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 D3.1 - Update

SDN framework functional architecture

Due date of deliverable: December 31, 2014
Actual submission date: January 16, 2015
Update submitted: May 8, 2015

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

Lead contractor for this deliverable:
NXW, Giada Landi

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 describes the functional architecture of the Control Plane that operates the COSIGN Data Centre Network (DCN) following the principles of the Software Defined Networking (SDN). Starting from a survey of the state of the art for management and control of Data Centre networks, virtualization of network resources, infrastructures and services, and an analysis of the existing SDN controllers' software, this document identifies the main gaps and limitations of the solutions currently available in the market and in the open source community. Taking into account the control plane requirements and the initial architecture considerations from Work Package 1, this deliverable defines the main services which will be provided by the COSIGN control plane, its functional components and the workflows and interactions with the other layers of the overall COSIGN architecture. An initial and high-level description of the information models and interfaces between control plane and cloud orchestration platform (developed in Work Package 4) is also provided as starting point for the interface specification that will be addressed in detail in the next deliverable *D3.2 "SDN framework north-bound and south-bound interfaces specification*".

Document Information

Status and Version:	V1.1	
Date of Issue:	December 28th, 2014	
Dissemination level:		
Author(s):	**Name**	**Partner**
	Giada Landi	NXW
	Giacomo Bernini	NXW
	Gino Carrozzo	NXW
	Nicola Ciulli	NXW
	José Soler	DTU
	Cosmin Caba	DTU
	José Ignacio Aznar	I2CAT
	Amaia Legarrea	I2CAT
	Eduard Escalona	I2CAT
	Bingli Guo	UnivBris
	Shuping Peng	UnivBris
	Reza Nejabati	UnivBris
	Dimitra Simeonidou	UnivBris
	Fernando Agraz	UPC
	Salvatore Spadaro	UPC
	Jordi Perelló	UPC
	Albert Pagès	UPC
	Yaniv Ben-Itzhak	IBM
	Alessandro Predieri	IRT
	Matteo Biancani	IRT
Edited by:	Giada Landi	NXW
Checked by :	Sarah Renee Ruepp	DTU

Table of Contents

1 Introduction

This deliverable describes the functional architecture of the Control Plane that operates the COSIGN Data Centre Network (DCN) following the principles of the Software Defined Networking (SDN). Starting from a survey of the state of the art for management and control of Data Centre networks, virtualization of network resources, infrastructures and services, and an analysis of the existing SDN controllers' software, this document identifies the main gaps and limitations of the solutions currently available in the market and in the open source community. Taking into account the control plane requirements and the initial architecture considerations from Work Package 1, this deliverable defines the main services which will be provided by the COSIGN control plane, its functional components and the workflows and interactions with the other layers of the overall COSIGN architecture. An initial and high-level description of the information models and interfaces between control plane and cloud orchestration platform (developed in Work Package 4) is also provided as starting point for the interface specification that will be addressed in detail in the next deliverable D3.2 "*SDN framework north-bound and south-bound interfaces specification*".

Following the notion of recursive abstraction of resources and services introduced in the SDN architecture by the Open Networking Foundation (ONF), the COSIGN control plane operates the heterogeneous optical devices available at the COSIGN DCN data plane through advanced optical virtualization techniques. This approach allows for obtaining a unified view of the multi-technology DCN. Thus, dedicated core modules and applications within the SDN controller can exploit this common model to implement basic and advanced network services which provide the network connectivity inside the data centre. This connectivity will be compliant with the dynamicity and QoS constraints required by the cloud services. On the other hand, the integration with the DC management and cloud orchestration platforms will introduce the level of automation and the optimization in the usage of the whole DC resources that are becoming the most critical challenges for DC operators. In fact, these core functions within the SDN controller provide a further level of abstraction, exposing technology independent APIs which can be used by external applications at the cloud platform level to manipulate the DC connectivity and create enhanced multi-tenant infrastructures for on-demand cloud services.

1.1 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 D3.2 – SDN framework north-bound and south-bound interfaces specification

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
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
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
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
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	
01			
02			
03			

2 State of the Art Survey

2.1 Management and Control of Data Centre Networks

Modern Data Centres (DCs) are becoming increasingly complex and massive. Proliferation of new technologies such as virtualization is adding yet another level of complexity while enabling higher workloads to be placed on the network. In order to cope with different demands of Data Centre providers and of Data Centre tenants, a control and management architectures are required.

The management layer creates rules and policies for:

- Providing different networking models for the needs of different applications or tenants. For instance, flat networks or VLANs for separation of servers and traffic.
- Managing IP addresses, allowing for dedicated static IPs or DHCP. Floating IPs allow traffic to be dynamically rerouted to any computing resource, which allows to redirect traffic during maintenance or in the case of failure.
- Integrating with commodity gear or advanced networking services from supported vendors.
- Deploying different network policies.

Then, the control layer is responsible to configure the programmable network devices by translating the rules and policies defined by the network manager according to the network topology and devices. Following the Software Defined Networking (SDN) architecture, the control layer can be implemented through an SDN controller, which should support all configuration protocols of the programmable network devices residing in the network, e.g., OpenFlow [OF].

There are several industry closed network control architectures, such as Cisco UCS (Unified Computing System) [UCS], and Juniper QFabric [QFabric].

Cisco UCS provides an (x86) architecture data centre server platform composed of computing hardware, virtualization support, switching fabric, and management software. It includes a Cisco UCS manager software embedded into the 6100/6200 series Fabric Interconnect, which is accessed by the administrator through a common browser, or a Command Line Interface (CLI), or programmatically through an API. Virtual Machines (VMs) can be moved from one physical chassis to another, applications may be moved between Virtual Machines, and management may even be conducted remotely from an iPhone using SiMU (Simple iPhone Management of UCS). In addition to the embedded software, administrators may also manage the system from VMware's vSphere.

The Juniper QFabric System is composed of multiple components working together as a single switch to provide high-performance, any-to-any connectivity and management simplicity in the Data Centre. The QFabric System flattens the entire Data Centre Network (DCN) to a single tier where all access points are equal, eliminating the effects of network locality and making it the ideal network foundation for cloud-ready, virtualized Data Centres. QFabric is a highly scalable system that improves application performance with low latency and converged services in a non-blocking, lossless architecture that supports Layer 2, Layer 3, and Fibre Channel over Ethernet capabilities. The QFabric Director component provides control and management services for the full QFabric System.

On the open source side, there are several well-known cloud infrastructure management systems, such as OpenStack [OpenStack], CloudStack [CloudStack], and OpenNebula [OpenNebula].

OpenStack is an IaaS cloud computing project that provides a free open source middleware released under the terms of the Apache License. It consists of a series of interrelated projects that controls large pools of processing, storage, and networking resources throughout a Data Centre, all managed through a dashboard (Figure 1). The goal of OpenStack is to allow any organization to create and offer cloud computing capabilities using open source software running on standard hardware. Among the others, it includes OpenStack Compute, Storage and Network components. The OpenStack Compute component is a piece of software for automatically creating and managing large groups of virtual private servers. The OpenStack Storage component is used for creating redundant, scalable object

storage using clusters of commodity. Finally, the OpenStack Network component, called Neutron, is a pluggable, scalable and API-driven system for managing networks and IP addresses. Like other aspects of the cloud operating system, it can be used by administrators and users to increase the value of existing Data Centre assets. Neutron ensures the network will not be the bottleneck or limiting factor in a cloud deployment and gives users a real self-service to create their own customized network environments, even over their network configurations.

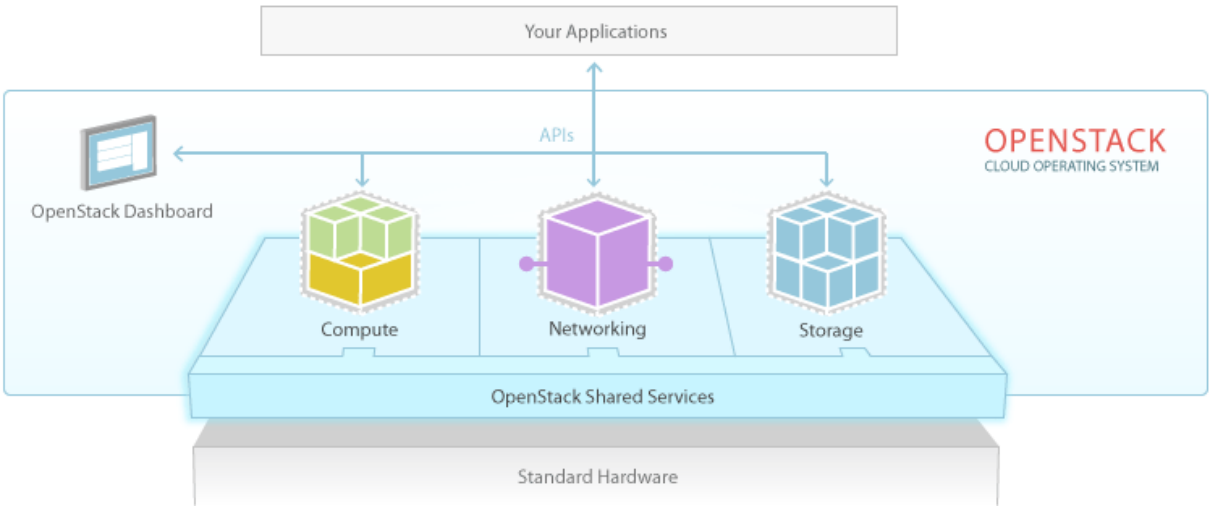


Figure 1 – OpenStack structure [OpenStack]

Apache CloudStack is an open- source software designed to deploy and manage large networks of Virtual Machines, as a highly available, highly scalable Infrastructure as a Service (IaaS) cloud computing platform. CloudStack is used by a number of service providers to offer public cloud services and by many companies to provide an on-premises (private) cloud offering, or as part of a hybrid cloud solution. CloudStack is a turnkey solution that includes the entire "stack" of features most organizations want with an IaaS cloud: compute orchestration, Network-as-a-Service (NaaS), user and account management, a full and open native API, resource accounting, and a first-class User Interface (UI). CloudStack currently supports the most popular hypervisors: VMware, KVM, XenServer, Xen Cloud Platform (XCP) and Hyper-V. Users can manage their cloud with an easy to use Web interface, command line tools, and/or a full-featured RESTful API. In addition, CloudStack provides an API that's compatible with AWS EC2 [AWS-EC2] and S3 [AWS-S3] for organizations that wish to deploy hybrid clouds.

The OpenNebula project aims to lead innovation in enterprise class cloud Data Centre management. It develops an advanced, highly scalable and adaptable solution for building and managing virtualized Data Centres and enterprise clouds. The project provides a modular system that can implement a variety of cloud architectures and can interface with any Data Centre service. OpenNebula provides a flexible middleware that enables Data Centres to build IaaS cloud computing environments. It supports different virtualization technologies including XEN, KVM, VMWare and VirtualBox and it allows a wide range of customization on the private cloud.

The proposed features of such Data Centre management solution are numerous and diverse.

Herein we list some of the features which are relevant for SDN:

- Dynamic workload management: automates the distribution of compute, network, and storage resource across the physical infrastructure while adhering to defined policies on load balancing, data security, and compliance.
- Broad network virtualization capabilities: support for embedded software-based network management as well as VLANs.
- MPLS support in the cloud: dedicate an entire VLAN to a specific account for MPLS support between network nodes.

- Secure cloud deployments: ensures that all memory, CPU, network, and storage resources are both available and isolated from one user account to another within the cloud.
- Virtual routers, firewalls and load balancers: supports integration with both software and hardware firewalls and load balancers to provide additional security and scalability to a user's cloud environment.

Other relevant features for a cloud management platform include a rich management user interface, the hypervisor independency, an easy creation process based on virtual infrastructure templates, management functions for VMs snapshots, synchronization, live migration and high availability, usage metering, event/audit logs with alerts and notifications, mechanisms for host maintenance and open APIs for cloud consumers.

It should be noted that these cloud platforms provides high-level management functions for the overall DC infrastructure and cloud services, orchestrating the whole set of physical and/or virtual resources composing the IaaS offer. However, they rely on specific tools for controlling and managing the network part of the Data Centre. For example, Neutron in OpenStack provides REST APIs to create and manipulate multi-tenant virtual networks or network appliances like firewalls and load balancers, but the actual configuration of the network is enabled through dedicated plugins which interacts directly with the data plane or, in most of the cases, with control entities that operates over the network devices using the specific protocols supported by the different DCN technologies. This approach is adopted in particular in SDN architectures applied to Data Centres. The cloud management platform is in charge of orchestrating the whole DC infrastructure and the mix of computing, storage and networking resources. On the other hand, the network component of the cloud platform "delegates" the actual network operation to an SDN controller which is responsible to apply the cloud platform's directives and configure the DCN data plane devices. This approach allows to partially decouple the control of the DCN from the management of the whole cloud service, hiding the technological and topological details of the DCN infrastructure from the cloud platform. The result is a DCN control plane which is able to automate most of the network configuration process, taking also autonomous decisions to improve network performance and utilization, but still driven from the characteristics and dynamicity of the cloud services. The COSIGN architecture adopts this decoupling between cloud management - service orchestration platform and DCN control plane: Work Package 4 (WP4) will define and implement the first component, i.e. the cloud platform with service orchestration functions, while Work Package 3 (WP3) is dedicated to the DCN control plane and its SDN controller.

