconf/config/optical-provisioning-manager:connection			
Request parameters	connection_type	enum	The type of the connection. The following options are available: point-to-point: point-to-multipoint: anycast:
	src_endpoint_type	enum	The type of the source endpoint (e.g., an IP address)
	src_endpoint	string	Source end point for the optical connection
	dst_endpoint_type	enum	The type of the destination endpoint (e.g., an IP address)
	dst_endpoint	list<string>	Destination end point for the optical connection. It is a single value for point-to-point connections and a list of values for point-to-multipoint and anycast connections. In the first case the connection is established between the source and all the destination endpoints, while for anycast connections a single destination is chosen from the pool.
	bidirectional	boolean	Indicates if the connection service is bidirectional
	recovery (optional)	enum	The recovery strategy to be adopted for the optical connection. The following options are available: **Unprotected:** in case of failures, the control plane does not apply any countermeasure. **Protected:** during the setup phase the control plane establishes an additional, possibly disjoint,

			connection where the traffic is transferred in case of failure on the primary path. **Restoration:** when the control plane detects a failure in a connection, it tries to establish dynamically a new working path.
	qos_parameters	qos object	Specification of the desired QoS. It includes the following optional elements: minimum_reserved_bw, max_bw, max_delay, packet_loss, class_of_service
	time_constraints (optional)	time constraints object	If specified, the connection will be established at the requested starting time and removed when the duration timer expires. If missing, the connection is immediately established and it will be removed through an explicit request. The object includes the following elements: **start_time:** time instant to activate the connection **duration:** duration time, in seconds, of the service
	monitoring_info (optional)	list<monitoring info object>	The list of parameters which can be monitored through the north-bound interface. Each monitoring object includes the following elements: **parameter:** type of parameter to be monitored (e.g., failures). **collection_mode:** mechanism to retrieve the monitored parameter. It can assume the following values: “polling”, “periodical”, “asynch”. **frequency (optional):** specified in case of periodical mode, it indicates the frequency of the notifications **threshold (optional):** specified in case of asynch mode, it indicated an expression which is evaluated to decide if triggering the event
Response parameters	connection_id	string	The unique identifier of the created connection
	status	enum	Current status of the connection
Successful response codes: 201 CREATED			

Error response codes: 400 BAD REQUEST, 403 FORBIDDEN 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE

Table 17 – PUT /restconf/config/optical-provisioning-manager:connection/connectionID specification

PUT /restconf/config/optical-provisioning-manager:connection/connectionID			
Request parameters	recovery (optional)	enum	The recovery strategy to be adopted for the optical connection. The following options are available: **Unprotected:** in case of failures, the control plane does not apply any countermeasure. **Protected:** during the setup phase the control plane establishes an additional, possibly disjoint, connection where the traffic is transferred in case of failure on the primary path. **Restoration:** when the control plane detects a failure in a connection, it tries to establish dynamically a new working path.
	monitoring_info (optional)	list<monitoring info object>	The list of parameters which can be monitored through the north-bound interface. Each monitoring object includes the following elements: **parameter:** type of parameter to be monitored (e.g., failures). **collection_mode:** mechanism to retrieve the monitored parameter. It can assume the following values: “polling”, “periodical”, “asynch”. **frequency (optional):** specified in case of periodical mode, it indicates the frequency of the notifications **threshold (optional):** specified in case of asynch mode, it indicated an expression which is evaluated to decide if triggering the event
	qos_parameters (optional)	qos object	Specification of the QoS parameters to be changed. The following optional elements can be accepted in a modification request: minimum_reserved_bw, max_bw.

	time_constraints (optional)	time constraints object	If specified, the starting time and/or the duration of the services to be modified. The object includes one or both the following elements: **start_time:** time instant to activate the connection **duration:** duration time, in seconds, of the service
Response parameters	status	enum	Current status of the connection modifications
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 405 METHOD NOT ALLOWED 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 18 – GET /restconf/config/optical-provisioning-manager:connection/connectionID specification

GET /restconf/config/optical-provisioning-manager:connection/connectionID			
Request parameters	--	--	--
Response parameters	connection_id	string	The unique identifier of the connection
	status	enum	Current status of the connection
	connection_type	enum	The type of the connection. The following options are available: point-to-point, point-to-multipoint, anycast
	src_endpoint_type	enum	The type of the source endpoint (e.g., an IP address)
	src_endpoint	string	Source end point for the optical connection
	dst_endpoint_type	enum	The type of the destination endpoint (e.g., an IP address)
	dst_endpoint	list<string>	Destination end point for the optical connection. It is a single value for point-to-point or anycast connections and a list of values for point-to-multipoint connections. In case of anycast connections

			specifies the single destination that has been selected.
	bidirectional	boolean	Indicates if the connection service is bidirectional
	recovery (optional)	enum	The recovery adopted for the optical connection. The following options are available: **Unprotected:** in case of failures, the control plane does not apply any countermeasure. **Protected:** during the setup phase the control plane establishes an additional, possibly disjoint, connection where the traffic is transferred in case of failure on the primary path. **Restoration:** when the control plane detects a failure in a connection, it tries to establish dynamically a new working path.
	qos_parameters	qos object	Specification of the applied QoS. It includes the following optional elements: minimum_reserved_bw, max_bw, max_delay, packet_loss, class_of_service
	time_constraints (optional)	time constraints object	If specified, the connection will be established at the requested starting time and removed when the duration timer expires. If missing, the connection has been immediately established and it needs to be removed through an explicit request. The object includes the following elements: **start_time:** time instant to activate the connection **duration:** duration time, in seconds, of the service
	monitoring_info (optional)	list<monitoring info object>	The list of parameters which have been activated for monitoring (if any) through the north-bound interface. Each monitoring object includes the following elements: **parameter:** type of parameter to be monitored (e.g. failures). **collection_mode:** mechanism to

			retrieve the monitored parameter. It can assume the following values: “polling”, “periodical”, “asynch”. **frequency (optional):** specified in case of periodical mode, it indicates the frequency of the notifications **threshold (optional):** specified in case of asynch mode, it indicated an expression which is evaluated to decide if triggering the event **current_value:** current value of the parameter
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 19 – GET /restconf/config/optical-provisioning-manager:connections specification

GET /restconf/config/optical-provisioning-manager:connections			
Request parameters	--	--	--
Response parameters	Connections	List<Connection_details object>	The description of the details for each connection established in the DCN. The format is the same used for the single connection.
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 20 – DELETE /restconf/config/optical-provisioning-manager:connection/connectionID specification

DELETE /restconf/config/optical-provisioning-manager:connection/connectionID			
Request parameters	--	--	--
Response parameters	--	--	--

Successful response codes: 204 NO CONTENT
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE

3.3.3.3 Message exchange sequence

The provisioning of DCN optical connections is a service implemented by a module in the COSIGN SDN controller called Optical Provisioning Manager and is typically invoked by external components, like the orchestrator or a cloud platform for the management of the DCN infrastructure. The sequence of messages exchanged between these entities during the provisioning and lifecycle of an optical connection is shown in Figure 7.

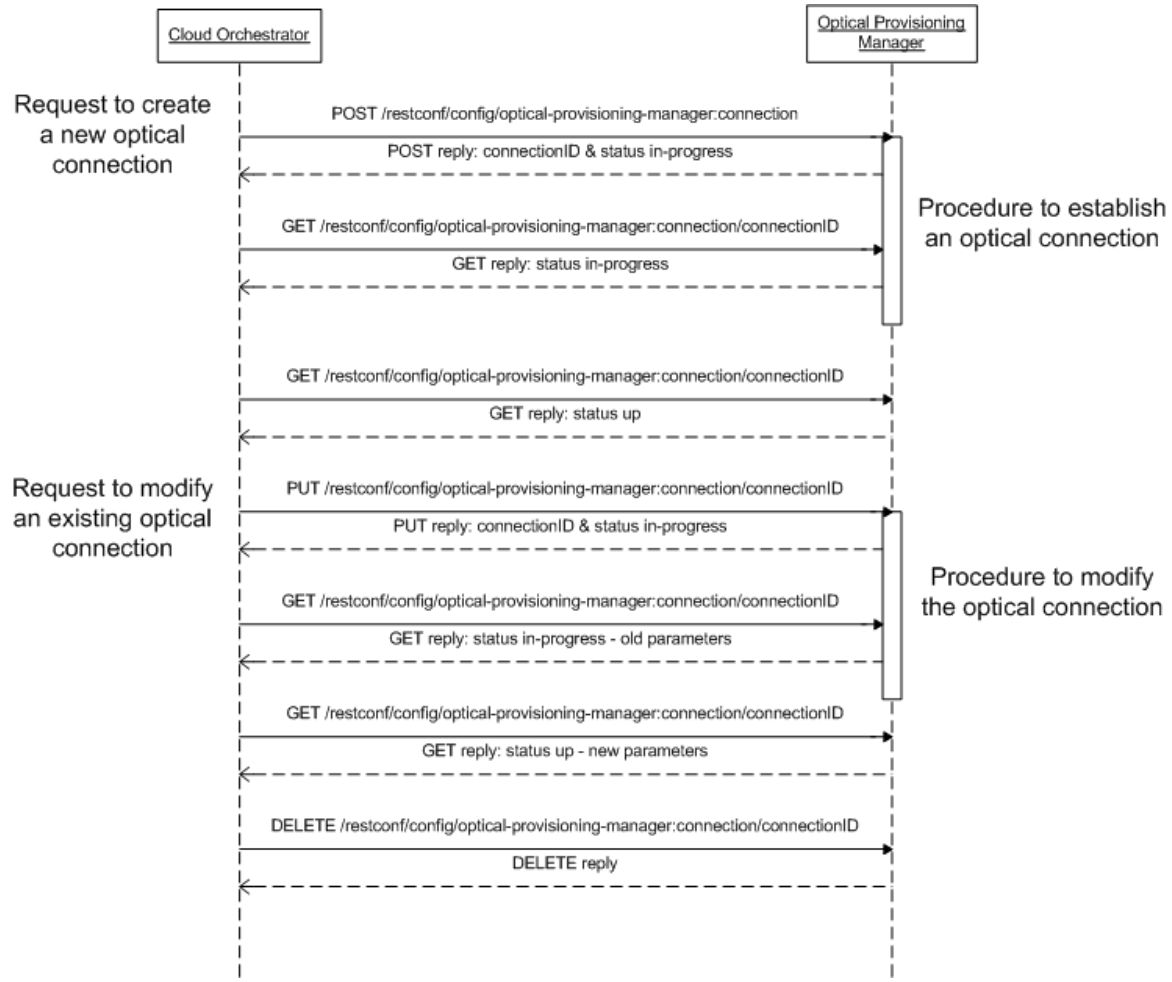

Figure 7 – Message sequence diagram for the provisioning of optical connections

The request of a new optical connection is triggered with a POST message that activates the procedures to setup the optical path on the SDN controller side. The details of the SDN controller internal processing to establish an optical connection are available in the workflows specified in deliverable D3.1 [3]. The POST reply returns the connection ID which identifies the optical connection resource and can be used in all the following messages to refer the specific connection. During the elaboration of the request, GET messages to retrieve information about the given connection will receive replies with status “in-progress”. Only once the connection is correctly established at the data plane, the status parameter becomes “up”. If something fails (option not shown

in the picture), the status becomes “failed-setup” and all associated data plane resources are dismissed and made available for new connections.

An active optical connection (in status “up”) can be modified through a PUT message. Again the status during the updating procedures is “in-progress” and becomes “up” after the successful modification of the parameters. In case of successful update, the following GET requests will receive the new parameters in the reply. However, if the modification fails, the old parameters are maintained and the status remains active. It is a task of the client to request the explicit deletion of the connection if required.

In general, an optical connection can be manually terminated using a DELETE message. In this case the resource is maintained in the SDN controller database, but the data plane resources are released and the status returned in the following GET replies is “terminated”. A terminated connection cannot be modified anymore. If the initial creation request (POST message) specified the time constraints (i.e., start time and duration), the optical connection is automatically terminated when the time-out expires.

3.3.3.4 Matching with the reference SDN controller

This service will be developed from scratch as an internal OpenDaylight plugin based on the MD-SAL architecture. The plugin will interact with other OpenDaylight modules. In particular it will communicate with the topology manager to retrieve information about the capabilities of the DCN nodes and the resource availability in the DCN links. Moreover, it will use the abstract methods exposed on the north-bound side of the OpenFlow plugin to configure the cross-connections on the data plane devices and establish the optical path.

3.3.4 Forwarding Rules Manager

The Forwarding Rules Manager defines the client flows which are to be transmitted over a given optical circuit identified by a *connectionID* (this circuit is previously created by the Optical Provisioning Manager through the APIs defined in section 3.3.3). The service translates the commands to add or remove client flows to/from a given optical circuit into the equivalent commands to add/remove flow entries in/from the optoelectronic switches at the edges of the optical circuit itself. In particular, the *connectionID* identifies the specific optical circuit, together with the corresponding end point nodes to be configured and the associated egress/ingress ports on the optical side. The *of_classification* identifies the traffic belonging to the given client flow and is translated into the corresponding OpenFlow match field. Finally, the resulting flow entries are configured in each end point using the OpenFlow plugin.