2.2 Network Virtualization in Data Centre Networks

Deliverable D1.3 [2], produced in WP1, which reports on existing optical and control plane technologies has analysed a variety of solutions for network virtualization considering both industrial products and open-source software. In both cases, network virtualization strongly relies on the SDN paradigm. A conceptually centralized SDN controller configures the underlying physical network devices and/or the servers at the network edges to create multiple instances of virtualized and isolated networks which share a common physical infrastructure.

The network virtualization solutions analysed in WP1 mostly operates on L2-L3 networks, without considering the virtualization of optical technologies. Two main categories of network virtualization have been identified: **overlay-based network virtualization** and **direct fabric programming**. The former solution delivers customer-defined infrastructures, often including L4-L7 services and based on the required application-layer connectivity. The multi-tenant overlay networks are implemented encapsulating the traffic at the end-point level, using VXLAN, NVGRE or STT technologies. The traffic generated at the network edges is delivered through virtual tunnels created on top of the physical infrastructure, without requiring any configuration of the internal network devices. Overlay-based virtualization is simple and fast to deploy, since it requires only configuration at the network edges, without any impact on the configuration of the physical connectivity. However, this is also its major limitation, since the decoupling between overlay and physical networks means lack of visibility and, consequently, lack of control on the data plane resources. In other terms, the physical connectivity is used by the virtual tunnels as it is, without any coordinated adaptation to optimize the network performance and usage, to meet specific requirements at the virtual infrastructure level or just

to react to data plane failures. On the other hand, underlay-based network virtualization solutions with direct configuration of the network fabric dynamically create network paths programming the virtual or physical devices, for example using the OpenFlow protocol [OF], and maintain more control on the data plane configuration.

The current research is focusing on possible mechanisms to efficiently coordinate these two virtualization approaches. This coordination has twofold benefits: on one hand it allows to reconfigure the underlying data plane depending on the overlay virtual networks that are currently established over the infrastructure and, on the other hand, it allows to plan new overlay networks (or modify the existing ones) depending on the capacity of the connectivity provisioned through the configuration of virtual paths in the data plane. This mixed approach will be adopted also in COSIGN, with the objective of bringing a powerful level of mutual awareness and cooperation between the underlay-based and overlay-based virtualization.

In the COSIGN architecture (*Figure 2*), as initially defined in Deliverable 1.1 [1], network virtualization based on direct programming of network devices is implemented through infrastructure control functions provided at the SDN controller level. This kind of virtualization, applied at the infrastructure level, will compose and deliver virtual network slices with optical capabilities and able to expose a certain level of network programmability. The SDN controller, in fact, will also provide dedicated functions to operate, manage and control each virtual slice, for example for monitoring or provisioning of virtual paths and connectivity. This infrastructure-level virtualization will be entirely addressed in WP3 and matches the Virtual Data Centre use case defined in WP1.

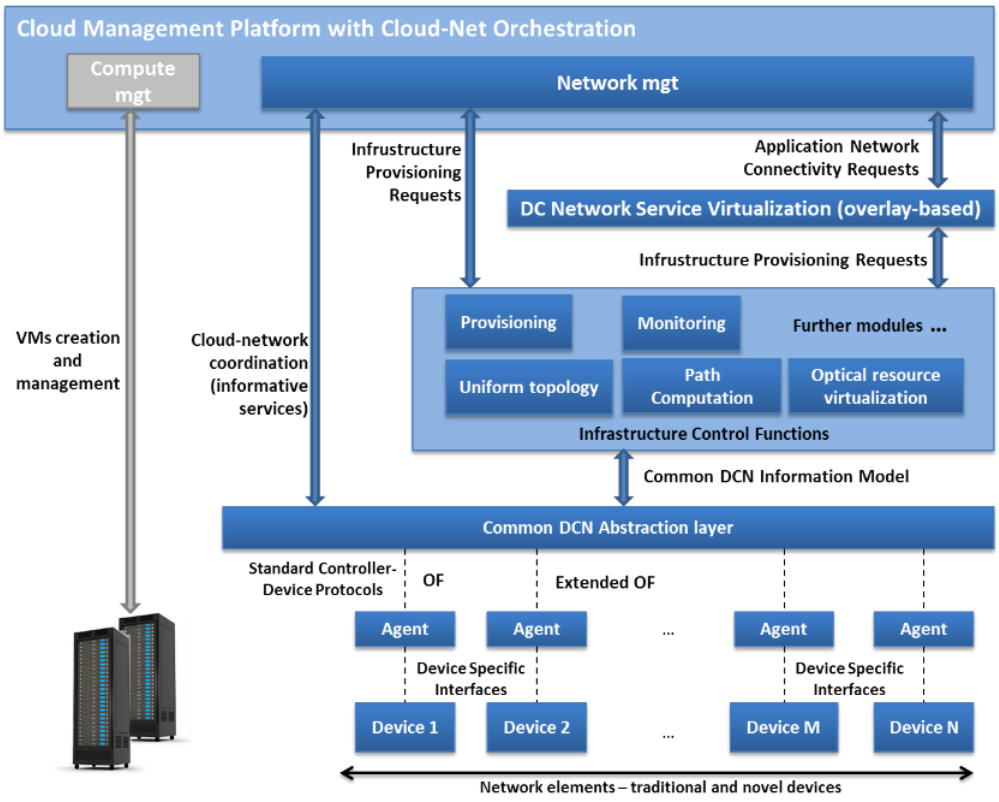


Figure 2 – COSIGN architectural blueprint [1]

The overlay-based virtualization functions, with the objective of delivery customized overlay networks matching a high-level description of application-layer connectivity requirements (e.g. in terms of traffic profiles) will be addressed in WP4. In COSIGN, the overlay-based virtualization can be dynamically composed with infrastructure-level virtualization, obtaining overlay networks established on top of highly-programmable virtual slices. This approach allows the bi-directional cooperation between the COSIGN architectural layers and functional entities managing overlay-based (WP4) and infrastructure-level (WP3) network virtualization, with all the benefits discussed above for the mixed virtualization model.

An additional innovation of the COSIGN project is the adoption of an optical data plane in the DC network, based on a mixed set of heterogeneous technologies. In order to efficiently manage the whole infrastructure through unified procedures, COSIGN architecture includes a further layer of virtualization mechanisms applied at resource level. The virtualization of optical resources allows for generating an abstracted and uniform view of the different optical technologies available at the data plane, while maintaining a powerful description of resource capabilities. This unified model simplifies the manipulation of the heterogeneous resources and constitutes a fundamental enabler for the infrastructure-level virtualization responsible to create virtual optical slices. Optical resource virtualization is discussed in further details in the following subsection.

2.2.1 Virtualization of Optical Resources

Optical Network Virtualization is defined as the composition of multiple isolated virtual optical network slices simultaneously coexisting over shared physical optical network infrastructure. In each virtual slice, virtual optical nodes are inter-connected by a set of virtual optical links. Virtualization of optical network resources i.e., optical nodes and optical links is often achieved by partitioning or aggregation of the corresponding physical resources. Generally, the virtualization of optical nodes should be considered jointly with the virtualization of optical links. The granularity of virtual links in a virtual network slice is inherited from the switching capabilities supported by the optical switching nodes as well as capabilities of optical transponders.

The two most important principles of optical network virtualization are isolation and coexistence of multiple virtual optical network slices. Isolation implies that different slices sharing the same optical network should not interfere with each other. Coexistence means that the network slices can be supported and provisioned in parallel to different administrative entities.

Compared to the network virtualization in Layer 2 and 3, optical network virtualization is still under initial study. This is mainly because of the complexity of virtualization approach suitable for optical networks and the analogue nature of optical networks. The underlying optical network technologies will impact the number and characteristics of virtual network slices that can be built on top of a physical optical network infrastructure.

In COSIGN, we are adopting heterogeneous advanced optical network technologies including both switching (Large-port space switch, Fast switch, Wavelength selective switch, High-performance electronic switch) and transport (Wavelength Division Multiplexing – WDM – and Space Division Multiplexing – SDM) technologies. Therefore, it is challenging to design and develop proper virtualisation mechanisms for such optical data plane. The features of each technology need to be investigated carefully to fully explore its advantage while virtualising it. Meanwhile, the unique optical layer constraints (e.g. wavelength/spectrum continuity and impairments) need to be taken into account when composing virtual optical slices over these technologies.

The optical network virtualisation should also be coordinated with the IT virtualisation (compute and storage). The seamless integration of the optical network virtualisation and the IT virtualisation will become a key highlight of the COSIGN project.

2.3 SDN and NFV

The new all-optical Data Centre Network proposed in COSIGN poses several challenges to the control and management, as well as to the infrastructure virtualization and orchestration, which are tackled in WP3 and WP4 of the project. In particular, WP3 puts its focus on the control layer. In the framework of COSIGN, the role of the control plane is two-fold. On the one hand, it is the responsible entity for configuring the optical data plane and providing the network intelligence. Thus, the control plane has to implement a set of procedures that allow the setup, tear down and modification of connectivity services that will be used by the upper layer applications. On the other hand, the control plane has to implement the tools necessary to enable the virtualization of the optical network infrastructure. The goal here will be to provide an abstracted and unified view of the underlying heterogeneous optical data plane to the application and orchestration layer. This logical representation of the network will hide technology specific details of the data plane to simplify the configuration of the network infrastructure from upper layers (e.g., create virtual optical slices).

In this context, the goal of COSIGN is to leverage arising software-based paradigms, such as SDN and NFV, by providing them with capabilities to control, manage, virtualize and orchestrate the proposed optical data plane.

The SDN architecture aims at improving the programmability, automation and control of the network [SDN-ONF]. To achieve this, SDN decouples the control and data planes, and keeps the network intelligence logically centralized. The main entity in this model is the software-based SDN controller, which maintains the global view of the network infrastructure and centralizes the network intelligence. The controller has two main communication interfaces. The southbound interface (Data plane to Controller Plane Interface – D-CPI) communicates the controller with the underlying network infrastructure, thus enabling the management of the physical devices. The northbound interface (Application plane to Controller Plane Interface – A-CPI) allows the communication between the control and application layers. Hence, the controller enables the network infrastructure to be abstracted for upper layers applications and network services so it can be treated as a logical or virtual entity.

From a functional point of view, with SDN, operators can modify the network behaviour and deploy new applications and services in few hours or days. In this way, they can develop their own applications to configure, manage, secure and optimize the network instead of waiting for desired features to be deployed in vendor's equipment. Moreover, the SDN architecture supports the implementation of customized common network services, such as routing, bandwidth management, traffic engineering, QoS, etc., which are then available to the application and orchestration layer through the northbound interface. Therefore, SDN-based control simplifies the network design and operation, and allows enterprises and carriers to gain vendor-independent control over the entire network. In addition, SDN makes the management of the network devices easier since the southbound interface can be implemented by means of just one protocol (e.g. OpenFlow). Thanks to this simplified operation, network administrators can programmatically configure the abstracted network rather than having to configure each single device manually. It is worth noting here the importance of the OpenFlow protocol as a key enabler of the SDN paradigm. OpenFlow was the first standard defined to implement the southbound interface. In brief, OpenFlow allows moving network control out of the network devices (either physical or virtual) to the control plane since it enables direct access to and manipulation of the forwarding plane of such devices.

In light of the above, the OpenFlow-based SDN architecture overcomes several limitations associated to current networking technologies. More specifically, it provides a centralized control of heterogeneous environments, so that Data Centre operators can easily develop SDN-based orchestration and management tools to deploy, configure and update the underlying DCN infrastructures in an automated and integrated way. This reduced complexity in automation minimizes the operational costs and errors, and facilitates a quick deployment of new custom services and applications thus increasing business agility. In this regard, SDN accelerates business innovation since DC operators can dynamically program the DCN to meet business needs and user requirements as they arise. Moreover, through the flow-based control model, operators are able to increase the granularity of the network control. In particular, they can apply policies at different levels (such as session, user, device, etc.) in a highly abstracted and automated way. This control allows cloud operators, for instance, to support multi-tenancy while keeping traffic isolation, security and flexible resource management in the network. Deliverable D1.3 [2] provides further details on the SDN paradigm.

Network Functions Virtualization (NFV) leverages IT virtualization technology to transform the way that operators design their networks [NFV]. In brief, NFV moves the functionalities carried out by specialized network hardware to virtual machines placed in a Data Centre. Hence, by consolidating diverse network hardware appliances into industry standard high volume servers, switches and storage, network operators can reduce the variety and number of equipment, capital expenses, power consumption and time to market associated to new service deployments. More precisely, using COTS hardware (i.e., servers and storage) to deploy the Virtual Network Functions (VNFs) instead of dedicated devices improves capital efficiencies and reduces the variety of equipment, which also results in more efficient space usage. Furthermore, the flexibility in assigning VNFs to hardware allows to deploy the software in the most appropriated places for each case (e.g., in customer's premises, Data Centres, central offices, etc.). This aids scalability, performance, resource sharing and resilience. Besides this, decoupling functionality from location also allows to reduce power

consumption since workloads can be redistributed, and unused hardware can be powered off. Finally, the software-based service provisioning offered by NFV minimizes the network operator cycle of innovation, thus reducing the time to market, and reduces the investment recovery effort in front of hardware-based functionalities.

From a functional perspective, NFV is applicable to any kind of data plane packet processing or control plane procedure. Some examples of network processes that can be virtualized are traffic analysis, Service Level Agreement (SLA) monitoring, application-level optimization (e.g., cache servers, load balancing), policy control, charging platforms, etc. In the same line, operators can virtualize network devices and their functionalities as well. For example, operators can virtualize switching elements such as NAT servers or routers, tunnelling gateways (IPsec/SSL VPN), firewalls, Authentication, Authorization and Accounting (AAA) servers, etc.

NFV objectives of reducing CAPEX, OPEX, space and power consumption can be achieved using techniques currently used in Data Centres. Nonetheless, the SDN approach, which relies on the separation between the control and the data planes, can improve performance and facilitate management, operation and maintenance procedures.

In light of the above, SDN arises as a valid candidate to deploy the control plane of COSIGN since it provides an appropriated architectural design to implement the required functionalities (configuration, management, monitoring, etc.). It is clear though, that the model will need to be revisited to adapt it to the specific requirements of the proposed data plane. In this line, the OpenFlow protocol can be extended to obtain an OpenFlow-controlled optical data plane. Furthermore, some effort will be necessary to implement the network infrastructure abstraction. Specifically, the information model to represent the different data plane devices homogeneously will need to be defined and integrated into the SDN architecture.

The NFV model could be applied to the control plane of COSIGN as well. As a matter of fact, all the network intelligence associated to completely optical networks (i.e., route computation, bandwidth assignment, traffic engineering, etc.) has been traditionally moved to compute nodes since optical devices lack these kinds of functionalities. However, the NFV paradigm is still in an early state of definition and design so further analysis on the subject will be needed.

2.4 SDN Controllers

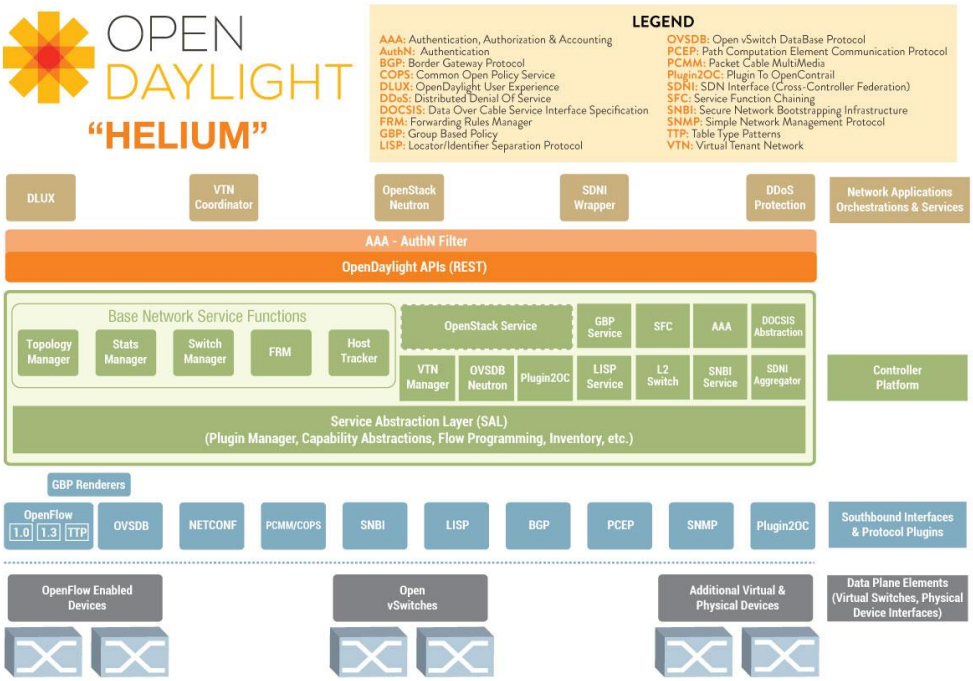
Deliverable D1.3 [2] provides a survey of the main open source SDN controllers from a functional point of view. This section extends this analysis focusing mostly on the software aspects of each SDN controller platform, with the objective of selecting a suitable candidate for the development of the COSIGN SDN controller.

The SDN controller software analysis follows a common template, designed to cover the most important aspects to be considered for the COSIGN controller. In particular:

- Flexibility of the software architecture, focused on the possibility to extend the controller with southbound plugins for heterogeneous optical devices and internal or external applications for optical resource virtualization and DCN control and optimization.
- Stability and maintenance of the software, availability and quality of development documentation, in order to rely on a solid platform for COSIGN developments.
- Usability of the software and strength of the community, to guarantee a strong impact of COSIGN achievements.
- Level of expertise and involvement of COSIGN partners in the SDN controller development, for quick and efficient developments of COSIGN extensions in the proof-of-concept prototype and to facilitate COSIGN contributions to the open source initiatives, mediated through COSIGN partners with relevant roles in the related communities.

2.4.1 Open Daylight

OpenDaylight	
Web-site	http://www.opendaylight.org
Brief description	OpenDaylight is an open source platform for network programmability that enables SDN through a combination of components including a fully pluggable controller, interfaces, multi-protocol plug-ins and applications. The core part of the project, the modular, pluggable and flexible controller, is implemented in Java. The northbound and southbound interfaces have clearly defined and documented APIs. With this common platform, both customers and vendors can innovate and collaborate in order to commercialize SDN- and NFV-based solutions. OpenDaylight is widely supported by both the industry and the open source community. It has several means to help developers (a wiki, mailing lists, IRC channels, events organization, etc.). However, since it is a huge and high modularity project with massive function modules, developing some kinds of extensions may require big effort.
Software	
Latest sw version and release date	The latest stable version of OpenDaylight is called Helium and was released in September 2014. The OpenDaylight community is very active, so several commits are done weekly in the working code repositories of the different projects that compose the OpenDaylight initiative.
Software architecture	The OpenDaylight SDN platform architecture is composed of three main layers (*Figure 3*): • The Network Application Orchestration and Services: The top layer consists of business and network logic applications for network control, provisioning and monitoring. In addition, more complex and advanced orchestration applications needed for cloud, data centre and (network function) virtualization services also reside in this layer. • The Controller Platform: The SDN abstraction happens in this intermediate layer. It is composed of a collection of dynamically pluggable modules to perform a variety of network tasks and services. The SDN controller platform offers a set of network services such as the Topology Manager, the Statistics Manager, the Switch Manager, the Forwarding Rules Manager and the Host Tracker. These modules provide a set of common APIs to the applications and orchestration layer to let it program the network and the controller behaviour. • Southbound Interfaces and Protocol Plugins: OpenDaylight supports multiple southbound interfaces through a heterogeneous set of dynamically loadable plugins. In this way, OpenDaylight implements a wide set of southbound protocols, such as OpenFlow 1.0, OpenFlow 1.3, Open vSwitch Database (OVSDB), etc., to control the physical hardware within the network which is composed of a wide range of physical and virtual devices, (e.g., switches and routers) that provide the actual connective fabric between the network endpoints. These modules are dynamically linked into a Service Abstraction Layer (SAL). The SAL exposes device services to the network function modules in controller and determines how to fulfil the requested service irrespective of the underlying protocol used between the controller

	and the network devices.  Figure 3 – OpenDaylight architecture (Helium version)
Source code distribution	The OpenDaylight project maintains a git repository located at: http://git.opendaylight.org OpenDaylight has currently more than 30 projects each maintaining its own repository. The list of projects can be found at: https://git.opendaylight.org/gerrit/#/admin/projects/
Software governance	OpenDaylight is a Linux Foundation Collaborative Project and implements many open source best practices familiar to other leading projects. Each of the OpenDaylight projects is made available under the Eclipse Public License (EPL-1.0). Any individual, company or organization can engage directly and immediately to begin contributing to the project. Although source code can be pulled anonymously from the repo, contributors need to have an OpenDaylight Account to push their developments.
Sw package distribution	The different stable releases of OpenDaylight (Helium and its predecessor Hydrogen), can be downloaded from downloads section in the project website (http://www.opendaylight.org).
Core language	Java
Applications language or interface	OpenDaylight supports the OSGi framework and bidirectional REST for the northbound API. The OSGi framework is used for applications that will run in the same address space as the controller while the REST (web based) API is used for applications that do not run in the same address space (or even necessarily on the same machine) as the controller. In MD-SAL, it also supports DOM APIs which is mostly useful for XML-driven application types.
Usability	
Installation	Installation instructions are available and up-to-date in the wiki of the project

documentation	(https://wiki.opendaylight.org). Moreover, an installation guide can be downloaded with the software (http://www.opendaylight.org/software/downloads).
Operative documentation	A user guide can be downloaded along with the OpenDaylight release. Additionally, a starting guide is also available in the website (http://www.opendaylight.org/resources/getting-started-guide).
Development	
Development documentation	A wiki is available for developers at (https://wiki.opendaylight.org). It is a live document so it is updated periodically as long as the OpenDaylight project evolves. There is also a set of mailing lists and an IRC channel.
Documentation for application development	Refer to previous point.
Development roadmap	Refer to each project section in the OpenDaylight developers' wiki (https://wiki.opendaylight.org/view/GettingStarted:Developer_Main). Also, several sample applications are developed. (https://wiki.opendaylight.org/view/OpenDaylight_Controller:Sample_Applications)
Toolchain	Java 1.7 + Maven + OSGi
IDE integration	As reported in the wiki of the project (https://wiki.opendaylight.org/view/GettingStarted:_Eclipse), at present only the Controller and the OpenFlow plugin projects have been validated to import into Eclipse without compile errors.
Deployment	
System requirements	OpenDaylight Controller runs in a JVM. Being a Java application, it can potentially run on any operating system that supports Java. However, for best results a recent Linux distribution and Java 1.7 are recommended. The only information about system requirements for running the controller has been found is more than 1G RAM. Nonetheless, OpenDaylight provides a set of preconfigured VMs with the Hydrogen release of the controller installed. Such VMs contain a GNU/Linux Ubuntu server 13.04 distribution and recommend to allocate 4GB of RAM memory to run the controller. Although the OpenDaylight controller is developed as a normal Java project, it makes heavy use of the Xtend language in some places. While development is possible with bare tools, it is recommend to use Eclipse with the Xtend plugin.
Reference target OS	Recent Linux distribution is recommended although any OS running a Java 1.7 virtual machine should work.
Runtime dependencies	Java 1.7
Key features to support COSIGN extensions	
Supported OpenFlow	OpenFlow versions 1.0 and 1.3 are supported.

version	
Supported protocols at D-CPI	Netconf, SNMP, OVSDB, PCMM/COPS, SNBI, Plugin2OC, LISP, PCEP, BGP
Support for additional protocols at D-CPI	It is possible to support multiple south-bound plugins in OpenDaylight. Furthermore, OpenDaylight has been designed to allow the extensibility of every part of the architecture including the southbound interface.
Integration with Cloud Management Systems	OpenDaylight has driver for Neutron ML2 (Modular Layer 2) plugin to enable communication between Neutron and OpenDaylight. On the SDN controller side, OpenDaylight has northbound APIs to interact with Neutron and use OVSDB for southbound configuration of vSwitches on compute nodes.
A-CPI language	For the A-CPI, OpenDaylight supports the OSGi framework where Java is the developing language, bidirectional REST employing JSON and DOM API employing XML.
Available core functions or services	The base network service functions available in the OpenDaylight controller are: • Topology Manager • Statistics Manager • Switch Manager • Forwarding Rules Manager • Host Tracker
Available applications	OpenDOVE and VTN projects extist within the OpenDaylight initiative to implement overlay network virtualization. Moreover, OpenStack integration is enabled through a Neutron API.
Available A-CPI methods	The associated module directory for the northbound (REST) API reference content is listed below: • Topology REST API • Host Tracker REST API • Flow Programmer REST API • Static Routing REST API • Statistics REST API • Subnets REST API • Switch Manager REST API • User Manager REST API • Container Manager REST API • Connection Manager REST API • Bridge Domain REST API • Neutron ML2 / Network Configuration API
Support for	ODL-SDNi App entered as an incubation project in May 2014. This project is aimed

C2C communication	to provide establishment of East-West interface (SDNi communication) between multiple opendaylight controllers. The application will be responsible for sharing and collecting information to/from such federated controllers.
Exploitability	
License	Eclipse Public License (EPL-1.0)
Membership	Not required
Potential impact	
Major industrial partners	IBM, CISCO, BROCADE, MICROSOFT, JUNIPER, DELL, HP, ERICSSON, INFINERA, INTEL, ORACLE, …
Expert partners from COSIGN consortium	IBM, NEXTWORKS, UNIVBRIS, UPC
Popularity	High. OpenDaylight has a strong support from both the industry and the Open Source community.
Usage in research projects	LIGHTNESS, STRAUSS, CONTENT
Usage in industrial products	Brocade has developed a commercial controller built directly from OpenDaylight code: The Brocade Vyatta Controller.

2.4.2 Floodlight

Floodlight (FL) is a Java based, modular SDN controller platform. The last official release of FL was in December 2014, and the community is quite active. FL is well integrated with Eclipse IDE and it uses Maven + Java1.7 for development. It does not use OSGi.