3.3.4.1 REST APIs

Table 21 – List of APIs for CRUD operations on forwarding rules

Operation	URI	Description
POST	/restconf/config/forwarding-rules-manager/connectionID	Creates a new forwarding rules for optical "connection ID" over specific end points
PUT	/restconf/config/forwarding-rules-manager/connectionID	Modifies all the forwarding rules for an established optical connection with identifier “connectionID”
PUT	/restconf/config/forwarding-rules-manager/connectionID/forwarding-rule-ID	Modifies the forwarding rules with identifier "forwarding-rule-ID" for an established optical connection with identifier “connectionID”
GET	/restconf/config/forwarding-rules-	Retrieves all the forwarding rules which

	manager/connectionID	identify client flows mapped over an established optical connection with identifier “connectionID”
GET	/restconf/config/forwarding-rules-manager/connectionID/forwarding-rule-ID	Retrieves a specific forwarding rule with identifier “forwarding-rule-ID” mapped over an established optical connection with identifier “connectionID”
DELETE	/restconf/config/forwarding-rules-manager/connectionID	Removes all the forwarding rules associated to client flows mapped over an optical circuit with identifier “connectionID”
DELETE	/restconf/config/forwarding-rules-manager/connectionID/forwarding-rule-ID	Removes a specific forwarding rule with identifier “forwarding-rule-ID” mapped over an established optical connection with identifier “connectionID”

Table 22 – POST /restconf/config/forwarding-rules-manager/connectionID

POST /restconf/config/forwarding-rules-manager/connectionID			
Request parameters	connection_id	string	The unique identifier of the created connection
	of_classification	string	The requested Open-Flow classification of the flows over the connection ID
Response parameters	Forwarding_rule_id	string	The unique identifier of the created forwarding rule.
	Status	enum	Current status of the client flow
Successful response codes: 201 CREATED			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 23 – PUT /restconf/config/forwarding-rules-manager /connectionID

PUT /restconf/config/forwarding-rules-manager/connectionID			
Request parameters	of_classification	string	The requested Open-Flow classification of the flows over the connection ID

Response parameters	status	enum	Current status of the flow modifications
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 405 METHOD NOT ALLOWED 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 24 – PUT /restconf/config/forwarding-rules-manager /connectionID/ forwarding-rule-ID

PUT /restconf/config/forwarding-rules-manager/connectionID/forwarding-rule-ID			
Request parameters	of_classification	string	The requested Open-Flow classification of the flow with identifier "forwarding-rule-ID" over the connection ID
Response parameters	status	enum	Current status of the flow modifications
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND, 405 METHOD NOT ALLOWED 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 25 – GET /restconf/config/forwarding-rules-manager/connectionID

GET /restconf/config/forwarding-rules-manager/connectionID			
Request parameters	--	--	--
Response parameters	connection_id	string	The unique identifier of the connection
	{forwarding-rule-ID; of_classification}	Dictionary	Dictionary for mapping the the unique identifiers of all of the forwarding rules and their corresponding of_ classifications
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 26 – GET /restconf/config/forwarding-rules-manager/connectionID/forwarding-rule-ID

GET /restconf/config/forwarding-rules-manager/connectionID/forwarding-rule-ID			
Request parameters	--	--	--
Response parameters	connection_id	string	The unique identifier of the connection
	forwarding-rule-ID	string	The unique identifier of the forwarding rule
	of_classification	string	The requested Open-Flow classification of the flow identified by the forwarding rule over the connection ID
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 27 – DELETE /restconf/config/forwarding-rules-manager/connectionID

DELETE /restconf/config/forwarding-rules-manager/connectionID			
Request parameters	--	--	--
Response parameters	--	--	--
Successful response codes: 204 NO CONTENT			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 28 – DELETE /restconf/config/forwarding-rules-manager/connectionID/forwarding-rule-ID

DELETE /restconf/config/forwarding-rules-manager/connectionID/forwarding-rule-ID			
Request parameters	--	--	--

Response parameters	--	--	--
Successful response codes: 204 NO CONTENT			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

3.3.5 DCN information services

The DCN information service allows for collecting of information about the status and the performance of the network infrastructure or the network services (i.e., optical connections, virtual network slices or overlay networks) established over the DCN. Statistics and monitoring information are thus associated to different level of resources, from physical nodes or ports to more abstracted resources as optical paths or virtual networks. The access to the information service is thus regulated through well-defined policies, so that different types of user have visibility on a limited set of data. For example, the access to data plane statistics is reserved to the DCN administrator only, while monitoring data about a specific optical connection or virtual network are available for the tenants owning those resources (i.e., the user who has requested them). Moreover, to enable a certain level of mutual awareness between the IT and the network side of the data centre, some abstract information about the physical network capabilities and utilization can be disclosed to selected applications operating on the north-bound of the SDN controller, including the cloud orchestrator.

The basic mechanism to collect monitoring data from the COSIGN control plane is based on the polling model, where REST clients use GET messages to request statistics and monitoring data on demand. In OpenDaylight, these monitoring parameters can be modelled as read-write or read-only structures in the YANG model (i.e., a *container*), which describe the associated service and are stored in the OpenDaylight internal configuration or operational data store. In general, they can be retrieved with a GET message to URLs structured as *"/restconf/config/<module>:<container>"* or *"/restconf/operational/<module>:<container>"*, depending on the data store where they are located. On the other hand, some services already available in OpenDaylight allow to access the data with Remote Procedure Calls (RPCs), still defined in the YANG model. In this case, they can be retrieved with a POST message to URLs structured as *"/restconf/operations/<module>:<rpc>"*. The OpenDaylight statistics service, which is a good candidate for the provisioning of some information for the DCN service, follows this approach.

However, the polling mode is not suitable in cases where the client needs to be informed of specific events occurring in real-time. For example, an unrecoverable failure in an optical path needs to be notified immediately to the applications which manage the overlay virtualization, so that they can take appropriate action for the impacted overlay networks and limit the service disruption suffered by the user. In OpenDaylight, the *notification* statement of the YANG language can be used to model asynchronous notification events triggered directly from the SDN controller and delivered to the applications within a notification event stream. At the REST API this type of interaction is handled through web sockets.

The following specification of the DCN information service REST APIs is focused on the monitoring of the physical network infrastructure, while the options for the monitoring of the COSIGN control plane services (e.g., the provisioning of optical connections) are specified in the definition of the related interfaces in the previous sections.

3.3.5.1 Information model

The information model for the DCN monitoring service is mostly derived from the model of the statistics defined in the OpenFlow specification [OF], with group, table, port, queue, meter, individual

and aggregate flow statistics. However, it has been extended to include specific statistics for the optical ports, as shown in Figure 8.

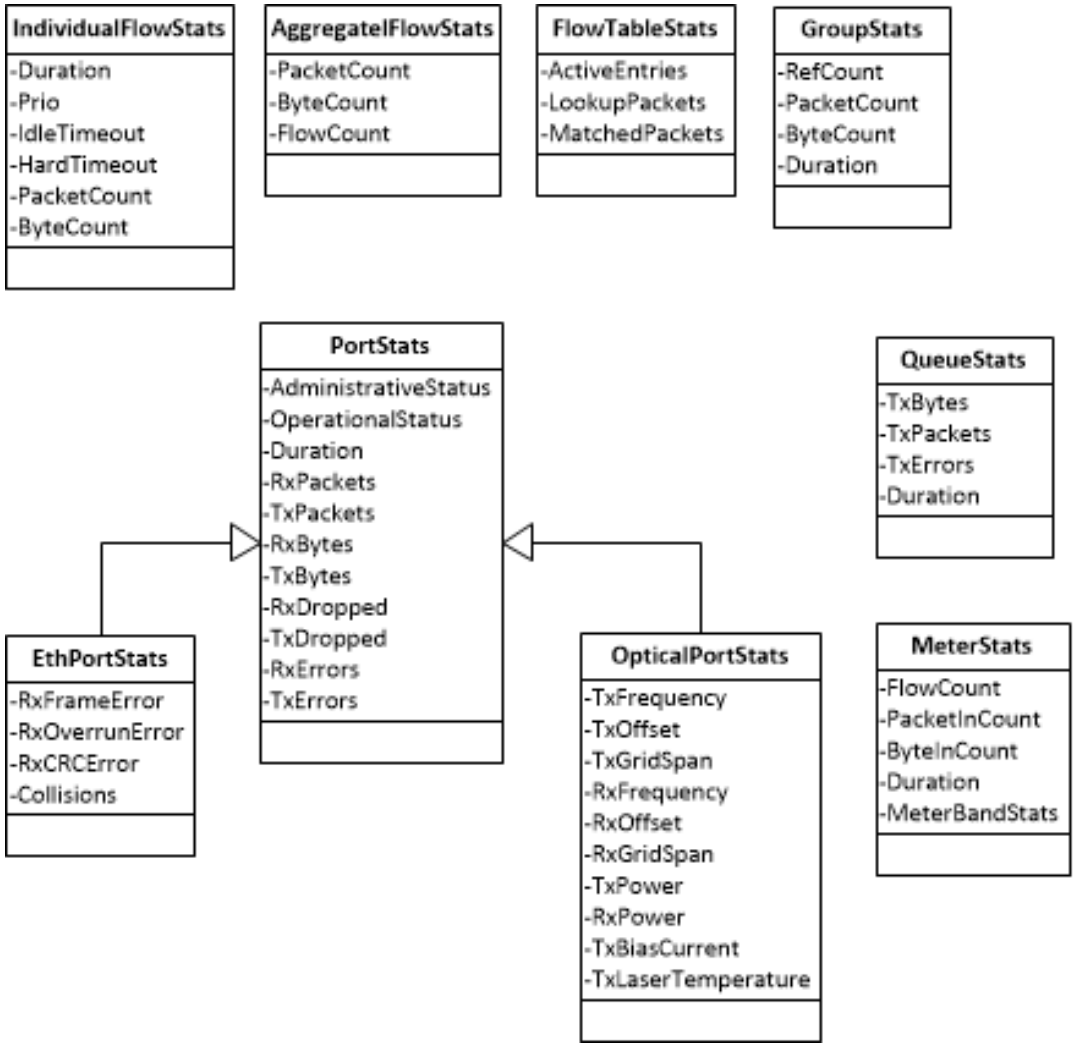

Figure 8 – Information model for the DCN information service

3.3.5.2 REST APIs

Table 29 – List of APIs for the DCN information service

Operation	URI	Description
GET	/restconf/config/opendaylight-inventory:nodes	Retrieve all the nodes available in the network, each of them with the list of its ports.
POST	/restconf/operations/opendaylight-port-statistics:get-all-node-connectors-statistics	Retrieve statistics about all the ports in a node
POST	/restconf/operations/opendaylight-port-statistics:get-node-connector-statistics	Retrieve statistics about a given port in a node
POST	/restconf/operations/opendaylight-flow-statistics:get-all-flows-	Retrieve statistics about all the flows in a node

	statistics-from-all-flow-tables	
POST	/restconf/operations/opendaylight-flow-statistics:get-flow-statistics	Retrieve statistics about a given flow in a node
POST	/restconf/operations/opendaylight-flow-table-statistics:get-flow-tables-statistics	Retrieve statistics about all the tables in a node
POST	/restconf/operations/opendaylight-queue-statistics:get-all-queues-statistics-from-given-port	Retrieve statistics about all the queues in a port

Table 30 – GET */restconf/config/opendaylight-inventory:nodes specification*

GET /restconf/config/opendaylight-inventory:nodes			
Request parameters	--	--	--
Response parameters	Nodes	List<Node object>	The description of the nodes, including their NodeId and their ports.
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 31 – POST /restconf/operations/opendaylight-port-statistics:get-all-node-connectors-statistics specification

POST /restconf/operations/opendaylight-port-statistics:get-all-node-connectors-statistics			
Request parameters	NodeId	String	The unique identifier of the node.
Response parameters	Ports	List<Port object>	The list of the ports. Each Port object includes the Port Id, the Port Type and the PortStats, as defined in the information model above.
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 32 – POST /restconf/operations/opendaylight-port-statistics:get-node-connector-statistics specification

POST /restconf/operations/opendaylight-port-statistics:get-node-connector-statistics			
Request parameters	NodeId	String	The unique identifier of the node.
	PortId	String	The unique identifier of the port.
Response parameters	PortType	Enum	Type of the port: optical or eth.
	PortStatistics	PortStats object	Port statistics. The actual parameters depend on the specific type of the port. See the information model in section 3.3.5.1 for the whole specification.
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 33 – POST /restconf/operations/opendaylight-flow-statistics:get-all-flows-statistics-from-all-flow-tables specification

POST /restconf/operations/opendaylight-flow-statistics:get-all-flows-statistics-from-all-flow-tables			
Request parameters	NodeId	String	The unique identifier of the node.
Response parameters	Flows	List<Flow object>	The list of the flows in a node. Each Flow object includes: • Flow Id • Match • Instruction • IndividualFlowStats, as specified in the UML diagram in section 3.3.5.1
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 34 – POST /restconf/operations/opendaylight-flow-statistics:get-flow-statistics

POST /restconf/operations/opendaylight-flow-statistics:get-flow-statistics			
Request parameters	NodeId	String	The unique identifier of the node.
	TableId	String	The unique identifier of the table.
	FlowId	String	The unique identifier of the flow.
Response parameters	Match	Match object	The classifier configured for the flow.
	Instruction	List<Instruction object>	The list of instruction configured for the flow.
	FlowStatistics	FlowStats object	The flow statistics, following the format specified in section 3.3.5.1.
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 35 – POST /restconf/operations/opendaylight-flow-table-statistics:get-flow-tables-statistics specification

POST /restconf/operations/opendaylight-flow-table-statistics:get-flow-tables-statistics			
Request parameters	NodeId	String	The unique identifier of the node.
Response parameters	Tables	List<Table object>	The list of the tables in a node. Each Table object includes: • Table Id • TableStats, as specified in the UML diagram in section 3.3.5.1
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

Table 36 – POST /restconf/operations/opendaylight-queue-statistics:get-all-queues-statistics-from-given-port

POST /restconf/operations/opendaylight-queue-statistics:get-all-queues-statistics-from-given-port			
Request	NodeId	String	The unique identifier of the node.

parameters			
	PortId	String	The unique identifier of the port.
Response parameters	Queues	List<Queue object>	The list of the queues configured for the given port. Each Queue object includes: • Queue Id • QueueStats, as specified in the UML diagram in section 3.3.5.1
Successful response codes: 200 OK			
Error response codes: 400 BAD REQUEST, 403 FORBIDDEN, 404 NOT FOUND 500 INTERNAL SERVER ERROR, 503 SERVICE UNAVAILABLE			

3.3.5.3 Matching with the reference SDN controller

The DCN information service can be based on the Inventory and Statistics service already available in OpenDaylight as components of the MD-SAL. However they require extensions to include the additional parameters specified for the optical devices, with particular reference to the statistics for optical ports (see section 3.3.5.1 for the information model) and additional options for Match and Action parameters in the specification of the flows. These two last aspects are described in more details in section 4.4.2, which specifies the OpenFlow extensions for the configuration of the data plane.

3.3.6 Management services: configuration of SLAs and policies

The COSIGN Infrastructure Control Layer exposes north-bound APIs for management services such as configuration of policies and SLAs. The two control plane entities responsible for implementing the functionality for the management services are the *Policy Manager* and *SLA Manager* respectively (Figure 9). The role of the policy and SLA manager entities is to receive the policies/SLAs from upper layer entities (e.g., cloud management platform) and process them accordingly (e.g., store, validate, enforce, etc.).

In the context of a DCN, a *policy* is a rule through which the DC operator controls/specifies the behaviour for a set of network resources. The target resource for a policy may be a data plane resource (e.g., network function, links, etc.) or a control plane component (e.g., routing, monitoring, etc.).

An *SLA* captures the agreement between the DC operator and a customer. Examples of parameters included in an SLA are: guaranteed QoS, service downtime, etc. An SLA refers to network resources that are allocated to a customer.

The Infrastructure Control Layer contains a *data store* where all the data is kept in the form of a data object hierarchy. Hence, the configured policies and SLAs are also kept in the data store (Figure 9).

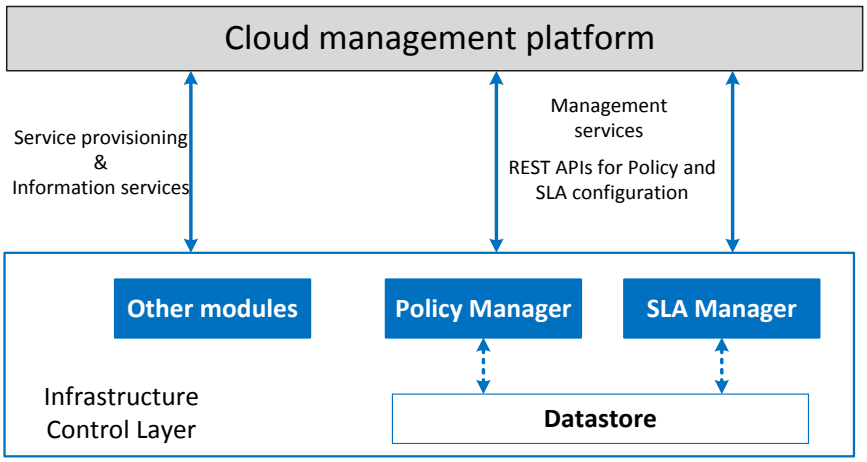

Figure 9 – North-bound APIs for the management interface

3.3.6.1 Information model

Figure 10 shows an UML representation for the information model that captures policy-related resources in the data store. A *Policy* object comprises a *Match* object which identifies the target resource, and an *Action* object which defines the specific action to be enforced on that resource. There can be various types of policies (*PolicyImpl),* which implement the *Policy* interface. Because policies may have different scopes, they are grouped in containers (*PolicyContainer).* A specific implementation (*PolicyContainerImpl*) of the *PolicyContainer* interface defines a logical group of policies that relate to each other, possibly requiring conflict resolution among them.

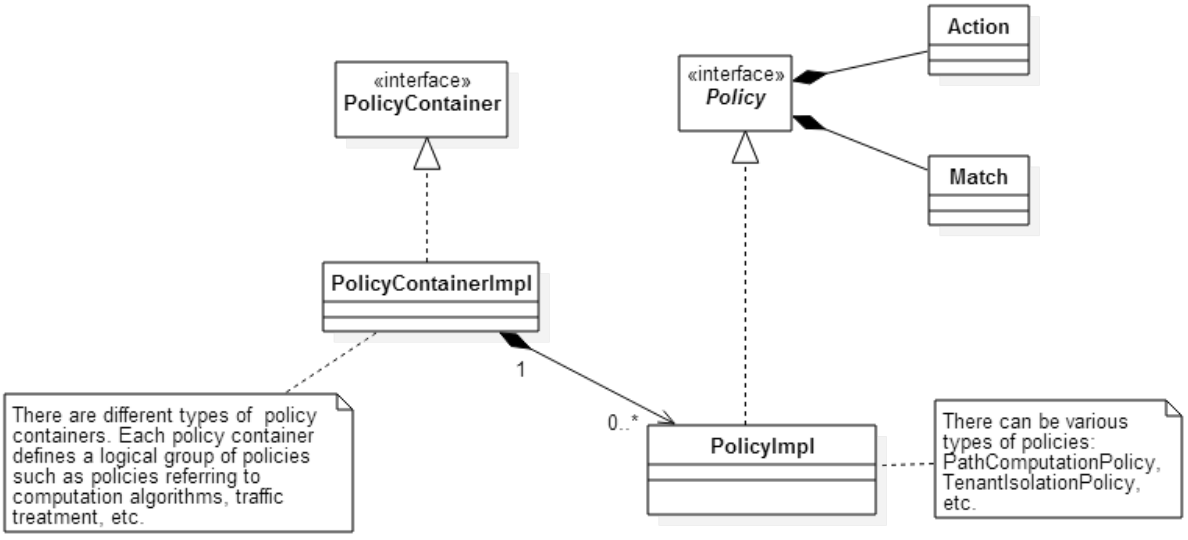

Figure 10 – An information model for policy data objects

Figure 11 shows a similar information model but for SLA data objects. A *ServiceLevelAgreement* object has a similar structure as a *Policy* object (i.e., with *Match* and *Action*). SLAs are associated with tenants, because they capture the agreements between a tenant (customer) and the DC operator. Hence, a *Tenant* object contains a set of specific SLAs (*SlaImpl*), which must be enforced on the resources allocated to the tenant.

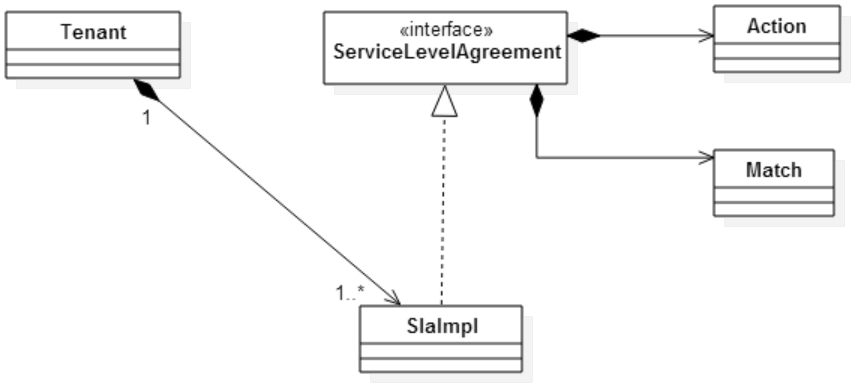

Figure 11 – An information model for SLA data objects

3.3.6.2 REST APIs

Since the reference SDN controller in COSIGN is OpenDaylight, the definition of the REST APIs is aligned with the existing OpenDaylight REST APIs, which are based on the RESTCONF protocol. The RESTCONF URIs reflect the internal structure of the data store due to the automatic creation of the APIs (based on the YANG data model definition).

The data store contains configuration data and operational data. The policies and SLAs are configuration data hence the URI includes the */config/* part. Next in the URI is the name of the specific module considered, which is either */policy/* or */sla/* (Table 37). For the case of a policy, the last part of the URI identifies the policy container, which contains a list of policies (Figure 10). For SLAs, the last part of the URI denotes the tenant, which contains a list of SLAs (Figure 11).

The management services discussed in this section are based on CRUD (Create, Read, Update, Delete) operations, which are applied to data objects (policies and SLAs) that reside in the data store. Therefore, even if the RESTCONF protocol supports it, there is no RPC communication or event notification for the REST APIs described herein.

Table 37 – List of APIs for management of SLAs and policies

Operation	URI	Description
GET	/restconf/config/policy:<policy_container>	Retrieve policies configured in the specified policy container
GET	/restconf/config/policy:<policy_container>/<policy_id>	Retrieve target policy
POST	/restconf/config/policy:<policy_container>	Add a new policy for to policy container
PATCH	/restconf/config/policy:<policy_container>/<policy_id>	Merge policy into existing policy within policy container. Allows updating sub-resources within the targeted resource. E.g. update the actions for an existing policy.
PUT	/restconf/config/policy:<policy_container>/<policy_id>	Update the entire target policy to a new policy.
DELETE	/restconf/config/policy:<policy_container>/<policy_id>	Delete target policy

GET	/restconf/config/sla:<tenant>	Retrieve SLAs for tenant
GET	/restconf/config/sla:<tenant>/<sla_id>	Retrieve SLA with ID for tenant
POST	/restconf/config/sla:<tenant>	Add a new SLA to tenant
PATCH	/restconf/config/sla:<tenant>/<sla_id>	Merge SLA into existing SLA for a given tenant.
PUT	/restconf/config/sla:<tenant>/<sla_id>	Update the target SLA to a new SLA
DELETE	/restconf/config/sla:<tenant>/<sla_id>	Delete target SLA

Some of the REST requests must carry in their message body a textual description of a policy or an SLA. For example adding a new policy using the POST method must also convey the description of the new policy inside the message body.

3.3.6.3 Matching with the reference SDN controller

The reference SDN Controller (SDNC) for COSIGN is OpenDaylight (ODL). There are several projects in ODL that are related to the management of services described in this section: ***Group-based Policy*** [ODL-POL], ***Service Function Chaining*** plugin [ODL-SFC], and ***Network Intent Composition*** plugin [ODL-NIC]. However, after careful investigation of the possibility for re-using parts of the ODL projects mentioned above for the implementation of the policy and SLA managers, it has been decided that it is best to develop these two COSIGN modules from scratch. Therefore, the policy and the SLA manager modules will not re-use any of the existing ODL modules. New YANG models will be defined for policies, and new ODL modules for policy and SLA managers will be created based on these YANG models.

The main reason for the decision not to reuse pre-existing elements in ODL, related to policy-based functionality, is that the ODL modules mentioned above have a narrow scope, which is particularly related to defining policies for network connectivity between endpoints or groups of endpoints. On the other hand, the policy and SLA modules for COSIGN have a much wider scope, including other aspects, apart from network connectivity policies. Some of these are policies for controlling the different provisioning algorithms, policies for controlling the orchestration of resources, etc. Basically, the COSIGN policy and SLA managers will be able to handle not only policies related to resource provisioning between endpoints, but also various other configuration policies that allow administrators to easily manage the data centre network (i.e., data plane and control plane logic).

4 Specification of south-bound interface

In a generic SDN context, the South-bound interface provides access to data plane elements for configuration, control and monitoring through standardised protocols in order to cope with the heterogeneity of data plane elements and vendors. In the following sections a description of the intended elements for the implementation of the south-bound interface in the specific COSIGN architecture is provided.