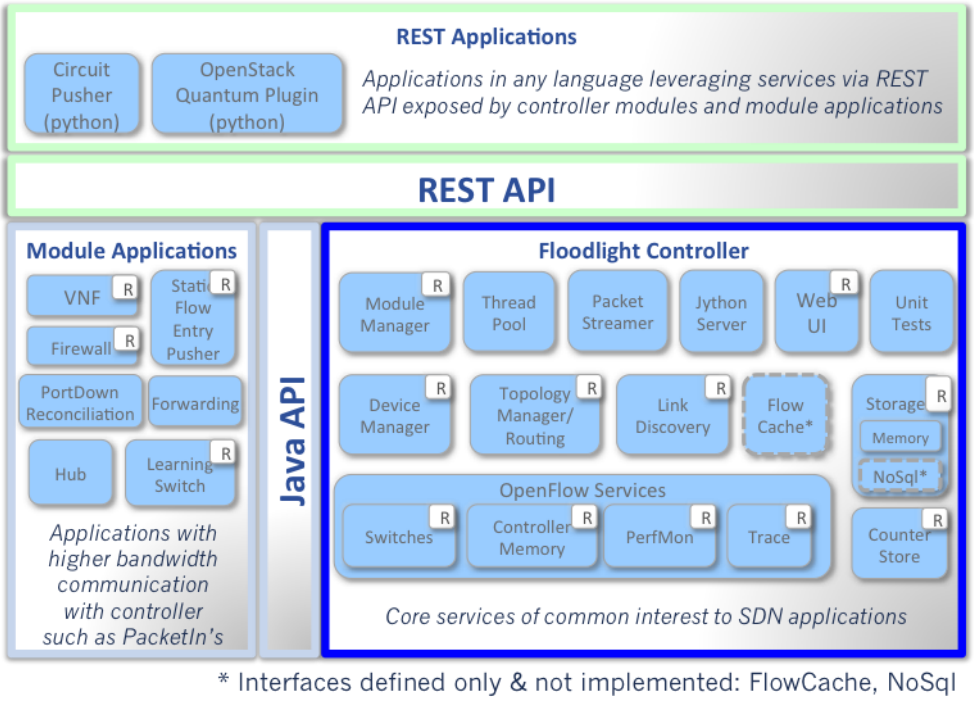
FL does not provide integration with a database. However, it contains an internal API for integration with databases and it currently provides an in-memory key-value store through this API. There is no a High Availability (HA) solution implemented in FL.

Existing functions (also providing a REST API) are:

- Firewall: reactively allows/denies flows based on rules (packet header matching) defined at the controller;
- Load Balancer: supports load balancing for ICMP, UDP and TCP services;
- Virtual Network Filter: network virtualization module which provides a REST API for integration with OpenStack. It allows traffic between MAC addresses (VMs) based on their association with a virtual network: the Virtual Network Filter permits traffic flows between two VMs only if they belong to the same virtual network;
- Forwarding (hop-based shortest path): can work with OpenFlow (OF) islands connected through non-OF devices.

The D-CPI implements the OpenFlow 1.0 and OpenFlow 1.3 in the official release, through a unified API.

The A-CPI is based on RESTful web services. The A-CPI provides capabilities for interrogation of the network topology, setting up OF rules in devices, setting up virtual networks (through Virtual Network Filter) and defining firewall rules. FL integrates with OpenStack through the REST API provided by the Virtual Network Filter module.

Floodlight	
Web-site	http://www.projectfloodlight.org/floodlight/
Brief description	Floodlight is a Java based, modular SDN controller platform. Some of its most important benefits are: 1) It is a stable controller with significant support from the open source community. 2) It is well integrated with Eclipse IDE, and relatively easy to use / develop. 3) Full support for OF 1.3, and experimental support for OF 1.1, OF 1.2 and OF 1.4. Some of the weak points of FL are: 1) It does not provide a HA solution. 2) Last official release was in December 2014. 3) It does not come with a database although it contains an internal API for integration with databases and it currently provides an in-memory key-value store through this API.
Software	
Latest sw version and release date	The latest official version of Floodlight is v1.0 and its release date is 30th of December 2014.
Software architecture	The next picture shows the architecture of Floodlight.  Figure 4 – FL architecture

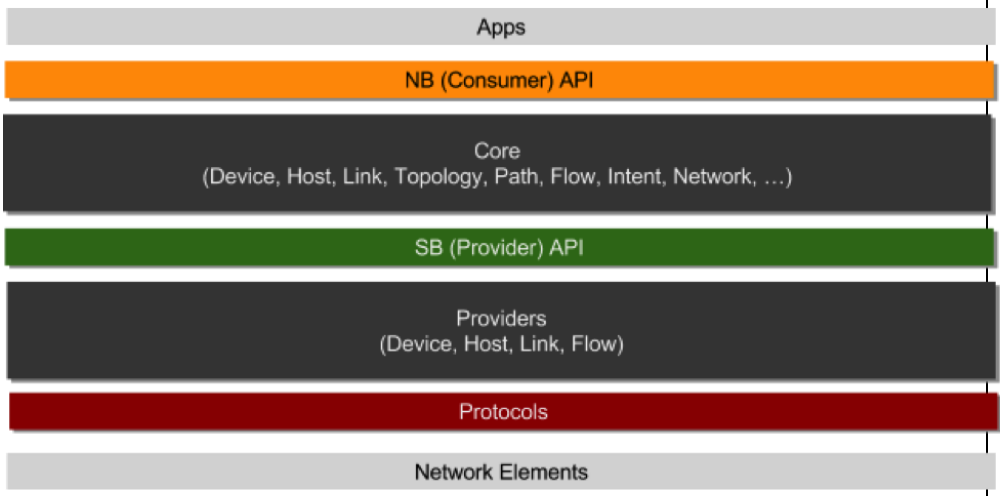
	Floodlight contains a set of modules (the blue rectangle in *Figure 4*), which provide the core functionality for the controller. The modules marked with the letter "R" expose their functionality through REST API. The functionality of the core modules can also be accessed by other modules within Floodlight, through the internal Java API. Floodlight also provides some additional internal Java API-based modules, named "Module Applications" in *Figure 4*. Floodlight also comes bundled with two applications which use the REST API of the controller: (1) Circuit Pusher and (2) an OpenStack plugin. All the modules can be started/stopped from a configuration file, only at compile time.
Source code distribution	Floodlight can be fetched from a git repository located on GitHub (https://github.com/floodlight/floodlight).
Software governance	Floodlight is distributed with Apache license. The exact conditions for using and distributing Floodlight are stated in https://github.com/floodlight/floodlight/blob/master/LICENSE.txt. To contribute to the Floodlight open source project, one must first get the source code from Github. The next step is to modify the source code, modification which constitutes the contribution. The added/modified code must be unit tested, documented (purpose, design, usage). The Apache license header must be added to the source code files. *No further information has been found regarding the process for getting the contribution accepted by the community.
Sw package distribution	One can get Floodlight directly from Github or as a .gz/.zip file. There is also a VM with the pre-installed controller (http://www.projectfloodlight.org/download/).
Core language	Java
Applications language or interface	The controller modules can be written in Java. Applications that make use of the REST API can be written in any language as long as they correctly interface with Floodlight using the REST API.
Usability	
Installation documentation	The up to date documentation for installation of Floodlight is provided in: http://docs.projectfloodlight.org/display/floodlightcontroller/Installation+Guide The installation steps are detailed enough and easy to follow.
Operative documentation	Up to date documentation about how to operate Floodlight is provided in: http://docs.projectfloodlight.org/display/floodlightcontroller/Installation+Guide The documentation is detailed enough and easy to follow.
Development	
Development documentation	Detailed, up to date documentation for developers is provided in: http://docs.projectfloodlight.org/display/floodlightcontroller/For+Developers It covers the architecture, software design, and tutorials for how to develop core modules in Floodlight. It also explains how to expose new services (Java or REST) from the developed modules.

Documentation for application development	A detailed tutorial of the steps required to write applications that use the REST API of Floodlight is presented in: http://docs.projectfloodlight.org/display/floodlightcontroller/For+Developers
Development roadmap	There is no clear development roadmap at this moment.
Toolchain	Java 1.7 + Maven
IDE integration	Eclipse: tested and proved to work very well. It may easily work with any IDE since the project uses very common features that are available in any IDE.
Deployment	
System requirements	No information regarding system requirements has been found.
Reference target OS	Recommended: - Ubuntu 10.04 or higher; - MAC OS X 10.6 or higher;
Runtime dependencies	Java 1.7
Key features to support COSIGN extensions	
Supported OpenFlow version	Supports OF1.0 and OF1.3. Experimental support for OF 1.1, OF1.2 and OF 1.4 exists.
Supported protocols at D-CPI	There is no support for other protocols apart from OF, on the D-CPI.
Support for additional protocols at D-CPI	There is no standard support for multiple southbound plugins. It is relatively easy to add a new module in Floodlight which can serve as a southbound plugin.
Integration with Cloud Management Systems	One of the application modules provided by Floodlight (*Figure 4*) is the Virtual Network Filter (VNF), which provides a REST API for integration with Quantum/Openstack. It is not mentioned if the REST API is still compatible with newer version of the Openstack networking API (e.g. Neutron, ML2).
A-CPI language	The provided A-CPI is REST based and it uses JSON for data formatting. Internal Floodlight application modules can be written in Java.
Available core functions or services	The available core functions of Floodlight are: - Device inventory (hosts). - Topology manager (inventory of switches, link, etc.).

	- Static flow entry pusher. - Monitoring the performance of the controller. - Forwarding (Dijkstra's SPF), works also for OF islands interconnected through non-OF switches.
Available applications	It does not support applications related to COSIGN, such as virtualization, overlays, etc. It provides a simple plugin for OpenStack integration (i.e. Virtual Network Filter).
Available A-CPI methods	The available REST API on the A-CPI supports the following functions: - retrieval of network and topological elements (attached hosts, switches, links, etc.) - retrieval of OF statistics about the network. - data about the controller (memory usage, status, etc.) - set up static flows in the network - set up firewall rules - functions for Quantum OpenStack plugin through the Virtual Network Filter (setting up virtual networks).
Support for C2C communication	Floodlight does not provide support for any type of C2C communication, that is: - no HA; - no east-west bound interface; - no hierarchical controllers;
Exploitability	
License	Apache license.
Membership	No
Potential impact	
Major industrial partners	The major industrial partner is Big Switch Networks which is the maintainer of the Floodlight project.
Expert partners from COSIGN consortium	DTU has experience of developing with Floodlight.
Popularity	Floodlight was the preferred open source SDN controller before the release of OpenDaylight. There are other controllers that reuse parts Floodlight at their core. An example of such a controller is OpenIRIS [OpenIRIS].
Usage in research projects	A list of research projects based on Floodlight can be found in: http://docs.projectfloodlight.org/display/floodlightcontroller/Floodlight+Projects

Usage in industrial products	Floodlight represents the core of the commercial Big Switch Network Controller.

2.4.3 ONOS

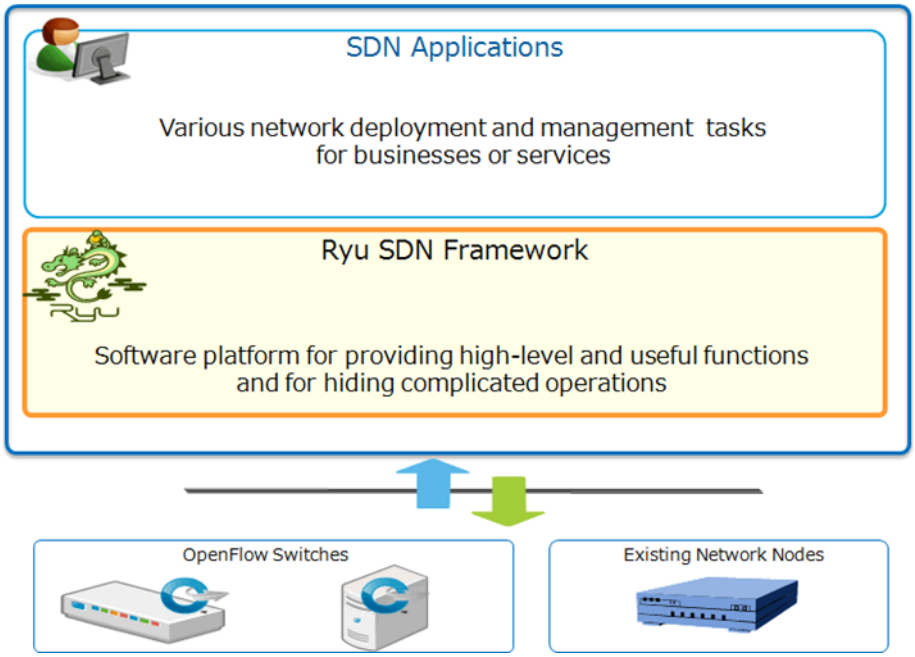
ONOS (Open Network Operating System)	
Web-site	http://onosproject.org/
Brief description	ONOS is an open source SDN controller platform, which is supported by ON.LAB. ON.LABis an organization that maintains other open source projects such as OpenVirtex, Mininet, and OpenCloud. (http://onlab.us/projects/) Several instances of ONOS can run, simultaneously, over multiple (clustered) servers, to provide performance, scalability and high availability. The first release of ONOS was on 5th of December 2014. Several use cases are bundled with ONOS, such as packet-optical integration, network functions as a service, BGP-based integration with traditional networks, etc. The ONOS controller has a modular architecture based on Karaf (an OSGi framework), where the modules can be managed (e.g. activated/deactivated) at run time. Particular to ONOS is a northbound API, named the "Intent framework", which allows administrators to set up policies regarding the desired behaviour of the network. These policies (intentions) are compiled by ONOS and translated into OpenFlow commands inserting the necessary flow entries and configuration in the network devices. Moreover, ONOS preserves the policies when the network is changing, by automatically reacting to the changes (e.g. link failures) and re-compiling the policies into new sets of flow entries.
Software	
Latest sw version and release date	5th of December 2014,
Software architecture	 The figure shows the high level architecture of ONOS. At the lowest layer there are the network elements which ONOS controls. The next tier, Protocols, contains the protocol controllers (e.g. OpenFlow) that are used to communicate with the network devices.