4.1 Reference technologies and standards

The main function of the SDN control plane south-bound interface is to enable communication between the SDN controller and the network nodes (both physical and virtual switches or routers) so that the devices can discover network topology, define network flows and implement requests relayed to it via north-bound APIs. So, a south-bound controller plugin is a functional component that does the following:

1. Provides an abstraction of network devices functionality
2. Normalizes their APIs to common contracts
3. Handles session and connections to them

With the discussion above, COSIGN will choose OpenDaylight as the reference SDN controller as its south-bound interface is capable of supporting multiple protocols (as shown in Figure 12). Each of the South-bound Plugins serves a different purpose, with some overlapping. For example, the OpenFlow plugin might serve the Data-Plane needs of an OVS element while the OVSDB plugin can serve the management plane needs of the same OVS element [ODL-wiki]. For COSIGN purpose, OF protocol is selected as the south-bound protocol, so in this section we only focus on two plugins, as presented in Table 38: OpenFlow and OVSDB.

Figure 12 – South-bound plugin supported in ODL Lithium release

Table 38 – Plugins Description

South-bound Plugins	Description
OpenFlow	The OpenFlow plugin project intends to develop a plugin to support implementations of the OpenFlow specification as it develops and evolves. Specifically the OpenDaylight OpenFlow plugin project has developed a plugin aiming to support OpenFlow 1.0 and 1.3.x.
OVSDB	OVSDB (Open vSwtich Database) is a management protocol used to manipulate the configuration of Open vSwitches. The OpenDaylight OVSDB south-bound plugin consists of one or more OSGi bundles addressing the following services or functionalities : 1. Connection Service - Based on Netty 2. Network Configuration Service 3. Bidirectional JSON-RPC Library 4. OVSDB Schema definitions and Object mappers

	5. Overlay Tunnel management 6. OVSDB to OpenFlow plugin mapping service 7. Inventory Service

Both OpenFlow v1.0 and v1.3 are supported in OpenDaylight Lithium. Network devices supporting either of these versions of the protocol can be supported concurrently by the controller. The OpenFlow protocol specification already has some extensions for optical resources which can be re-used to partially model the COSIGN data plane resources, e.g., OpenFlow v1.0 with circuit extensions v0.3 and OpenFlow v1.4. However, these protocol versions are not supported in the release OpenDaylight OpenFlow plugin. Thus, the COSIGN OpenFlow plugin development will add support for these extensions to the release version. More details of the extensions will be presented in the following sections.

Also, in general, the development process consists of following steps:

1. Definition of YANG models (API contracts): For Model-Driven SAL, the API contracts are defined by YANG models and the Java interfaces generated for these models. In this phase, developers select existing models which they want, write own models or augment (extend) existing ones.
2. Code Generation: Java Interfaces, implementation of Transfer Objects and mapping to Binding-Independent form is generated for the plugin. This phase requires the proper configuration of the Maven build and YANG Maven Tools.
3. Implementation of plugin: The actual implementation of the plugin functionality and the plugin components.

4.2 OpenFlow agents and OpenFlow drivers

The control layer (i.e., Infrastructure Control Layer) in COSIGN architecture operates on a unified, abstract view of the DCN. However, there are several technologies that are used in the DCN (e.g., optical/electrical, different vendors). In order to create a homogeneous representation of the network it is necessary to abstract these technologies. This function is accomplished by the Abstraction layer (Figure 13), which creates a common abstraction of the DCN by utilizing the functions available at the D-CPI. OpenFlow (OF) is the chosen D-CPI protocol in COSIGN. Two of the most important entities which are used to create the DCN abstraction are the OF agents and the OF drivers.

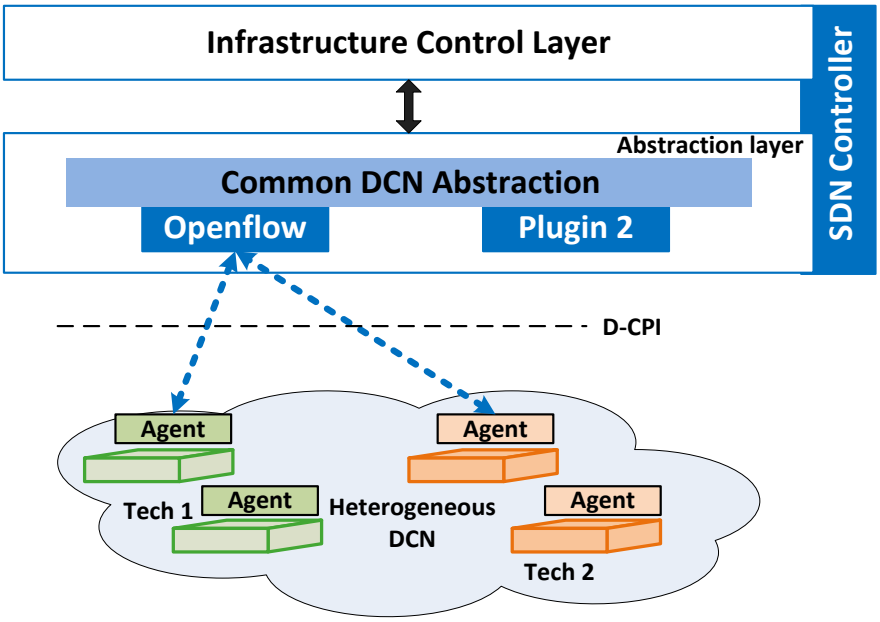

Figure 13 – Interaction between OpenFlow agents and drivers

4.2.1 OpenFlow agents

OF agents are entities that bridge between the capabilities of a network device and the capabilities offered by OF at the D-CPI. An OF agent operates on top of each network device that has to be managed by the control layer through OF (Figure 13). The relationship between an agent and a device is 1:1. The specific functions that an OF agent fulfils are:

- Technology specific mapping and translation: the agent maps the received OF protocol commands into devices specific commands (realized by the adaptation layer, Figure 14)
- Uniform resource model: the agent exposes a technology independent abstraction of the capabilities of the device. This allows for the control layer to have a uniform representation of different optical technologies (realized by the adaptation layer, Figure 14).
- Extended OF API: the agent implements the OF communication with the control layer and possible extensions to the OF protocol (Figure 14).

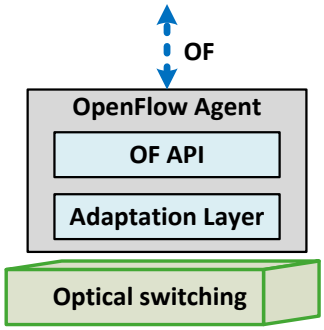

Figure 14 – Generic architecture for an OF agent

4.2.2 OpenFlow drivers

An OF driver is a plugin that realizes the OF communication with the DCN devices, and it operates at the lowest level in the SDN controller (Figure 13). Some of the functions that the OF driver must fulfil are:

- OF Connectivity with DCN: the OF driver must maintain connectivity with all the devices that are operated through OF in the DCN (connection management in Figure 15). The relationship between the OF driver and the DCN devices is 1:N.

- Extended OF API: the OF driver implements the OF message exchange with possible extensions specific to optical technologies used in the COSIGN data plane (Figure 15).

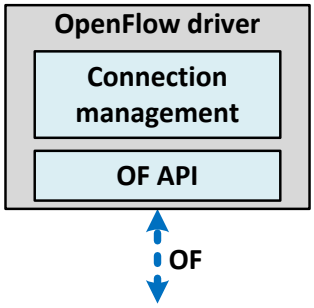

Figure 15 – Generic architecture for an OF driver

4.3 Modelling of physical resources

4.3.1 High Radix Top of Rack switch with mid-board optics

The TOR switch built by TU/e in collaboration with PhotonX is using a layer 2/3 Ethernet switch/router provided by Broadcom. Broadcom supports the use of OpenFlow through a dedicated software stack as described in Figure 16:

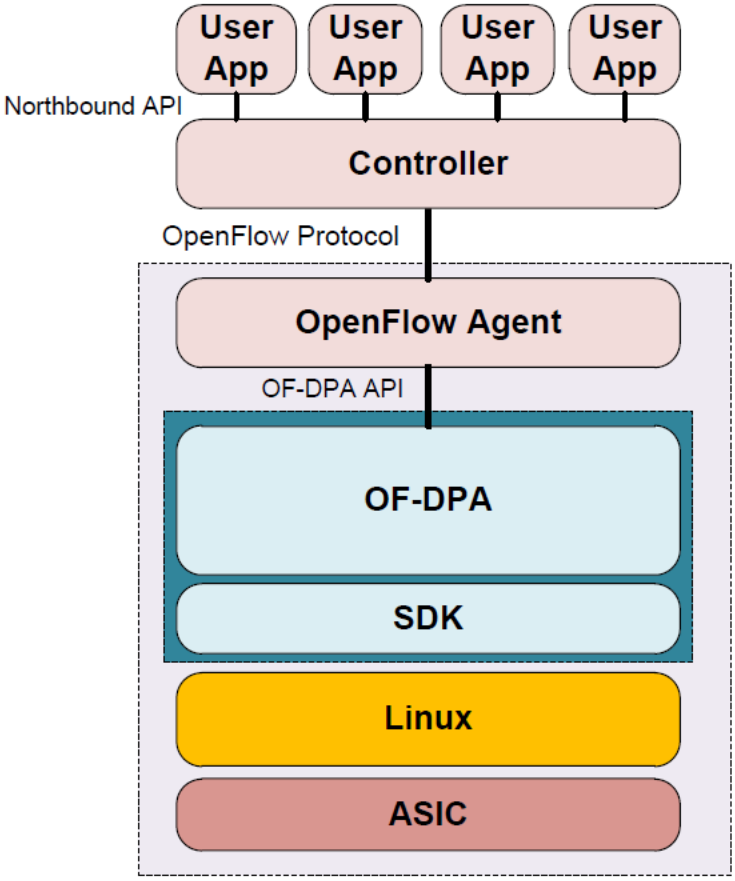

Figure 16 – Broadcom OFDPA component layering

The OF-DPA API, as defined in the Open Flow Data Plane Abstraction (OF-DPA [OFDPA]) API Guide and Reference Manual, presents a specialized Hardware Abstraction Layer (HAL) that allows programming Broadcom ASICs using OpenFlow abstractions. It does not, however, process OpenFlow protocol messages. To create a complete OpenFlow switch using OF-DPA, an OpenFlow agent is required. In addition, an OpenFlow Controller (in our case the COSIGN SDN controller) is

required to field an OpenFlow network deployment using OF-DPA-enabled switches. Figure 16 illustrates the relationship of OF-DPA with the other OpenFlow system components.

In the COSIGN project the agent implemented on top of the OFDPA API is Indigo. More details on indigo are available in section 4.5 and in Indigo github repository, available at https://github.com/floodlight/indigo.

4.3.2 Optical ToR

The capability of optical ToR, its resources ready to be configured and the cross connection are modelled with UML and presented in Figure 17.

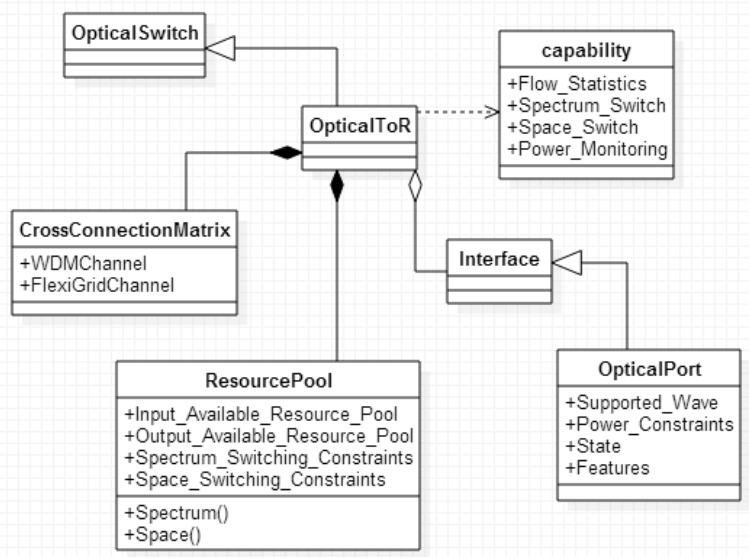

Figure 17 – Data Model of Optical ToR

Capability: it is assumed that the capability items of the OpenFlow (logical) switch have been configured as part of instantiation of these switches. Element <Flow_Statistics> defines the statistics capability of the Optical ToR switch, especially for the connection duration time. Element <Spectrum_Switch> and <Space_Switch> identifies the supported switching dimension of the ToR switch, which depends on the exact device features. Element <Power_Monitoring> denotes the supported optical signal power monitoring capability of port. More general definitions of these capability items can be found in OpenFlow Specification 1.3.

Interface: Optical Port

- Element <Supported_Wave> defines the supported wavelength of the optical port or presents as a tunable TX.
- Element <Power_Constraints> identifies the (Min and Max) optical signal power tolerance (e.g., db) of the port.
- Element <State> represents various elements of known state of port and contains three further elements: <oper-state>, <live>. Element <oper-state> represents the reported link state of the port and must have a value of either “up” or “down”. Element <live> must have a value of either “true” or “false”. A value of “true” means the port is active and sending/receiving signals.
- Element <Features> contains a list of OpenFlow Port Features which contains four sub-lists represented by elements <current>, <advertised>, <supported> and <advertised-peer>. These four lists must contain the features associated with the OpenFlow Port. The specific semantics of feature membership in each of these four sublists are defined in the OpenFlow protocol.

Resource Pool: This element abstracts/represents the resource availability (with elements <Input_Available_Resource_Pool> and <Output_Available_Resource_Pool>) and switching constraints (with <Space_Switching_Constraints and <Spectrum_Switching_Constraints>).

Specifically, the switching constraints include both design-inherited and resource availability induced limitation. From the resource allocation point of view, input and output resource could be allocated in <Spectrum> and <Space> dimension.

CrossConnectionMatrix: This element represents the existing optical cross connection settings with <WDMChannel> or <FlexiGridChannel> (depends on different scenarios of ToR switch).

4.3.3 Optical NIC

The capability of optical NIC and resources to be configured are modelled with UML and presented in Figure 18, as well as the forwarding rules.

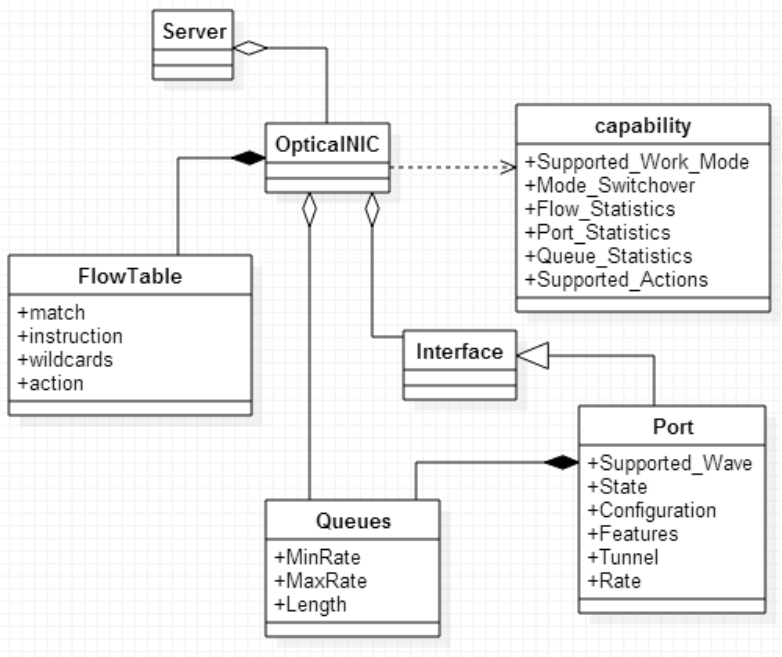

Figure 18 – Data Model of Optical NIC

Capability: Element <Supported_Work_Mode> identifies the supported working mode of NIC, including Ethernet and TDM mode (sending the packet in time slot way). Element <Mode_Switchover>.indicates that the NIC could dynamically change the working mode without any packet loss. Element <Supported_Actions> specifies the action types which could be merged into the current action set of flow entries of the flow table. Element <Flow_Statistics>, <Port_Statistics> and <Queue_Statistics> define the statistics capability (e.g., byte/bit count) of the Optical NIC which are optional in COSIGN approach since the statistics is a used for traffic engineering purpose, not a basic function.

Interface: Port

- Element <Supported_Wave> defines the supported wavelength of the optical port or presents as a tunable TX.
- Element <State> represents various elements of known state of port and contains two further elements: <oper-state>, <live>. Element <oper-state> represents the reported link state of the port and must have a value of either “up” or “down”. Element <live> must have a value of either “true” or “false”. A value of “true” means the port is active and sending/receiving signals.
- Element <Features> contains a list of OpenFlow Port features which contains four sub-lists represented by elements <current>, <advertised>, <supported> and <advertised-peer>. These four lists must contain the features associated with the OpenFlow Port. The specific semantics of feature membership in each of these four sublists are defined in the OpenFlow protocol.

- Element <Configuration> identifies several configuration options: <no-receive>, <no-packet-in> and <no-forward>. Element <admin-state> represents the reported admin state of the port and must have a value of either “up” or “down”. These three configuration options must have a value of either “true” or “false”.
- Element <Tunnel> enables the association of logical OpenFlow ports with an associated tunnel type and corresponding parameters for the tunnel.
- Element <Rate> identifies the current sending/receiving speed of the port.

Queues: The OpenFlow Queue is an instance of an OpenFlow resource. It contains list of queue properties. Element <port> associates an OpenFlow Queue with an OpenFlow Port. In COSIGN approach, the implementation of queue related monitoring/configuration is optional and nice to have for service QoS guarantee.

- Element <MinRate> must indicate the minimum rate of the queue by percentage as an integer representing one tenth of one percent.
- Element <MaxRate> must indicate the maximum rate of the queue by percentage as an integer representing one tenth of one percent.
- Element <Length> must indicate the instant utilization of the queue by percentage as an integer representing one tenth of one percent.

FlowTable: an OpenFlow Flow Table is identified by identifier within the context of the OpenFlow Capable Switch and OpenFlow Logical Switches.

- Element <match> denotes the types of match fields supported by the flow table. These match fields are defined in OpenFlow Specification version 1.3. An OpenFlow Logical Switch is not required to support all match field types and supported match field types don’t need to be implemented in the same table lookup.
- Element <wildcards> specifies the fields for which the table supports wildcarding (omitting).
- Element <instruction> denotes the types of flow instructions supported by the flow table. Flow instructions associated with a flow table entry are executed when a flow matches the flow entry in the flow table.
- Element <action> specifies the action types which could be merged into the current action set of flow entries of the flow table.

4.3.4 Large scale switch

The capability of the Polatis optical switch and its resources to be configured are modelled with UML and presented in Figure 19.

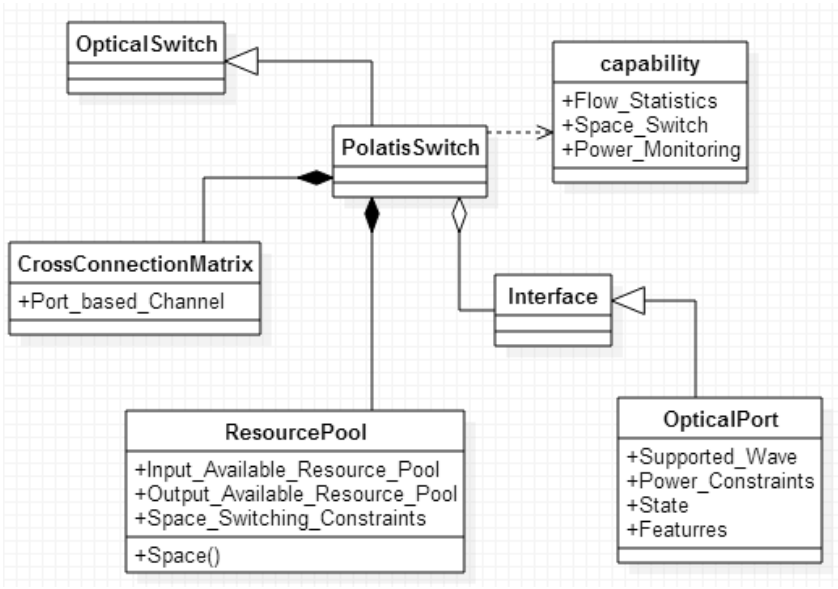

Figure 19 – Data Model of Polatis Switch

Capability: the definition of element <Flow_Statistics> and <Power_Monitoring> are the same as the corresponding definition for Optical ToR switch. Element <Port_Switch> identifies that this switch supports space switching.

Interface: the definition of optical port is the same as that for Optical ToR switch.

Resource Pool: This element abstracts/represents the resource availability and switching constraints using the following elements <Space_Switching_Constraints>. From the resource allocation point of view, the resource could be allocated only in <Space> dimension.

CrossConnectionMatrix: This element represents the existing optical cross connection settings with <Port_based_Channel>.

4.3.5 Fast switch

The capability of the Venture fast switch, its resources to be configured and cross connection matrix are modelled with UML and presented in Figure 20.

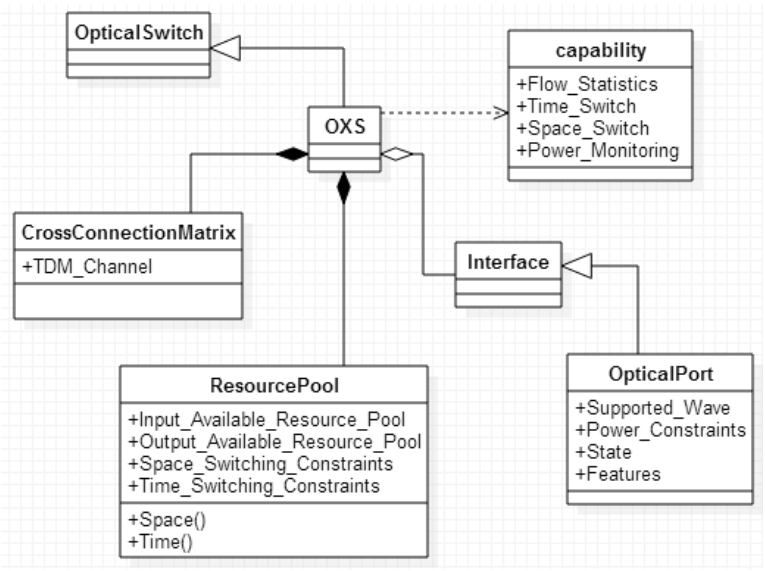

Figure 20 – Data Model of OXS Fast Switch

Capability: the major functional difference between OXS fast switch with the previous Polatis switch is that fast switch could support switching in time dimension (besides space dimension), so element <Time_Switch> is defined to advertise this capability.

Interface: the definition of optical port is the same as that for Optical ToR switch.

Resource Pool: besides the input/output resource pool and space dimension switching constraints, <Time_Switching_Constraints> is added to show the resource availability in time dimension. From the resource allocation point of view, the resource could be allocated in combined <Space> and <Time> dimension.

CrossConnectionMatrix: This element represents the existing optical cross connection settings with <TDM_based_Channel>. Generally, to properly describe a TDM channel, channel granularity and star time slot are required.

4.4 OpenFlow protocol extensions

The existing OF specification version 1.3.4 provides a set of capabilities for managing devices in DCNs. However, the DCN proposed in COSIGN will include new optical devices and new functionality for existing devices. Therefore, it is necessary to define a set of extensions to support the operations required by the control plane to manage the DCN. The extensions presented in this section augment the OF version 1.0 specification [OF] using the optical extensions document [OF-CS] and extensions taken from OF 1.4 .

Section 4.3 describes how the physical resources are modelled and section 4.4 discusses the OF protocol extensions required to control and manage the physical devices as they are presented in section 4.3. A set of extensions will be defined to support the capabilities captured in the models 4.3. These extensions are separated in three categories in sections 4.4.1, 4.4.2, and 4.4.3.

The most important addition of the extensions is the idea of a circuit port, as opposed to the normal packet port. This is semantically different within OpenFlow, as packets may not be inspected to create flows with traditional Ethernet matches, but circuits do not support packet inspection. Instead, the extensions enable switching in the optical space, time and frequency domains for devices supporting circuit switching.

4.4.1 Device-controller initial synchronization

In the OF protocol synchronization messages are sent in either controller-switch or switch-controller directions. Some messages initialise or verify a controller-switch connection and may also be used to measure the connection's latency or bandwidth. In COSIGN, we consider the OF optical extensions document [OF-CS] and OF v1.4 as a source of useful extensions, many of the messages could be reused directly, e.g., Hello, Echo, Error and Set config. However, because the reference version of OpenDaylight, Lithium, supports v 1.0 and v1.3, we backport only those features considered useful.

Furthermore, the controller must request the capabilities of a switch by sending a features request; the switch must respond with a features reply that specifies the capabilities of the switch. This is commonly performed upon establishment of the OpenFlow channel. OF v1.4 includes extensions to fields to specify the working frequencies of transmitter and receiver that we will add to Lithium. However, to support all the new device features in COSIGN, we need to further extend the features reply message to enable advertising of the new features, as presented in Table 39.