	The Providers tier contains several plugins (providers) that provide low level services for managing the network. Some of the plugins are protocol-aware, and they are responsible for protocol specific communication with the network, while other plugins combine different services to provide more useful services and data to the ONOS core. The SB (Provider) API tier is an API that allows providers to plug into the ONOS core. The Core tier is the main subsystem of ONOS. It contains several managers that deal with various aspects of network abstractions: Devices, Hosts, Topology, etc. The Core is based on a distributed in-memory data store. This allows clustering and high availability. The NB (Consumer) API is an API that allows external entities to interact with ONOS. This comprises the Intent framework and other REST APIs (discussed later).
Source code distribution	Github: https://github.com/opennetworkinglab/onos
Software governance	In general, ONOS is led by a board of advisors and the community. Contributions from the community are encouraged. A detailed explanation about the governance in ONOS can be found here: https://wiki.onosproject.org/display/ONOS/ONOS+Governance Also, the mailing lists (for discussions regarding contributions and others)are: https://wiki.onosproject.org/display/ONOS/ONOS+Mailing+Lists Finally, a guide for how to contribute to ONOS is here: https://wiki.onosproject.org/display/ONOS/A+Beginner%27s+Guide+to+Contribution
Sw package distribution	Zip, tar, and VM: https://wiki.onosproject.org/display/ONOS/Downloads
Core language	Java
Applications language or interface	The controller modules can be written in Java and must adhere to the OSGi style to be possible to integrate them in ONOS. Interfaces towards external entities are based on REST APIs.
Usability	
Installation documentation	The installation documentation is up-to-date and detailed: https://wiki.onosproject.org/display/ONOS/Installing+and+Running+ONOS https://wiki.onosproject.org/display/ONOS/Development+Environment+Setup
Operative documentation	There is a user guide on the wiki: https://wiki.onosproject.org/display/ONOS/User%27s+Guide
Development	
Development documentation	There is a developer guide: https://wiki.onosproject.org/display/ONOS/Developer%27s+Guide
Documentation for application	There are several tutorials for writing applications in ONOS:

development	https://wiki.onosproject.org/display/ONOS/Tutorials+and+Walkthroughs
Development roadmap	ONOS will have releases every 3 months. The next release is planned for 28th of February 2015. A summary for the next release is given here: https://wiki.onosproject.org/display/ONOS/Upcoming+Release+Contents#UpcomingReleaseContents-Blackbird. Among other features, the next release will include enhancements for OF1.3 and a new southbound plugin (maybe OVSDB as mentioned on the official web-site).
Toolchain	E.g. Java 8 + Maven + Karaf (OSGi)
IDE integration	IntelliJ IDEA, Eclipse
Deployment	
System requirements	For a VM running ONOS, the following are recommended: • Ubuntu Server 14.04 LTS 64-bit • 2GB or more RAM • 2 or more processors
Reference target OS	ONOS was tested on OS X and Ubuntu 14.04 64 bit.
Runtime dependencies	Java 8, Apache Karaf 3.0.2 or later, Apache Maven 3.0 or later
Key features to support COSIGN extensions	
Supported OpenFlow version	OF 1.0, OF 1.3. The current version of ONOS does not support multiple tables for OF 1.3. This is scheduled for the next release in February 2015. Extensions for optical control with OF 1.3 are also supported in the current release. The extensions allow matching on optical channel signal type (a byte indicating the type) and signal ID, where the signal ID is given by the grid type, the channel spacing, the channel number and the spectral width.
Supported protocols at D-CPI	OF
Support for additional protocols at D-CPI	No additional protocols are currently supported. The architecture of ONOS allows easy integration of other plugins on the southbound.
Integration with Cloud Management Systems	The NFaaS use case shows integration with OpenCloud, which is a cloud platform based on Openstack components. https://wiki.onosproject.org/pages/viewpage.action?pageId=2130965 http://opencloud.us/faq.html
A-CPI language	REST web services for external applications.

	Java APIs for internal controller modules.
Available core functions or services	• Device Subsystem - manages the network devices • Link Subsystem - manages the network links • Host Subsystem - manages the hosts attached to the network • Topology Subsystem - manages the network topology as a graph • PathService- path computation service • FlowRule Subsystem - manages the flows rules installed in the network • Packet Subsystem - manages the interaction using OF packets (listen and send)
Available applications	The use cases provided with ONOS might be useful for COSIGN: - Packet- optical integration https://wiki.onosproject.org/display/ONOS/Packet+Optical - NFaaS https://wiki.onosproject.org/pages/viewpage.action?pageId=2130965
Available A-CPI methods	A summary of the supported REST APIs is given here: https://wiki.onosproject.org/pages/viewpage.action?pageId=1048699#AppendixB:RESTAPI%28Draft%29-Device
Support for C2C communication	ONOS supports HA in the form of clustering.
Exploitability	
License	Apache 2.0 license
Membership	No
Potential impact	
Major industrial partners	AT&T, Ciena, Ericsson, Fujitsu, Huawei, Intel, NEC, NSF, NTT Communications
Expert partners from COSIGN consortium	None
Popularity	ONOS is newly released.
Usage in research projects	--
Usage in industrial products	--

2.4.4 Ryu

SDN controller name	
Web-site	http://osrg.github.io/ryu/
Brief description	Ryu provides software components with well defined API that make it easy for developers to create new network management and control applications. Ryu supports various protocols for managing network devices, such as OpenFlow (1.0, 1.1, 1.2, 1.4 and Nicira Extensions), Netconf, OF-config, etc. Best fit for network-application prototyping.
Software	
Latest sw version and release date	Latest release 3.13 Sept. 2014. Monthly minor release (3.12 Aug. 2014)
Software architecture	 Written entirely in Python programming language, the Ryu framework is component based. Existing components include south bound protocols support, event management, messaging, in-memory state management, application management, infrastructure services and a series of reusable libraries (e.g., NETCONF library, sFlow/Netflow library).
Source code distribution	git://github.com/osrg/ryu.git
Software governance	Apache 2.0 license, PEP8 coding style compliance expected of contributors.
Sw package distribution	Installable using standard Python's PIP tool with "pip install ryu" command. Prepackaged VMs available for evaluation: http://osrg.github.io/ryu/resources.html

Core language	Python 2.6+, greenlets, cooperative pseudo-multithreading (utilizing single core).
Applications language or interface	Python 2.6+ for controller applications, REST for external clients.
Usability	
Installation documentation	Available, up-to-date, detailed.
Operative documentation	Available, up-to-date, detailed.
Development	
Development documentation	Available, up-to-date, not very detailed.
Documentation for application development	Available, up-to-date, detailed. Includes some tutorials.
Development roadmap	Describe, if available.
Toolchain	Python 2.6+ with following libraries: • python-eventlet • python-routes • python-webob • python-paramiko
IDE integration	Any text editor, Python plugin exists for Eclipse. Community edition of PyCharms is another good option.
Deployment	
System requirements	Single lightweight python process.
Reference target OS	Recent GNU/Linux.
Runtime dependencies	Python packages • python-eventlet • python-routes • python-webob • python-paramiko
Key features to support COSIGN extensions	

<table>
<tr><td>Supported OpenFlow version</td><td>1.0 through 1.4 with Nicira extensions.</td></tr>
<tr><td>Supported protocols at D-CPI</td><td>Netconf, OF-config, SNMP, OVSDB.</td></tr>
<tr><td>Support for additional protocols at D-CPI</td><td>Possible to support multiple south-bound plugins. Relatively easy to create new plugins.</td></tr>
<tr><td>Integration with Cloud Management Systems</td><td>Has OpenStack plugin. Using REST APIs.</td></tr>
<tr><td>A-CPI language</td><td>Python for internal applications.
REST web services for external applications.</td></tr>
<tr><td>Available core functions or services</td><td>Parsers and generators for packets of different network protocols. Ability to start plugin’s own processing loop.</td></tr>
<tr><td>Available applications</td><td>A dumb OpenFlow 1.0 responder for benchmarking the controller framework. Intended to be used with oflops cbench.
An OpenFlow 1.0 L2 learning switch implementation.
MAC address based isolation logic.
VLAN based isolation logic.
Flow table updater for OpenStack integration
Switch and link discovery module. (Planned to replace ryu/controller/dpset.)</td></tr>
<tr><td>Available A-CPI methods</td><td>http://ryu.readthedocs.org/en/latest/components.html lists the following extensions/plugins supporting Ryu OpenStack integration (documentation still uses “Quantum” name)

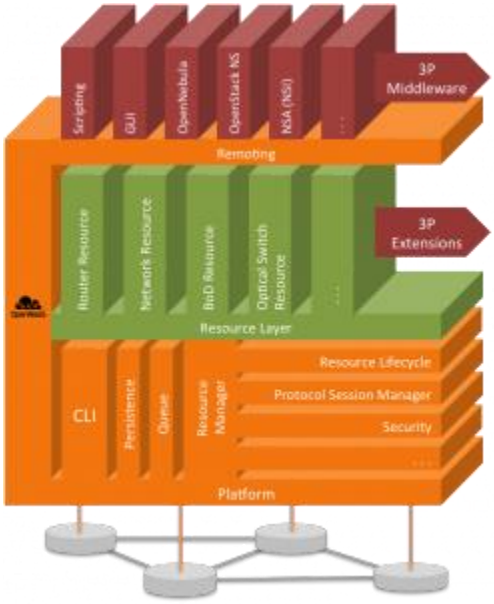
<table><tr><td>Application</td><td>Purpose</td></tr><tr><td>ryu.app.gre_tunnel</td><td>Flow table updater for OpenStack integration. Despite of the name, this isn’t GRE specific.</td></tr><tr><td>ryu.app.tunnel_port_updater</td><td>This module updates OVS tunnel ports for OpenStack integration.</td></tr><tr><td>ryu.app.quantum_adapter</td><td>Listen OpenFlow port status change notifications from switches. Consult ovsdb to retrieve the corresponding port uuid. Notify relevant parties, including quantum (via Ryu plug-in) and Ryu applications. (via Ryu Events)</td></tr></table></td></tr>
</table>

	ryu.app.rest	This module provides a basic set of REST API.
		• Network registration
		• End-point port management
		  o OpenFlow port number
		  o MAC address (for anti-spoofing)

	ryu.app.rest_conf_switch	This module provides a set of REST API for switch configuration. - Per-switch Key-Value store
	ryu.app.rest_quantum	This module provides a set of REST API dedicated to OpenStack Ryu plug-in. • Interface (uuid in ovsdb) registration • Maintain interface association to a network
	ryu.app.rest_tunnel	Provide a set of REST API for tunnel key management. Used by OpenStack Ryu plug-in. • Tunnel key registration for a network • Manage switches and their ports which are used to establish a tunnel
Support for C2C communication	Some infrastructure preparations for HA using Zookeeper exist. The actual use of HA capabilities would be provided by networking apps.	
Exploitability		
License	Apache 2.0 license	
Membership	No	
Potential impact		
Major industrial partners	NTT	
Expert partners from COSIGN consortium	None	
Popularity	Mainly used in academic studies, since it is easy to learn and to use for creating new simple applications.	

Usage in research projects	--
Usage in industrial products	--

2.4.5 OpenNaaS

OPENNAAS	
Web-site	http://opennaas.org/
Brief description	See website http://opennaas.org/
Software	
Latest sw version and release date	Opennaas-0.30 released on 03/11/2014
Software architecture	 See slides here: http://www.slideshare.net/sergifiguerola/opennaas-singapoure-glif2013
Source code distribution	Source code available in github: https://github.com/dana-i2cat/opennaas/tree/master
Software governance	OpenNaaS is open source and driven by the needs of its community. If you are using OpenNaaS and want to be updated on new features and releases, and participate in the discussion of feature requests, then join the OpenNaaS users mailing list. If you want to write extensions for OpenNaaS or want to know what's happening at the OpenNaaS core development team, join the OpenNaaS developers mailing list. More information in http://opennaas.org/partners/

Sw package distribution	OpenNaaS is shipped in an executable distribution. The distribution is a zip file.
Core language	Java
Applications language or interface	OpenNaaS offers a Java interface and a REST API as A-CPI. Java applications consuming the Java interface may be integrated inside the OpenNaaS platform. Remote applications can use the REST API, or any other API offered by third-party applications.
Usability	
Installation documentation	http://confluence.i2cat.net/display/OPENNAAS/Deploy+and+run+OpenNaaS
Operative documentation	http://confluence.i2cat.net/display/OPENNAAS/Load+resources+in+OpenNaaS
Development	
Development documentation	http://confluence.i2cat.net/display/OPENNAAS/Developers+Guide
Documentation for application development	http://confluence.i2cat.net/display/OPENNAAS/Developers+Guide Basic Tutorial: http://new.opennaas.org/create-a-new-resource/
Development roadmap	Available in following slides: http://www.slideshare.net/sergifiguerola/opennaas-singapoure-glif2013 (page 17)
Toolchain	Java 7 + Maven + OSGi
IDE integration	Eclipse
Deployment	
System requirements	The minimal system requirements are: Oracle's Java Runtime Environment (JRE) 1.7.x (Java 7). About 100 MB of free disk space, needed for running full OpenNaaS assembly
Reference target OS	GNU/Linux Ubuntu 14.04 64 bit, although we have successfully tested OpenNaaS in Windows 7, Ubuntu 12.04 and MAC OSX systems too.
Runtime dependencies	Oracle Java 7 JRE
Key features to support COSIGN extensions	
Supported OpenFlow version	OpenNaaS does not directly support OpenFlow. Its goal is to orchestrate multiple SDN controllers which may themselves support OpenFlow. Applications using Floodlight and OpenDayLight are currently being developed.
Supported	OpenNaaS currently supports Netconf, and some proprietary protocols. It also

protocols at D-CPI	includes clients for web services API of other OF controllers (i.e. Floodlight-v0.9) and other provisioning applications (i.e. GEANT AutoBAHN).
Support for additional protocols at D-CPI	Adding supporting new protocols is supported by adding new modules to OpenNaaS.
Integration with Cloud Management Systems	OpenNaaS is a NaaS platform. Cloud Management Systems may use its services in order to configure desired network connectivity. In order to do so, a plugin in each of these systems consuming OpenNaaS API has to be developed. There is currently one of this plug-ins for the old OpenStack Quantum very limited functionality. There is currently no application in OpenNaaS consuming any Cloud Management System API.
A-CPI language	REST web services. Integrated application may consume the Java interface directly.
Available core functions or services	Device inventory, service execution, network level transactions.
Available applications	Device virtualization.
Available A-CPI methods	Resource reservation. Creation of virtual resources on demand. Retrieval of network resources.
Future applications and A-CPI methods	Retrieval of port and flow monitoring information. Circuit Reservation. Management of resources of type Network as a whole. PCE.
Support for C2C communication	Currently not supported. In the future, each resource of type Network may have its own controller application, which may be able to call others in a flat and/or hierarchical structure.
Exploitability	
License	OpenNaaS core is LGPLv3, applications, including the SDNController are licensed under ASFv2.
Membership	Not required
Potential impact	
Major industrial partners	Juniper
Expert partners from COSIGN consortium	I2CAT Foundation
Popularity	Medium. Thoroughly known in Research communities and forums.
Usage in research	Mantychore FP7, OFERTIE, CONTENT, SODALES, XIFI, COSIGN,

projects	GEANT3+
Usage in industrial products	OpenNaaS has been tested and works on top of JunOS 10 suite.

2.4.6 Summary

The following table summarizes and compares the main features for the analysed SDN controllers.

Table 1 – Comparison of SDN controllers

Feature	**OpenDaylight**	**Floodlight**	**ONOS**	**Ryu**	**OpenNaaS**
Last official release	September 2014	December 2014	December 2014	September 2014	March 2014
Core language	Java	Java	Java	Python 2.6+ greenlets	Java
IDE integration	Eclipse, only for the Controller and the Openflow plugin.	Eclipse	IntelliJ IDEA Eclipse	Any text editor. Eclipse through plugins.	Eclipse
OSGi	Yes	No	Yes	No	Yes
Development tools	Java 1.7 + Maven + OSGi	Java 1.7 + Maven	Java 8 + Maven + OSGi	Python 2.6+ Python-eventlet Python-routes Python-webob Python-paramiko	Java 7 + Maven + OSGi
Database integration	No	No	No	No	No
Openflow version	1.0 and 1.3 in official release	1.0 and 1.3 in official release 1.3- actively developed	1.0 and 1.3 (without support for multiple tables and with extensions for optical layer)	1.0 -1.4 with Nicira extensions	No direct support.
Additional D-CPI protocols	Netconf, SNMP, OVSDB, PCMM/COPS, SNBI, Plugin2OC, LISP, PCEP, BGP	No	No	Netconf, OF-Config, SNMP, OVSDB	Netconf Proprietary protocols

Source code distribution	Git repository	Git repository	Git repository	Git repository	Git repository
Sw. package distribution	.gz/.zip file. Virtual machine	.gz/.zip file. Virtual machine	.gz/.zip file. Virtual machine	Install using python “pip” tool. Virtual machine	.zip file.
Software governance (licensing)	Eclipse public license (EPL-1.0)	Apache license	Apache 2.0 license	Apache 2.0 license	LGPLv3 (core) ASFv2 (applications)
Applications/ services language	Java for internal modules (same address space). REST APIs for external apps.	Java for internal modules (same address space). REST APIs for external apps.	Java for internal modules (same address space). REST APIs for external apps.	Python 2.6+ for internal modules. REST APIs for external apps.	Java for internal modules (same address space). REST APIs for external apps.Any for external apps.
A-CPI language	REST	REST: HTTP + JSON	REST	REST	REST
Cloud management systems integration	Openstack Neutron ML2	Openstack (Quantum)	OpenCloud (maintained by ON.LAB)	Openstack	Openstack (Quantum)
Available virtualization applications	OpenDove Virtual Tenant Network Integration with OpenStack through Neutron API	Virtual Network Filter- provides the integration with Openstack (MAC -based isolation logic).		MAC and VLAN-based isolation logic.	Device virtualization.
High Availability	Yes, through clustering	No	Yes, through clustering.	Some support for HA, using Zookeeper, exists.	No
East-West bound interface	Project for East-West interface entered incubation in May 2014.	No	No	No	No

Hierarchical controllers	No	No	No	No	No
Expert COSIGN partners	IBM, NEXTWORKS, UNIVBRIS, UPC	DTU	-	-	I2CAT

2.5 Gap Analysis and COSIGN Control Plane Work Positioning

The state of the art survey shown in the previous sections has highlighted how current Data Centre solutions are still relying on cloud management platforms which use the control plane of the DCN mainly as a simple "arm" to enforce network configuration on the data plane. Following this model, the interaction is mostly unidirectional, based on a client-server principle where the control plane executes some commands to create the network virtual resources requested by the cloud management. This approach brings to limited orchestration features, where IT (i.e. computing and storage) and network resources are handled separately following their own criteria, without any real cross-layer awareness of DCN conditions and capabilities or cloud applications' traffic constraints and profiles between cloud platform and DCN control plane. Network virtualization based on pure overlay mechanisms needs to assume the underlying network connectivity as an always present resource, leading to static configurations, overprovisioning and underutilization of the whole DCN.

COSIGN SDN-based control plane, mixed with the capabilities of the optical data plane, brings a new level of programmability to the DCN infrastructure. This is particularly suitable to be exploited by the upper layer applications, and in particular by the orchestration functions at the cloud management platform, to implement a closer bi-directional interaction and cooperation at the control level between the IT and the network side of the Data Centre infrastructure. This approach is very promising to achieve better utilization and automation at the DCN level, while providing more guarantees at the network connection level for the cloud services, for example in terms of QoS or service reliability. Ù

For these reasons, the COSIGN architecture defines powerful interfaces between the cloud orchestrator (to be developed in WP4) and the SDN controller (implemented in WP3). The objective is to bring high degrees of network awareness towards the cloud platform to enable a convergent virtualization and management of IT and network resources. On the other hand, the network knowledge exported at the cloud service level needs to be easy to manipulate. Thus suitable abstraction functions at the SDN controller will hide the technology complexity and heterogeneity that characterize the data plane.

On the opposite direction, the cloud platform will inject some knowledge of the cloud applications' requirements into the DCN control plane, with different types of models or levels of detail. This information may include just application-based policies that need to be automatically translated into a network configuration at the SDN controller. But they may also provide further details as expected traffic profiles, planned management actions (e.g., VM migrations or content replications for scheduled backups), specific requests with virtual infrastructures topologies and network functions, even combined with preferred locations or service recovery models, depending on the use case. The DCN control plane needs to implement the functions to satisfy these different requests in an automated manner (i.e., without requiring any manual network configuration from the DC administrator). Moreover, internal algorithms and procedures should be integrated with the objective of maximizing the DCN utilization considering not only the constraints and the capability of the data plane, but also the characteristics of the high level applications. This procedures may involve the reconfiguration of the data plane and the modification of the underlying network connectivity, however they should be fully transparent from the applications' and the cloud platform's point of view, without any impact or service disruption at the overlay networks' level.

3 COSIGN Control Plane High Level Architecture and Services

This section describes the functional architecture of the COSIGN DCN Control Plane (CP), identifying the services exposed towards the other architectural layers and the CP roles in the whole COSIGN workflows. Different deployment models compatible with the recursive SDN architecture defined by ONF are discussed, presenting their applicability in DC scenarios. Finally, starting from the control plane requirements defined in D1.3 [2], the final section analyses how these requirements impact the control plane architecture in terms of internal functions and external interfaces, providing a starting point to derive the control plane components that will be the object of Section 4.

3.1 General Architecture of the COSIGN Control Plane

The DC Control and Management layer is built upon Software Defined Networks (SDN) capabilities to leverage the added value of emerging optical technologies and aims to provide a converged network and IT virtualization platform for Data Centres. The DC Control and Management layer will result from the joint effort of WP3 and WP4, where WP3 will mostly focus on the DCN control aspects, while WP4 will address the whole DC management, including network, computing and storage resources, together with the orchestration of cloud services and applications. *Figure 5* outlines the most important features and capabilities for each of the COSIGN architectural layers, with particular attention to the DCN Control Plane. These capabilities and how they relate to the internal CP components or to other COSIGN layers will be further discussed in the next sections.

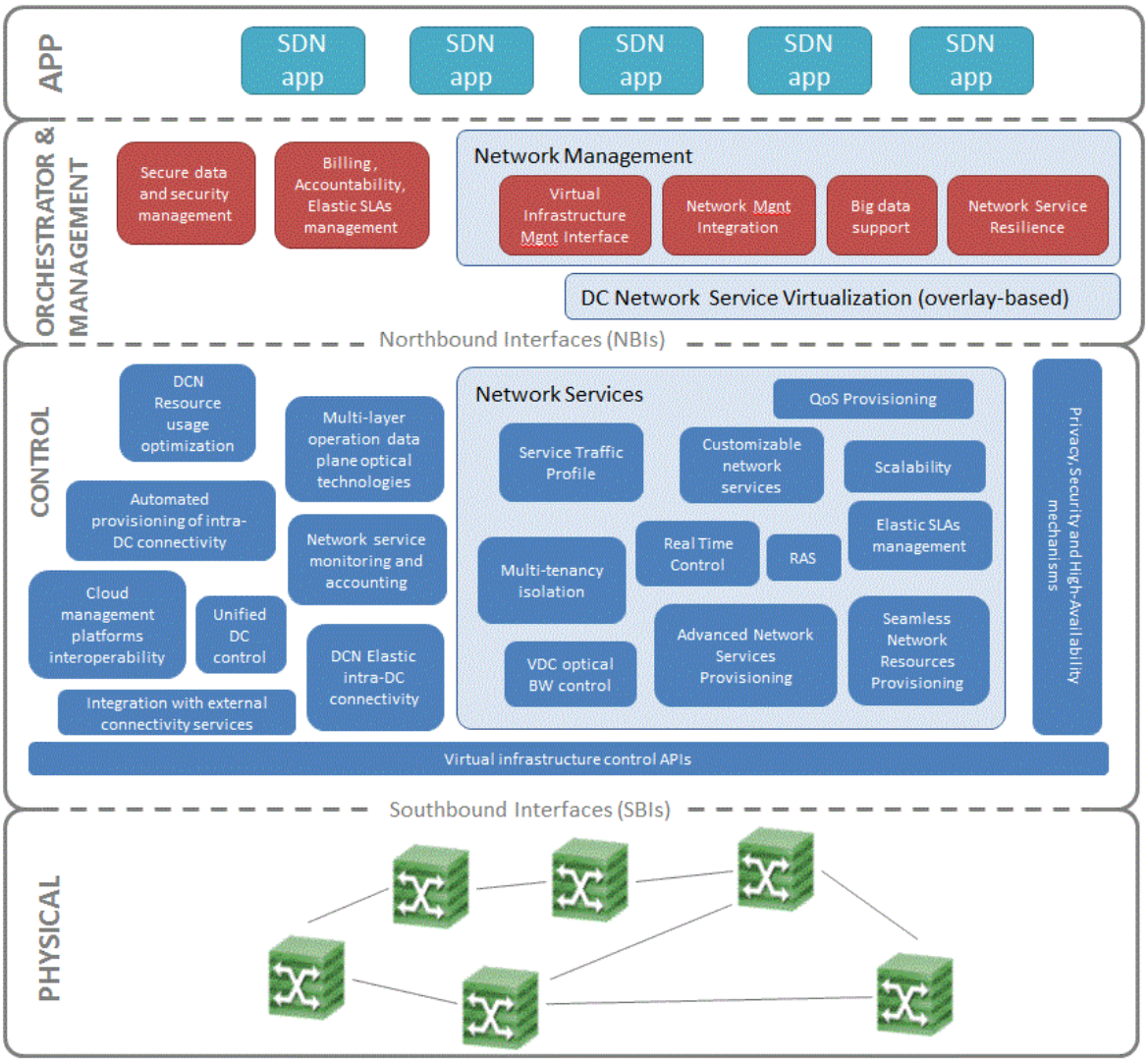


Figure 5 – Functions and capabilities of COSIGN architectural layers

In COSIGN architecture, the DCN Control Plane is placed between the DCN physical layer (WP2) and the DC Orchestration and Management plane (WP4), which is implemented through a cloud management platform (e.g. OpenStack). The cross-layer interaction is enabled through the northbound interface with the Orchestration and Management plane and the southbound interface with the DCN optical data plane. Considering a deployment where the CP functions are implemented in a centralized SDN controller (optionally with the contribution of some external SDN applications), these interfaces can be easily mapped respectively on the A-CPI and D-CPI interfaces defined in ONF (see Section 2.3 for details).