Table 39 – Extensions to Features Advertising

Features advertised		Extension	Descriptions
Switch features	uint32_t capabilities	OFPC_TDM_SWITCH = 1 << 9	Capabilities supported by specified switch.
		OFPC_PORT_SWITCH = 1 << 10	
		OFPC_WDM_SWITCH = 1 << 11	
		OFPC_FLEX_GRID_SWITCH = 1 << 12	

Port Features	Optical port	uint32_t tx_grid_freq_lmda	if this switch support flex-grid switching, this field should report the supported superchannel resolution	TX Grid Spacing Frequency Wavelength
		uint32_t rx_grid_freq_lmda	if this switch support flex-grid switching, this field should report the supported superchannel resolution	RX Grid Spacing Frequency Wavelength
		unit16_t min_tdm_slot	*x* ns	Supported minimum time slot size if this switch supporting TDM switch: e.g., larger than several ns

4.4.1.1 Message exchange sequence

Figure 21 presents the required message exchange in the controller-switch initial stage. The five main asynchronous message types are described as follows:

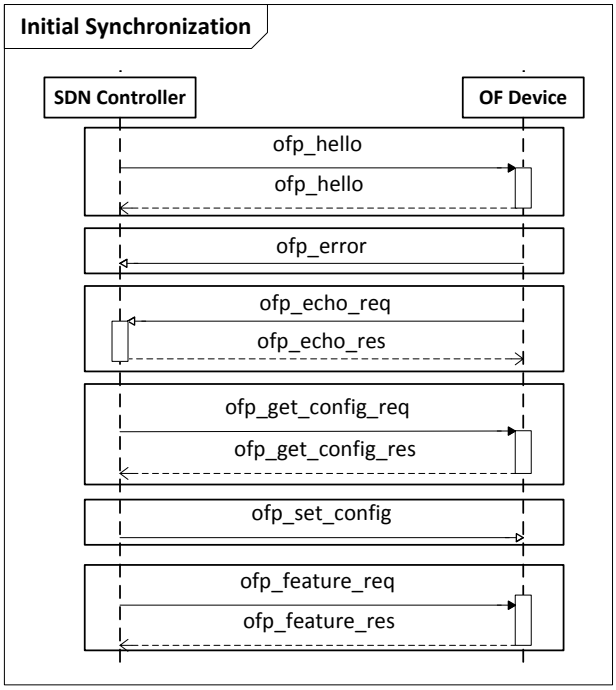

Figure 21 – Initial Synchronization Message Exchange Sequence

- *Hello* is used by either controller or switch during connection setup for version negotiation. If the version negotiation fails, an Error message is sent with type HelloFailed.
- *Error* can be sent by either the switch or the controller and indicates the failure of an operation. All error messages begin with the standard OpenFlow header, containing the appropriate version and type values, followed by the error structure. Type indicates the high level nature of the error and the combination of type and code determines the detailed content of the error.
- *EchoReq/EchoRes* is used to exchange information about latency, bandwidth and liveness. Echo request timeout indicates disconnection.
- *GetConfigReq/GetConfigRes/SetConfig* are used to query and set the fragmentation handling properties of the packet processing pipeline. GetConfigReq is an acknowledged message

(GetConfigRes) and is only initiated by the controller. SetConfig can only be initiated by the controller, and is unacknowledged.

- *FeatureReq/FeatureRes* is used to implement feature determination. The controller sends a FeatureReq to the switch and the switch will respond with its capabilities. Besides the information included in the OF specification, some extension have been proposed to report the capability of COSIGN new devices (as described in Table 39).

4.4.2 Data plane configuration

Data plane configuration refers to configuration of the data plane to process the traffic in a specific way, e.g., wavelength switching, creating overlays. This further translates into configuration of the flow tables, optical devices, etc. This section contains the OF extensions for configuration of the flow tables of the optical devices that are proposed in the COSIGN project.

A. Extensions for Ethernet switches

OF v1.3.4 enables a rich set of features to be implemented for Ethernet switches. However, if advanced overlay, QoS or OAM mechanisms must be supported, it is necessary to extend the functionality of OF v1.3.4. The following extensions are proposed for this purpose, according to [OFDPA].

a) *Extensions for match headers*. Beyond the standard match header fields presented in table 12 in [OF], the following extensions are proposed:

MAC_DST: is used for building a bridging table according to [OFDPA] page 167.

MPLS_L2_Port: identifies an MPLS pseudo wire endpoint.

MPLS_TTL: identifies the MPLS TTL.

MPLS_DATA_FIRST_NIBBLE: identifies if the packet has a pseudo wire MPLS control word (PWMCW) header or an Associated Channel Header (ACH) by matching on the first nibble.

MPLS_ACH_CHANNEL: identifies the pseudo wire ACH type.

NEXT_LABEL_IS_GAL: identifies if the next label in the stack is one of the MPLS reserved labels: GAL (Generic associated channel label). This label is used for denoting an OAM channel that is multiplexed in the MPLS pseudo wire.

VLAN_DEI: matches on the Drop Eligibility Indicator (DEI) field from the VLAN tag.

VRF: metadata that can be used to identify a virtual routing table (e.g., multicast routing table, unicast routing table, etc.).

ACTSET_OUTPUT: identifies the egress output port set by previous flow tables or group tables. It is used only in the egress flow table.

LMEP_Id: local identifier for the MEP or MIP.

OAM_Y1731_OPCODE: identifies the OAM PDU operational code.

OAM_Y1731_MDL: identifies the OAM PDU Maintenance Domain Level.

COLOR_ACTIONS_INDEX: pipeline metadata which denotes an entry in the colour-based actions flow table as specified in [OFDPA].

TxFCl: pipeline metadata indicating the number of transmitted OAM packets. This is sent to an application which performs network protection [OFDPA].

RxFCl: pipeline metadata indicating the number of received OAM packets. This is sent to an application which performs network protection [OFDPA].

RxTIME: Timestamp value for the current OAM packet. This metadata is sent to the network application in charge of performing network protection [OFDPA].

Protection_Index: indicates if the packet is destined for the protection or working path.

b) Extensions for OF actions

OFPAT_CKT_OUTPUT: indicates that the packet flow must be extracted from a circuit.

The corresponding data structure is [OFP-CS]:

```
struct ofp_action_ckt_output {
      uint16_t type;     /* OFPAT_CKT_OUTPUT */
      uint16_t len;      /* Length is 24 */
      uint16_t adaptation;    /* Adaptation type - one of OFPCAT_* */
      uint16_t cport;        /* Real or virtual OFPP_* ports */
          /* Define the circuit port characteristics if necessary */
      uint64_t wavelength;    /* use of the OFPCBL_* flags */
      uint32_t tsignal;    /* one of the OFPTSG_* flags. Not valid if
                              used with ofp_connect for TDM signals */
      uint16_t tstart;     /* starting time slot. Not valid if used
                              with of_connect for TDM signals */
      uint16_t tlcas_enable;   /* enable/disable LCAS */
};
```

OFP_ASSERT(sizeof(struct ofp_action_ckt_output) == 24);

OFPAT_CKT_INPUT: indicates that the packet flow must be inserted into a circuit.

The corresponding data structure is [OFP-CS]:

```
struct ofp_action_ckt_input {
      uint16_t type;          /* OFPAT_CKT_INPUT */
      uint16_t len;          /* Length is 24 */
      uint16_t adaptation;    /* Adaptation type - one of OFPCAT_* */
      uint16_t cport;         /* Real or virtual OFPP_* ports */
          /* Define the circuit port characteristics if necessary */
      uint64_t wavelength;      /* use of the OFPCBL_* flags */
      uint32_t tsignal;      /* one of the OFPTSG_* flags. Not valid
                            if used with ofp_connect for TDM signals */
      uint16_t tstart;       /* starting time slot. Not valid if used
                             with of_connect for TDM signals */
      uint16_t tlcas_enable;   /* enable/disable LCAS */
};
```

OFP_ASSERT(sizeof(struct ofp_action_ckt_input) == 24);

Set QoS From Table: sets the QoS index and the MPLS_TC from the shim header of a packet.

Pop CW or ACH: pops the CW header or the ACH. It is used only when popping the bottom of the stack pseudo wire MPLS label.

Pop L2 Header: used to pop the outer-most Ethernet header.

OAM_LM_RX_Count: takes as argument the LMEP_Id (the Id of the local maintenance endpoint), and indicates that the OAM counters must be updated for that MEP [OFDPA].

Check-Drop-Status: used to instruct the switch to check the packet against the Drop-Status table [OFDPA].

COPY_TC_IN: used to instruct the switch to copy the MPLS_TC from the outer to an inner label for a packet.

B. Extensions for SDM/WDM/TDM

These extensions are commands targeted for OF-enabled devices that perform SDM/WDM or TDM switching [OFP-CS].

a) SDM

ofp_connect: takes as arguments *uint16_t* ***in_port*** and *uint16_t* ***out_port***. It instructs the switch to create the connection between the input and output ports.

b) TDM

ofp_connect: takes as arguments *struct ofp_tdm_port* ***in_tport*** and *struct ofp_tdm_port* ***out_tport***, and it instructs the switch to create the connection between the two TDM ports, ***in_tport*** and ***out_tport***. The description of the TDM port is:

```
struct ofp_tdm_port
{
      uint16_t tport; /* port numbers in OFPP_* ports */
      uint16_t tstart; /* starting time slot */
      uint32_t tsignal; /* one of OFPTSG_* flags */
};
```

The *tsignal* field is one of the flags specified in the enumeration below, and it defines the minimum switching granularity for TDM.

```
enum ofp_tdm_gran
{
      OFPTSG_STS_1, /* STS-1 / STM-0 */
      OFPTSG_STS_3, /* STS-3 / STM-1 */
      OFPTSG_STS_3c, /* STS-3c / STM-1 */
      OFPTSG_STS_12, /* STS-12 / STM-4 */
      OFPTSG_STS_12c, /* STS-12c / STM-4c */
      OFPTSG_STS_48, /* STS-48 / STM-16 */
      OFPTSG_STS_48c, /* STS-48c / STM-16c */
      OFPTSG_STS_192, /* STS-192 / STM-64 */
      OFPTSG_STS_192c, /* STS-192c / STM-64c */ OFPTSG_STS_768, /*
      STS-768 / STM-256 */
      OFPTSG_STS_768c /* STS-768c / STM-256c */
};
```

c) WDM

of_connect: takes as arguments *struct ofp_wave_port* ***in_wport*** and *struct ofp_wave_port* ***out_wport***, and it instructs the switch to create the connection between the two WDM ports, ***in_wport*** and ***out_wport***. The description of the WDM port is:

struct *ofp_wave_port*

{

uint16_t wport; /* port numbers in OFPP_* ports */

uint8_t pad[6]; /* align to 64 bits */

uint64_t wavelength; /* use of the OFPCBL_* flags */

};

Using this command, the switch is instructed by the control plane to establish a cross connection with specified port number and wavelength. This is used in the optical ToR in the COSIGN DCN.

C. Extensions for optical port configuration

These OF extensions are used by the control plane to perform configuration of switch optical ports. The command used to configure switch ports is *ofp_port_mod*. This command contains a data structure defining the wanted configuration for the port. For the case of an optical port this configuration must include the following fields:

uint32_t ***configure***: bitmap of OFPOPF_* as described below

uint32_t ***freq_lmda***: the centre frequency.

uint32_t ***grid_span***: the size of the grid for the port.

uint32_t ***tx_pwr***: transmission power.

uint32_t ***start_freq/lmda:*** start frequency for spectrum attenuation setting.

uint32_t ***end_freq/lmda***: end frequency for spectrum attenuation.

uint8_t ***attenuation:*** port or spectrum based attenuation (0 mean port attenuation).

The ***configure*** field describes optical features that can be configured for the port. Any of the fields below can be configured:

```
enum ofp_optical_port_features {
      OFPOPF_RX_TUNE = 1 << 0, /* Receiver is tunable */
      OFPOPF_TX_TUNE = 1 << 1, /* Transmit is tunable */
      OFPOPF_TX_PWR = 1 << 2, /* Power is configurable */
      OFPOPF_USE_FREQ = 1 << 3, /* Use Frequency, not wavelength */
};
```

4.4.2.1 Message exchange sequence

Figure 22 shows four cases for OF operations that may involve the extensions defined in section 4.4.2 for match header fields, actions, creation of connectivity, and port configuration.

Case 1

This case illustrates two OF messages (Figure 22): a *Packet In* message arriving at the controller (from the OF device), which carries a match structure conveying information about the headers of the packet, and a *Packet Out* message, which carries the actions to be applied to the packets. These two messages may contain extensions for matching headers and actions as defined in section 4.4.2.

Case 2

In the second case, the *Flow Mod* message (Figure 22) contains both match header fields and actions data structures, thus it may contain any of the extensions described in section 4.4.2 subsection A.

The *Flow Mod* message can be sent by the controller in a proactive way, or as a reply to a *Packet In* message sent by the OF device. As mentioned, the data structures which convey information about the actions and the matching fields in the OF messages captured in Figure 22 (case 1 and 2), may be based on the OF extensions defined for COSIGN (section 4.4.2), in addition to the basic OF protocol (v 1.3.4).

Case 3

The third case depicts a new type of OF message: *ofp_cflow_mod* (*CFlow Mod*). This type of message is defined in [OFP-CS] and it is used to configure a cross connect table inside an optical switch. The *ofp_connect* data structure (defined in section 4.4.2), which conveys connectivity configuration, is carried as an action inside the *ofp_cflow_mod* OF message (case 3 in Figure 22). Hence, the OF extensions for SDN/WDM/TDM, belong to this case.

Case 4

In order to configure a port, the SDN controller must send a *Port Mod* OF message to the OF device (case 4 in Figure 22). The extensions defined in section 4.4.2 for optical port configuration belong to this type of OF message.

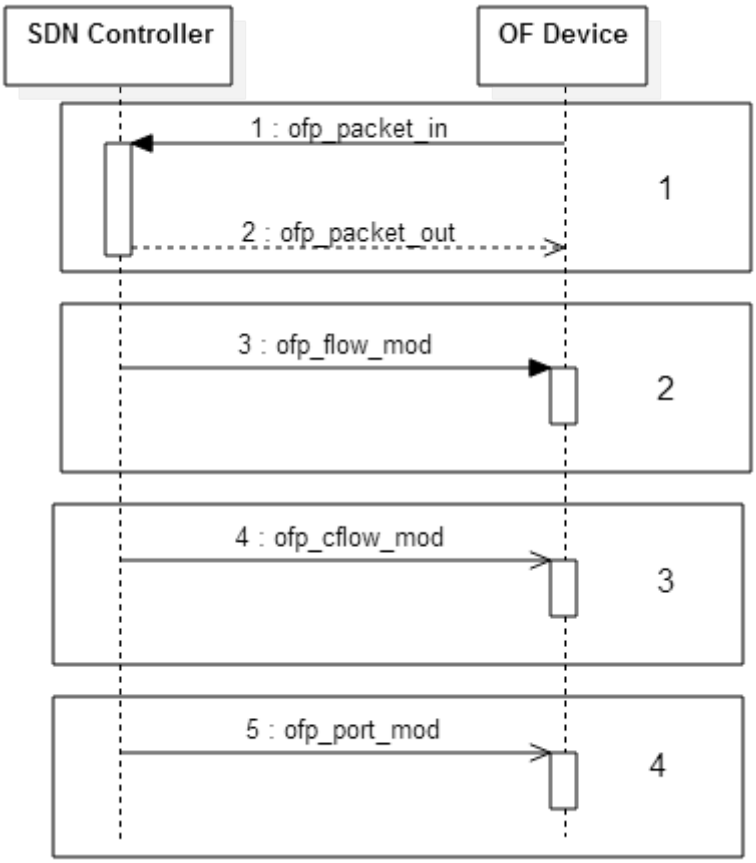

Figure 22 - Example of Packet out and Flow mod operations

4.4.3 Monitoring and statistics

Besides reading the features of switches and their ports, there are several messages used by the SDN controller to collect device information from the switch, such as statistics, or triggered by a switch, e.g., trap-like messages to signal device events. In general, there are three types of messages:

- Switch event monitoring: allows a controller to keep track of changes to the flow tables in a multi-controller scenario or in the event of hardware failure.
- Meter: used as a switch element that can measure and control the rate of packets. The meter triggers a meter band if the packet rate or byte rate passing through the meter exceeds a predefined threshold.
- Statistics collection: used to request statistics from a switch, e.g., statistics of flows, table, port and queues.

In the COSIGN scenario, the following extensions are proposed with respect to the monitoring, as shown in Table 40, and statistics messages as shown in Table 41. More details are presented in the following section.

Table 40 Extension to switch event monitoring messages

Monitoring capability		Extensions	Descriptions
Flow	request	struct ofp_connect connect	Add fields to match with optical circuit connection
	reply	uint16_t power	Add fields to report the power of specified optical circuit connection

Table 41 Extension to Statistics messages

Statistics		Extensions	Descriptions
Flow	request	struct ofp_connect connect	Add fields to match with optical circuit connection

4.4.3.1 Message exchange sequence

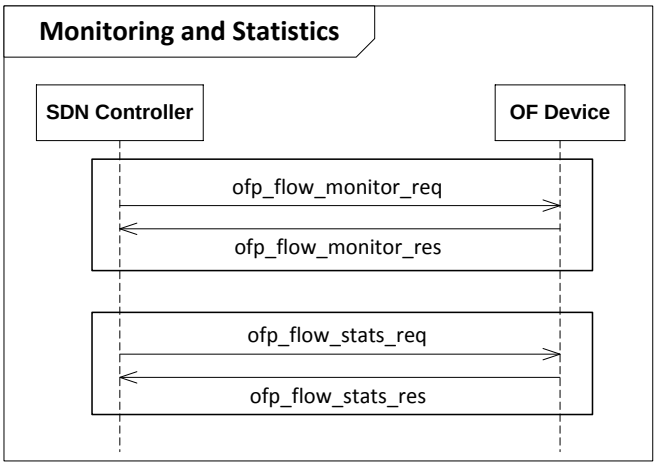

Figure 23 – Flow Monitoring and Statistics Message Exchange Sequence

FlowMonitoringReq/FlowMonitoringRes allows a controller to define a number of monitors, as shown in Figure 23, these can only be initiated by the controller, each selecting a table id and a match pattern that defines the subset monitored. When any flow entry is added, modified or removed in one of the subsets defined by a flow monitor, an event is sent to the controller to inform it of the change. In COSIGN, this message is extended to report the signal power of a specified circuit flow channel, which is helpful for the control plane to understand the channel performance in optical layer.

FlowStatsReq/FlowStatsRes is defined to request statistics information about individual and aggregate flows. FlowStatsReq can only be initiated by the controller. Also, it is extended to report the statistics for a specified circuit channel, e.g., duration.

4.4.4 Origin of OpenFlow Extensions

In Table 42 we list each of the individual OpenFlow extensions and the specification from which the extensions originated.

Table 42 – Origin of OpenFlow extensions deployed in COSIGN

Extension Type	Extension Name	Origin	Notes
1. Optical Device features advertising	Capabilities – type of optical switch, e.g., TDM, WDM	COSIGN	Table 39 – Extensions to Features Advertising
	TX grid spacing freq / lambda	COSIGN	
	RX grid spacing frequency / lambda	COSIGN	
	Minimum TDM slot	COSIGN	

2. Ethernet Devices Configuration	Match Features	OF [OFDPA]	See p. 63, 4.4.2 Data plane configuration
3.Packet-Circuit Input and Output	Conversions to and from circuit to packet flows	OF [OFP-CS]	See p. 64, *Extensions for OF actions*
4.Monitoring & Statistics	Circuit flow event monitoring	COSIGN	See p. 67, 4.4.3 Monitoring and statistics
5.Optical Flow Programming	SDM, TDM and WDM parameters Circuit input and output	[OFP-CS]	See p. 65, 4.4.2 Data plane configuration

4.4.5 Relation of physical resource models to extensions

Table 43 describes the relation between the physical resource models in Section 4.3 and the extensions specified in Section 4.4.

Table 43 – Mapping between OpenFlow extensions and information models of COSIGN devices

Extension type	**Extension name**	**Reference structure in physical resource models**
1. Optical Device features advertising	Capabilities – type of optical switch e.g. TDM, WDM	Used to identify the type of switch. See the *capability* element in the information models in sections 4.3.2, 4.3.4, 4.3.5.
	TX grid spacing freq / lambda	Used to describe the (technology-specific) features of an optical port. See the *features* element in the OpticalPort object in the information models in sections 4.3.2, 4.3.4, 4.3.5.
	RX grid spacing frequency / lambda	
	Minimum TDM slot	
2. Ethernet Devices Configuration	Match Features	Used to configure the TOR switches built by TU/e (see section 4.3.1. These extensions are defined in the OF-DPA API document and refers to the Open Flow Data Plane Abstraction information model specified in [OFDPA].
3.Packet-Circuit Input and Output	Conversions to and from circuit to packet flows	Used to specify technology-dependent types of action. The modelling of the circuit port characteristics depends on the *features* of the OpticalPort object in the information models in sections 4.3.2, 4.3.4, 4.3.5.
4.Monitoring & Statistics	Circuit flow event monitoring	Used to retrieve monitoring information from the devices, depends on their monitoring capabilities which are specified in the *Flow_Statistics* and *Power_Monitoring* elements of the *capability* objects in the information models in sections 4.3.2, 4.3.4, 4.3.5.

5.Optical Flow Programming	SDM, TDM and WDM parameters Circuit input and output	Used to send configuration commands to SDM, WDM or TDM optical switches. See the modelling of the *CrossConnectionMatrix* element in the information models in sections 4.3.2 (WDM), 4.3.4 (SDM), 4.3.5 (TDM).

4.5 OpenFlow agent implementation

4.5.1 OF agent for ToR

As mentioned in section 4.3.1, the TOR includes both an open flow API (OFDPA) and an open flow agent (Indigo). Figure 24 below shows a block diagram of how the Indigo agent is integrated into the SW stack.

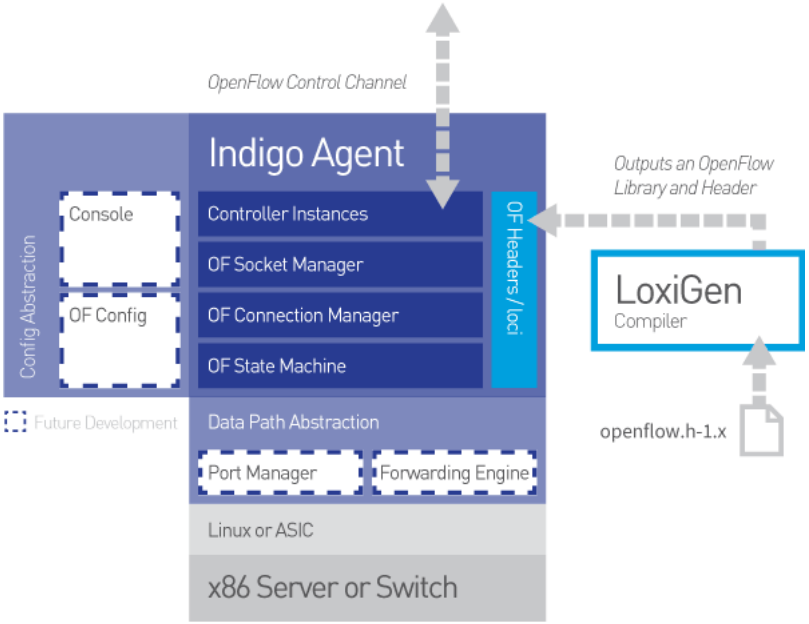

Figure 24 – Indigo agent block diagram

The Open Flow agent is in charge of translating the Open Flow instructions coming from the north bound interface facing the controller to the ofdpa instruction set supported by the ofdpa API. The supported tables and functions of the OF enabled TOR are defined in a Table Type Parameter (TTP) which is an abstract switch model that describes specific switch forwarding behaviours that an OpenFlow controller can program via the OpenFlow-Switch protocol. All the functions supported by the Broadcom based TOR switch are detailed in a json file following the OpenFlow TTP format [OF-TTP], which can be found at the following link:

https://github.com/Broadcom-Switch/of-dpa/blob/b84c4a1a8ea6c5522db08900bc2f2729fd228760/OF-DPA-2.0/ofdpa-v2.0.0.0.ttp.json

4.5.2 OF agent for Optical ToR

As shown in Figure 25, this OF agent is a software component residing between the Optical ToR switch and the SDN controller to enable the communication between them with OF-based protocol. Specifically, the OF agent translates the extended OF messages (as described in section 4.4) into a hardware specific set of control operations, while it also abstracts/maps the device feature/capability information into OF messages. In this way, the OF agent hides the technology-specific communication to the controller, therefore contributing to the homogeneous control of the heterogeneous data plane.

According the control plane requirements, the main responsibilities of the Optical ToR agent are summarized as following:

- Fit/abstract the reported Optical ToR capability (e.g., space and spectrum switching) into proper OF message.
- Resolve the OF configuration requirement from control plane as the configuration for Optical ToR.
- Notify any switch and resource status/availability change occurred in the device up to the control plane, as well as collecting the statistics information.

These requirements pose a set of specific procedures that compose the south-bound communication. Figure 25 illustrates the general connection of the Optical ToR OF agent from an implementation perspective. More specifically, the OF-agent is divided in two parts (or modules) each one implemented in a different programming language. The extended OF API and the technology specific mapping module have been implemented in C under Linux. On the one hand, the Optical ToR is programmed by means of an API provided by device provider in C++ under Windows, so using this API under Windows is a strict requirement. In light of this, we split the agent into two sub-modules (namely, C Agent and C++ Agent) which communicate by means of a UDP socket. Furthermore, the rationale behind using C in Linux is to reuse as much code as possible between the OF agents implementation for different devices. The OF agent communicates with SDN controller and device with an Ethernet connection and USB connection respectively.

Optical ToR switch is implemented with WSS based device which could switch (fixed/flex) spectrum channel from any input port to any output port (without spectrum overlap), so from the resource allocation view, a cross-connection table with combined space/spectrum dimension is maintained by control plane to indicate the existing connection.