In this view, the DCN Control Plane can be considered as a sort of mediator between the optical hardware devices deployed at the DCN infrastructure and the software platform, running at the upper layer, which is in charge of coordinating the whole DC actions for the provisioning of cloud services. In this role, the control plane needs to ensure a twofold objective:

- Translate the abstract network-related directives generated from the software cloud platform and enforce them through the actual configuration of the DCN data plane.
- Guarantee the efficient utilization of the DCN data plane, in terms of features and capabilities, to provide the DC network connectivity required by the upper layer software cloud platform, in support of on-demand IaaS provisioning and DC management actions.

3.1.1 SDN Architecture of COSIGN DCN Control Plane

The role of the DCN Control Plane, and more in general the WP3 scope, within the SDN architecture defined by ONF is presented in *Figure 6*.

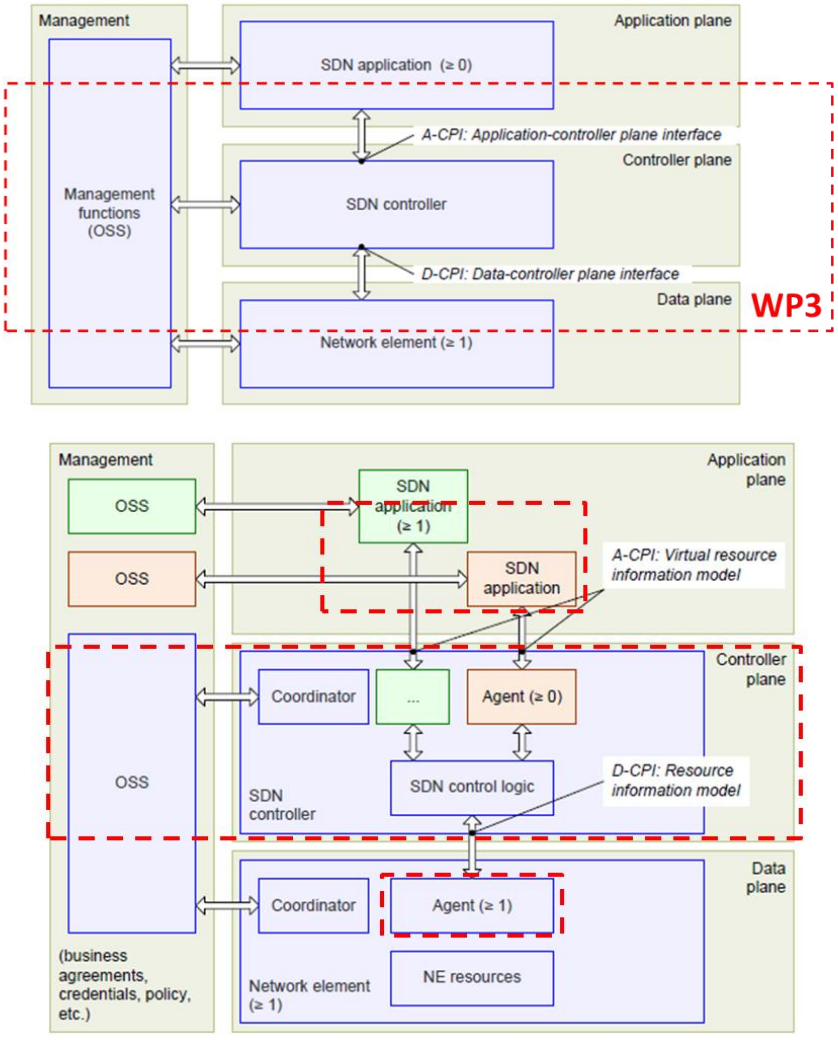


Figure 6 – WP3 role in SDN architecture

The *Network Elements* (NEs) represent groups of data plane resources of COSIGN DCN managed as single entities at the control plane level. In general, in COSIGN a Network Element corresponds to a single network device deployed in the DCN infrastructure. However, the concept of aggregating more devices under a single Network Entity which expose a global, single interface towards the control plane is particularly useful in DC environments since it allows for integration of portions of the DCN with devices controlled by proprietary management solutions. This type of interaction will be not further considered in this work, since our focus is on the SDN-based operation of the COSIGN data plane optical resources. These devices directly provide interfaces suitable for the D-CPI, through dedicated *Agents* developed in WP3, without any mediation of proprietary control or management systems.

The *SDN controller* is a logically centralized entity, with a synchronized and self-consistent view of the network under its exclusive control. Most of the COSIGN DCN Control Plane functions will be implemented within an SDN controller, through internal components which implement the SDN control logic for core network services. The *SDN applications*, running at the ONF Application Plane, can invoke services exposed by the controller (in this sense the orchestrator developed in WP4 is a typical example of SDN application) or can be invoked by the controller itself. Depending on the software design, some of the DCN Control Plane functional components can be developed in COSIGN as SDN applications, external from the controller. These network applications can provide support functions that can be re-used by the controller itself to implement its own services. However, they may also provide abstracted network services with their own APIs towards other COSIGN components, for example the cloud orchestrator or even other SDN applications at the Control Plane level, actually implementing the well-known SDN concept of recursive abstraction.

Finally, typical *Management Functions* at the control plane level include policy configuration or performance monitoring. In COSIGN, they may be extended to SLA management and configuration, configuration of access/modification rights to Virtual Data Centre (VDC) infrastructures, etc.

3.1.2 COSIGN DCN Control Plane Functions

The control plane functions expose some abstract, technology-independent DCN services (e.g., network connectivity, monitoring, optical infrastructure virtualization, etc.) which are consumed at the orchestrator layer. The integration of these services within the overall cloud and DC platform allows the orchestrator to provide a converged management of computation, storage and network resources, enabling the three use cases defined in deliverable D1.1 [1]:

- Use case 1: On-demand deployment and provisioning of converged Virtual Data Centres (VDCs).
- Use case 2: On-demand deployment and provisioning of application-level overlay virtual networks over programmable optical infrastructures.
- Use case 3: DC operational support and management orchestration.

COSIGN control plane must ensure that the orchestrator is supplied with the appropriate network information to implement its service logic. Therefore, the control plane has to abstract network resources complexity but still providing all the relevant underlying information (monitoring data, physical performance, network service parameters, etc…) to the upper layer in a way that allows the orchestrator to carry out its own tasks and processes.

Interfaces and communication flows between orchestrator and SDN control plane extends the currently available northbound interfaces (e.g. for Create, Read, Update, Delete – CRUD – operations on OpenStack Neutron resources). The objective is to enable the on-demand creation of enhanced and programmable virtual networks (virtual optical slices or overlay networks, depending on the use case), which can be managed and manipulated dynamically as workload requirements change. Therefore the COSIGN architecture includes mechanisms and algorithms for efficient allocation of virtual network resources at the SDN controller (WP3) and for converged virtual IT and network composition at the orchestration and management layer (WP4). Special attention in WP3 is given to the algorithms that exploit those unique features offered by the new optical devices developed in WP2.

The most important capabilities implemented at the COSIGN DCN control plane are outlined below.

Seamless Provisioning of Network Resources

Network resources are provisioned in a transparent end-to-end manner and the infrastructure underpinning the services is configured automatically to achieve fast service deployment (e.g. within minutes).

Advanced Network Services Provisioning

Advanced network services are provisioned on demand, allowing dynamic provisioning and configuration of the network services beyond the simple network connectivity, but including virtual network infrastructures with QoS or resiliency constraints and combined with orchestrated chains of complex network functions.

RAS

COSIGN DCN Control Plane integrates procedures for network service resiliency, in order to contribute to the overall DC service reliability and availability. The strategies for protection or restoration of network connectivity are integrated with the DC management mechanisms for automated and fast disaster recovery implemented at the orchestrator level. Moreover, the DCN Control Plane implements the procedures to establish the network connectivity required for the on-demand or scheduled backups and replications of data in local or remote DC sites.

Service level monitoring, elastic SLAs management

Network services are monitored and analysed according to specific SLAs that could vary in time to adapt to customer's needs. SLA related parameters are continuously monitored for SLA validation and countermeasures are taken automatically to prevent or react to SLA violations.

QoS provisioning

The DCN Control Plane implements mechanisms to provide Quality of Service according to the specific characteristics of the different services types, adopting techniques for bandwidth allocation r latency and resiliency policies to ensure a pre-determined service level.

Real Time Control

Real time control, detection and action are implemented to provide the capability to deal with performance issues or constraints.

Multi-tenant isolation

Each user of the system, together with their own virtual network infrastructures, will be isolated and unequivocally and uniquely recognized in the system in a secured and safe way.

Service Traffic Profile

Services and applications deployed in the DC will provide information about their expected traffic profile and connection requirements, for example maximum packet loss permitted, minimum guaranteed bandwidth, peak rate, jitter, latency, type of connectivity, recovery strategies, etc. The DCN Control Plane is able to translate these requirements into appropriate network configuration at the optical data plane.

VDC optical bandwidth control

In the Virtual Data Centre use case, the DCN Control Plane provides on-demand virtual optical infrastructures with a given set of capabilities and a certain level of programmability exposed through open APIs. This programmability allows the Virtual Infrastructure Operator (VIO) who operate the VDC to dynamically control the bandwidth allocation, within the boundaries of the virtual optical resources available in its own VDC.

Automated provisioning of intra-DC connectivity

The DCN Control Plane implements automated DCN configuration procedures to establish, modify and remove intra-DC connectivity services on demand. This feature can be used in combination with

the overlay network virtualization, to guarantee a proper underlying connectivity between the edges of the network, or in support of management actions at the DC level (e.g. fast migration of VMs or huge amount of data through dedicated connections).

Support for customizable network services

Customizable intra-DC network services are implemented to provide user level constrains like QoS parameters, bandwidth limitation, policing, shaping traffic, timing constrains, service resilience guarantees.

DCN resource usage optimization

The COSIGN DCN Control Plane provides intra-DC connectivity services compliant with a variety of operator-level constraints, including load-balancing strategies, energy efficiency, paths with minimum cost, etc.

Multi-layer operation of COSIGN data plane optical technologies

The DCN Control Plane is able to efficiently operate the heterogeneous optical technologies deployed in the physical layer. Each technology is controlled through dedicated plugins at the D-CPI and abstracted with a technology-independent information model to allow a unified control of the different devices. Strategies for network traffic adaptation and aggregation are applied to exploit the different capabilities and granularities offered at the COSIGN data plane.

DCN elastic intra-DC connectivity

The COSIGN DCN Control Plane is able to dynamically scale up and down the capacity of the established connections, in support of the elastic features of the cloud services.

Integration with external connectivity services (inter-DC)

The COSIGN Control Plane configures the DCN to efficiently support not only the intra-DC traffic, but also the traffic generated among computing resources located in different Data Centres. This involves also a suitable configurations of the external gateways at the DCN infrastructure. Possible interfaces for integrated management of intra-DC and inter-DC connectivity will be investigated.

Scalability

The scalability of the COSIGN Control Plane is a key requirement for the proper operation of the Data Centres with increasing dimensions and complexity. Therefore, the size (in terms of servers and optical devices to be managed) as well as the expected huge number of traffic flows among servers should not affect the properly working of the Control Plane operations.

3.2 Services

The COSIGN DCN Control Plane provides three main types of service, as detailed in the following subsections. These services are mostly transversal to the three use cases defined in deliverable D1.1 [1]. However, for each use case they may need to support more specific features in compliance with the requirements and business relationships defined for the associated use case.

3.2.1 Multi-Technology Optical Resource Virtualization

The optical resource virtualization is an internal service of the COSIGN DCN Control Plane, which constitute the basic building block to enable all the functions and services that needs to operate the COSIGN DCN physical infrastructure. The objective of this first level of virtualization is to provide a complete set of functions to manipulate and monitor the multi-technology optical resources through a more abstract interface. This approach allows the internal components of the SDN controller to deal with the multi-technology DCN in an efficient and uniform way, managing virtual optical resources characterized by a common set of parameters instead of the physical resource variety.

The relationship between physical and virtual resources may follow the 1:N model. A physical node or link is split in multiple virtual nodes or links, each of them with a disjoint subset of the original capabilities that characterized the corresponding physical resource. Another approach is to aggregate and abstract more physical devices in a single virtual resource.

However, it should be noted that single optical resources are managed only at the DC provider level, i.e. internally at the SDN controller or through management APIs, but they are not exposed as single, configurable entities to the DC provider's customers. This approach creates a sort of screen between the DCN infrastructure and the DC customers, preventing any data plane misconfigurations and guaranteeing the required level of isolation. Indeed, the final objective of the COSIGN control plane is to use the COSIGN optical data plane, enriched by multiple network technologies, to compose converged virtual optical network slices. These slices are the orchestrated entities that can be exposed to the different tenants, depending on the specific use case.