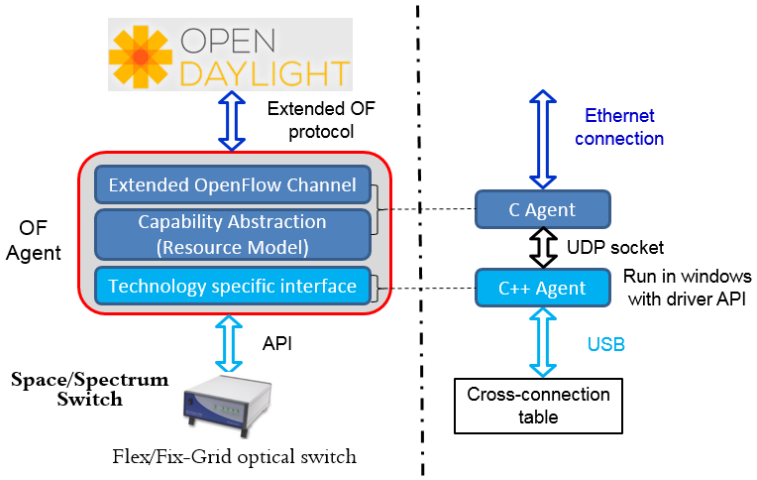

Figure 25 – Optical ToR Agent Design

4.5.3 OF agent for Polatis large Scale Switch

The OF agent implementation detail presented for Optical ToR fits here as well. However, the functional difference between the Optical ToR switch and the Polatis switch is that the Polatis switch is a space/port-based switch while the Optical ToR switch supports switching in space and spectrum dimension. So, without any implementation constraints, this agent is implemented in C with an extended OF library built-in to resolve the OF message from control plane and TL1 protocol library to construct device configuration message through their management interfaces (as shown in Figure 26). The agent communicates with both controller and device through an Ethernet connection. From the resource allocation point of view, a cross-connection table with space dimension is maintained by the control plane to indicate the existing connection.

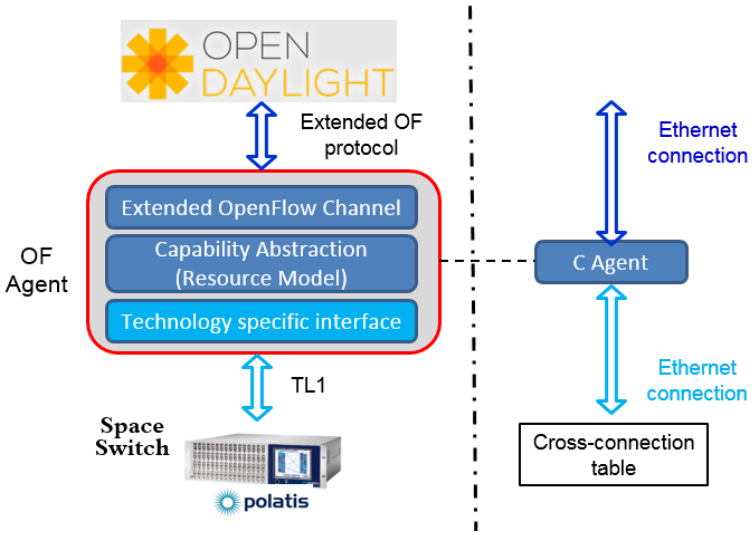

Figure 26 – Polatis Switch Agent Design

4.5.4 OF agent for Optical NIC

As described above, the FPGA-based optical NIC is built in server and communicates with the control plane through an Ethernet connection. So, its OF agent is also implemented in C as shown in Figure 27 and constructs devices configuration message through a specified Ethernet frame including the information defined in section 4.4. However, one of the differences between this NIC and the previous optical switches is that the NIC could parse incoming packets (e.g., read from PCI-E) like an electronic switch, which implies that it could forward traffic according their L2, L3 or even other layer features. Also, this NIC could schedule packets and send them out in a time slot way to initiate a TDM connection. In this case, from a resource allocation perspective, a flow table (instead of cross connection matrix) with flow entries including match field and its associated actions (e.g., forwarded in which time slot) is maintained in control plane to indicate the existing flow forwarding rules and resource availability.

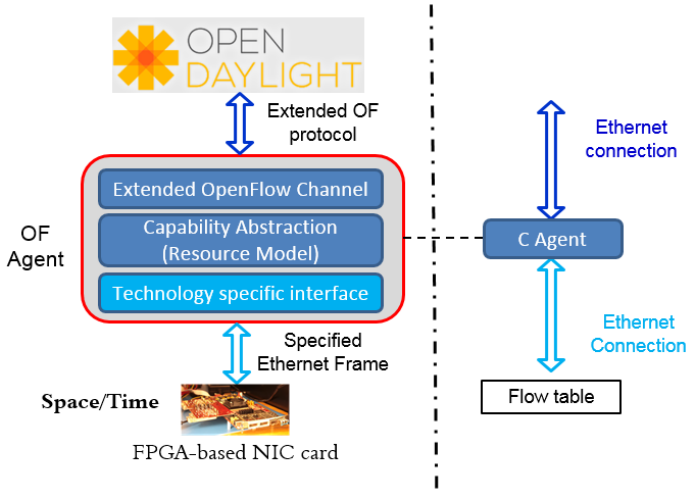

Figure 27 – Optical NIC Agent Design

4.5.5 OF agent for OXS

In the COSIGN scenario, an OXS switch works as a simple fibre switch in *ns* scale. For operation as a scheduled TDM switch, the possible connections depend on time slots. As such the OF agent should be able to push a switching schedule to the switch control system (e.g., FPGA board) along with a time for when this schedule is to be applied. Another problem is that we need to synchronize all the TDM switches to let them work in a perfect union. One option to implement this is that we introduce a global control FPGA which could coordinate with all the switches (e.g., synchronize with same

clock). In this case, control plane need work out the whole path of TDM channel (start/end with TOR or optical NIC) including the DPID of switches and input/output port pair, as well as the time slot to be allocated to this TDM channel. This information should be sent out from controller to the (only) OF agent and OF agent will talk to the global control FPGA only, as shown in Figure 28. Then, the global control FPGA will update the cross-connection table for each switch accordingly. On the other hand, there may be other options to implement this, e.g., developing OF agents for every OXS and keep the cross-connection table locally. In the future development, we need to justify these two options with respect to their implementation difficulty and scalability.

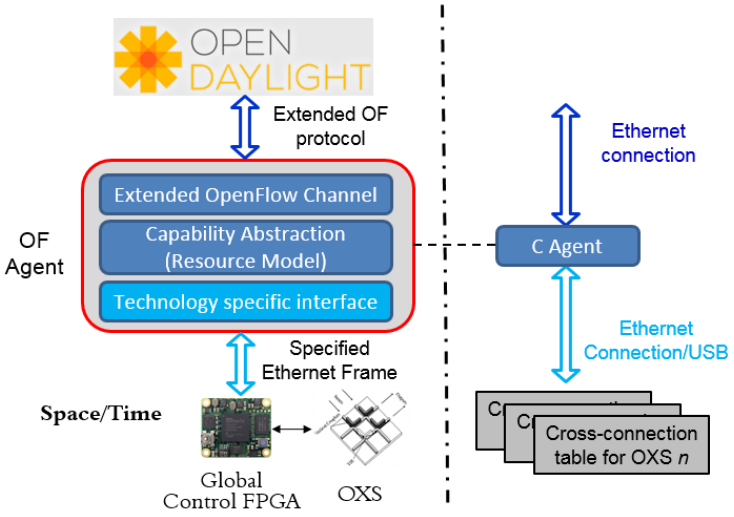

Figure 28 – OXS Agent Design

5 CONCLUSIONS

This deliverable has defined the south-bound and north-bound interfaces of the COSIGN DCN Control Plane. These interfaces allow for the interaction with the COSIGN data-plane on the south-bound side and the cooperation between network controller and cloud orchestrator or DC management platform on the north-side. Both interfaces have been defined considering the most relevant standards and common practice in the related areas, in order to simplify the possible adoption and interoperability of the COSIGN solution in existing SDN environments and data centre platforms.

The specification of the interfaces will feed the next WP3 activities for the development of the SDN framework for control and management of COSIGN data centre networks. A preliminary mapping with the services, components and APIs, already available in OpenDaylight (the reference SDN platform selected for COSIGN developments) has been provided in this document, as a starting point to drive the implementation of the COSIGN SDN framework.

In particular, the south-bound interface will be implemented as an extension of the OpenDaylight OpenFlow plugin on the controller side, while OpenFlow agents for the COSIGN devices will be developed to support the subset of OpenFlow messages identified in the document. On the north-bound side, RESTConf has been chosen as a reference protocol for the A-CPI of all the new services designed in COSIGN as internal components of the SDN controller. The OpenDaylight Model Driven Service Abstraction Layer (MD-SAL) framework can be used for the development of these services, in order to simplify the development of the service interfaces that can be implemented through the definition of the associated YANG models.

In the next period, WP3 will select a relevant subset of the services and the interfaces defined in this document, in order to proceed with the implementation of a preliminary version of the COSIGN SDN Control Plane prototype, planned for the end of the second year.

6 REFERENCES

[COSIGN-D31] COSIGN consortium, Deliverable D3.1, “SDN framework functional architecture”, December 2014

[JSON-YANG] L. Lhotka, “Modeling JSON Text with YANG”, IETF draft, work in progress, September 2013

[ODL-AAA] OpenDaylight AAA project, wiki page: https://wiki.opendaylight.org/view/AAA:Main

[ODL-DOVE] OpenDaylight OpenDOVE project, wiki page: https://wiki.opendaylight.org/view/Open_DOVE:Main

[ODL-NIC] OpenDaylight Network Intent Composition, wiki page: https://wiki.opendaylight.org/view/Network_Intent_Composition:Main

[ODL-OFL] OpenDaylight OpenFlow protocol plugin project, wiki page: https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:Main

[ODL-OFP] OpenDaylight OpenFlow protocol library project, wiki page: https://wiki.opendaylight.org/view/Openflow_Protocol_Library:Main

[ODL-POL] OpenDaylight Group Policy project, wiki page: https://wiki.opendaylight.org/view/Group_Policy:Main

[ODL-SFC] OpenDaylight Service Function Chaining, wiki page: https://wiki.opendaylight.org/view/Service_Function_Chaining:Main

[ODL-VTN] OpenDaylight Virtual Tenant Network project, wiki page: https://wiki.opendaylight.org/view/OpenDaylight_Virtual_Tenant_Network_(VTN):Main

[OF] Open Networking Foundation, “OpenFlow Switch Specification”, version 1.3.4, March 2014

[ODL-wiki] https://wiki.opendaylight.org/view/OVSDB_Integration:Design

[OFDPA] Broadcom, “OpenFlow Data Plane Abstraction: Abstract Switch Specification”, version 2.0, November 2014.

[OF-TTP] Open Networking Foundation, “OpenFlow Table Type Patterns”, version 1.0, August 2014

[OFP-CS] “Extensions to the OpenFlow Protocol in support of Circuit Switching- Addendum to OpenFlow Protocol Specification (v1.0) – Circuit Switch Addendum v0.3”, June 2010

[ONF-arch] Open Networking Foundation, “SDN architecture. Issue 1”, June 2014

[ONF-arch-0] Open Networking Foundation, “SDN architecture overview. Version 1.0”, December 2013

[ONF-NBI] Open Networking Foundation, “North-bound Interfaces Working Group (NBI-WG) Charter”, June 2013

[OpenStack] OpenStack web page: http://www.openstack.org/

[Opflex1] Smith, M. et alt. ”Opflex Control Protocol”, IETFs draft, work in progress, November 2014. https://tools.ietf.org/html/draft-smith-opflex-01

[Opflex2] ODLs Opflex Architecture Wiki Pages, https://wiki.opendaylight.org/view/OpFlex:Opflex_Architecture

[REST] R. Fielding. “Architectural Styles and The Design of Network-based Software Architectures”. PhD thesis, University of California, Irvine, 2000

[RESTCONF] A. Bierman, M. Bjorklund, K. Watsen, R. Fernando, “RESTCONF protocol”, IETF draft, work in progress, February 2014

[RFC6020] M. Bjorklund, “YANG – A data modelling language for the Network Configuration protocol (NETCONF)”, IETF RFC 6020, October 2010

[RFC6991] J. Schoenwaelder, “Common YANG Data Types”, IETF RFC 6991, July 2013

[RFC7223] M. Bjorklund, “A YANG Data Model for Interface Management”, IETF RFC 7223, May 2014

[RFC7277] M. Bjorklund, “A YANG Data Model for IP Management”, IETF RFC 7223, June 2014

[SDN-ONF] "Software-Defined Networking (SDN): The New Norm for Networks". Open Networking Foundation

[SDNi] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, R. Sidi, “SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains”, IETF draft, work in progress, June 2012.