At the resource virtualization level, the different optical technologies of the COSIGN data plane are abstracted with their key features and attributes defined in the information model and exposed to the control plane through an abstraction layer. The key challenge is the definition of an information model (i.e. resource characteristics and allowed operations) powerful enough to exploit all the unique features offered by the optical data plane, while maintaining an acceptable and manageable level of complexity. The granularities of different technologies will be abstracted in order to compose a virtual slice with end-to-end granularity matching to accommodate the service requirements.

3.2.2 Provisioning and Management of Virtual Optical Infrastructures

The COSIGN DCN Control Plane is able to compose and provide on-demand virtual optical infrastructures (i.e. network slices) consisting of multiple virtual resources and meeting specific requirements in terms of QoS, topology constraints, type of traffic to be supported, service resiliency, etc. These network slices are isolated entities which share the same physical DCN infrastructure. They are usually part of a larger IaaS offer, since they are integrated at the cloud orchestrated level with computing and storage resources to provide a complete virtual environment to run cloud services and applications. This means that virtual slices need to be elastic entities that can be up- or down-scaled depending on the cloud service dynamicity. Thus the Control Plane needs to expose suitable APIs for on-demand modification or to specify thresholds and events for triggering the automated adaptation of the virtual infrastructure to pre-defined conditions.

The Control Plane is responsible of the whole lifecycle of a virtual network slice, from creation to termination, and implements all the functions required to operate, monitor and manage its resources, in compliance with the established SLAs. The operation of a virtual slice may be implemented as an internal feature of the DCN Control Plane. For example, in case of failures at the data plane, the Control Plane may react moving some virtual resources to other physical resources. Moreover, the allocation of some virtual resources may be dynamically modified to re-optimize the DCN utilization following changes in the DC service demands. In this case, the Control Plane's procedures are fully hidden from the final consumer of the network slice.

However, a virtual slice may also expose a certain level of programmability, delegating part of its operation to upper layer entities, e.g. control applications deployed by the slice's tenant. This is a typical requirement of the VDC use case, where the DC provider's customers may require to directly operate their rented infrastructure, e.g., to allocate some bandwidth for end-to-end connectivity between specific end-points or to deploy their own virtual network functions.

This scenario is shown in *Figure 7*, where the DC provider operates the physical DCN with the SDN controller, creating two virtual optical infrastructures (the green and the red one) which are provided to Tenant A and Tenant B respectively. These infrastructures are totally isolated and each tenant can operate and configure its own slice through the programmable APIs exposed by the SDN controller. The type of operational functions allowed on each slice is regulated through policies that reflect the SLAs established with each tenant. In the example, the two tenants deploy their own SDN applications (located at the ONF architecture application plane), which interact with the SDN controller through the A-CPI offered by the DC provider. At the SDN controller, some agents under the DC provider's control are deployed to execute the application commands. These agents are the entities in charge of limiting the operational scope of each tenant's application instances to the resources belonging to the virtual slice associated to that tenant.

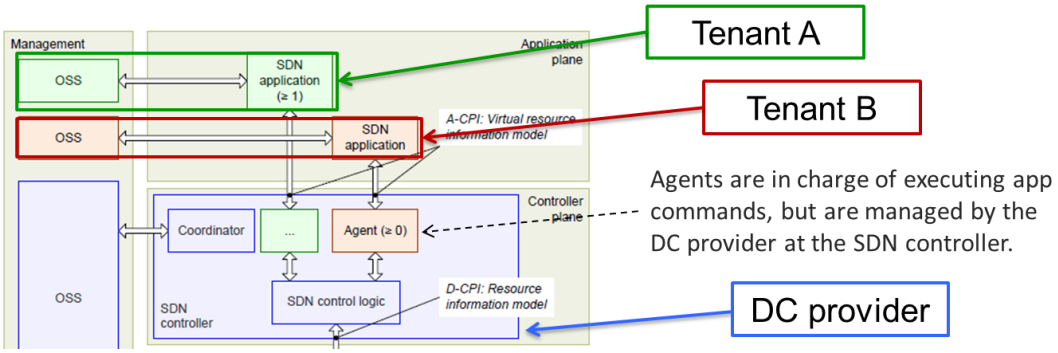


Figure 7 – Virtual slices controlled through SDN applications deployed by tenants

It should be noted that the DC operator may create virtual slices over the physical DCN not only for IaaS services offered to customers, but also for internal purposes related to the efficient sharing of the whole DC infrastructure or the specialized management of different technology domains. In this case, the level of programmability exposed for each slice will be very similar to the programmability of the physical infrastructure, even if expressed with abstracted and unified terms.

When the DC operator splits the DCN in multiple slices, he/she may decide to deploy other SDN controllers to operate each slice. On the other hand, each tenant may decide to operate the rented virtual slice with its own SDN controller, instead of using directly SDN applications. This leads to a hierarchy of SDN controllers (see *Figure 8*) that may be deployed in flexible scenarios with different technology domains or even different trust domains. The control plane functions provided by each SDN controller generates a hierarchy of resource abstractions, providing progressively higher views of the DCN through different information models. In the vertical dimension, each layer of SDN controller hides technological and topological details, while in the horizontal dimension it may reduce the view scope to subsets of DCN resources (e.g., a virtual network slice).

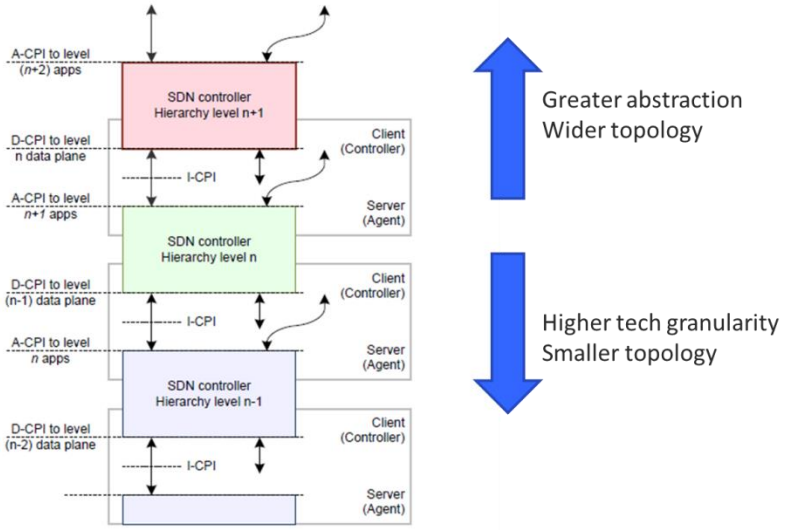


Figure 8 – Hierarchy of SDN controllers

3.2.3 Provisioning and Management of Intra-DC Optical Connectivity

The highest level of service provided by the COSIGN DCN control plane is the provisioning and management of on-demand or scheduled end-to-end connectivity over the optical intra-DC network. Optical connections can be created directly over the physical DCN or on top of virtual slices.

The first option is typically invoked to support management actions on the DC infrastructure, for example to establish dedicated paths for the migration of huge contents or VM images between different servers or storages located in the same DC or in remote sites for backup and restoration purposes.

Optical connections may be also created over tenant's virtual slices in the VDC use case. In this case, suitable APIs must be provided by the SDN controller to allow the tenants to setup, monitor and destroy end-to-end optical paths between their VM over the rented virtual infrastructures. This option allows to make use of internal functions exposed by the DCN Control Plane which may provide configurable parameters towards the consumers. For example, a tenant may request connections which minimize a given metric or which avoid some virtual nodes. However, it should be noted that nothing prevents the customer to apply their own computation algorithms and configure each virtual node accordingly. This depends essentially on the level of programmability which is allowed for each virtual infrastructures and which is regulated through per-tenant policies.

Finally, the provisioning of optical connections is a fundamental service to enable the use case focused on the on-demand deployment of application-level overlay virtual networks. In this scenario, the optical connectivity can be established directly on the physical DCN or over virtual slices created internally at the DC operator's level. As already discussed in previous sections, the main challenge is to bring a cross-layer awareness between overlay virtualization at the orchestration level and optical connections' provisioning in the underlying infrastructure at the DCN Control Plane. This means that optical connections should be re-adapted to changes in the overlay networks, while overlay networks should make a dynamic use of optical connectivity established on-demands, without relying on static and pre-configured connectivity.

From the analysis of the three main services provided by the COSIGN DCN Control Plane, it is evident that each service provides a further layer of abstraction (i.e. virtual resources, virtual infrastructures, virtual optical connections) which in some cases is even associated to different trust domains. For example, the virtualization of optical resources and the provisioning of virtual optical infrastructures are performed within the scope of the DC provider. However, the provisioning of optical connectivity is in the scope of the DC provider only when operating on the DCN physical infrastructure or over virtual slices still under full control of the DC provider itself. This is the case of network connectivity established for DC management tasks or in support of overlay network virtualization use case. However, in the VDC use case, the optical connectivity is handled over a virtual slice belonging to a specific tenant and it is mostly managed in the trust domain of the tenant itself.

Across these different virtualization layers, the DCN Control Plane needs also to provide a set of common, vertical features to be applied to the different virtual entities manipulated in each layer. For example, monitoring, resiliency, re-optimization of resource allocation in the underlying layer are common features that should be guaranteed for single virtual optical resources, composed virtual infrastructure or optical connections. This aspect can become complex and crucial in scenarios with multiple SDN controllers, where each controller has a partial view and control of the overall resources or operates at distinct layers. In fact, in this case mechanisms for distributed coordination, delegation and/or synchronization must be implemented.

3.3 Deployment Models

Three main deployment models can be considered when designing an SDN-based control plane:

- A **fully centralized model**, with a logically single SDN controller with the whole view of the entire DCN
- A **hierarchical model**, with a first layer of "child" SDN controllers responsible for disjoined portion of the DCN and a "parent" SDN controller which handle a more abstracted view of the whole DCN, obtained through the cooperation with the "child" controllers. This model could be adopted to handle very different technology domains through specialized "child" SDN controllers and requires only a north-south interaction between each child controller and its parent controller. Of course, this interaction can be recursive and involve several layers in the SDN controller hierarchy.
- A **distributed east-west model**, with a set of peer SDN controllers, each of them responsible for a portion of the DCN and cooperating together with horizontal communications to achieve the whole control and management of the entire DCN.

In general, the third model is very complex and particularly suitable to large scenarios characterized by a variety of trust domains where each network domain is controlled through proprietary protocols and a very limited amount of information can be exchanged outside the domain boundaries. The inter-controller communication can be regulated using protocols like BGP (Border Gateway Protocol) [RFC4271] or PCEP (Path Computation Element communication Protocol) [RFC5440]. However, this model is scarcely applicable to the intra-DC scenario, where a single DC provider is responsible for the whole DCN.

The hierarchical model may be of interest due to the heterogeneous nature of the COSIGN DCN. However, the complexity and the time delays required by the inter-controller synchronization and communication (also in this case the PCEP is a reasonable candidate, with the extensions for the stateful and active mode [PCESTAT], [PCEINIT]) can be considered as strong limitation in highly dynamic scenarios like DCN environment.

For this reasons, the COSIGN DCN Control Plane solution is based on the centralized deployment model, with a single SDN controller that maintains the overall view and control of the overall DCN. The specific characteristics of the different technologies available at the data plane are handled through dedicated plugins and abstracted with a powerful information model that allows for the application of unified functions and procedures in the SDN controller core. This approach avoids the need to introduce specialized controllers to deals with each optical device type deployed in the COSIGN DCN.

It should be noted that the SDN controller is a single entity from a logical point of view. However, from the architectural point of view, the controller can be deployed as a cluster of controllers for high availability or scaling reasons. This option is supported by some SDN controller platforms available today (including OpenDaylight and ONOS, see Section 2.4) and relies on mechanisms for distributed databases, avoiding the complexity of inter-controller protocols.

3.4 Cross-Layer Workflows of COSIGN Control Plane

In this section, the interaction between the components is described in terms of high-level functions within the workflows, identifying the entities outside WP3 that will be involved.

RAS
Service reliability and availability capability will interact with both physical and network management plane for network service resilience.

Seamless Network Resources Provisioning
The provisioning capability will interact with both the physical layer and DC network service virtualization through the provisioning requests.

Advanced Network Services Provisioning
The provisioning capability will interact with DC network service virtualization through the provisioning requests.

Service level monitoring, Elastic SLAs management
This capability will interact with both physical and management layers for service management and monitoring.

QoS provisioning
This capability will interact with the service provisioning facility in the DC network service virtualization.

Real Time Control
This capability will interact with both network management and physical layer for network services management.

Multi-tenant isolation
This capability will cooperate with both the management and physical layers for service management.

Service Traffic Profile (From UC2)
This capability interacts with the DC network service virtualization layer.

VDC optical BW control
This is a novel capability that will interact with both, the physical and DC network service virtualization layers.

Automated provisioning of intra-DC connectivity
This capability interacts mainly with the DC network service virtualization and management layer.

Support for customizable network services
This capability interacts with both, the DC network virtualization layer and physicallayer for provisioning and management of network services.

Multi-layer operation of COSIGN data plane optical technologies
This capability will interact with both network management and physical layers for network services management integration.

DCN Resource usage optimization
This capability will interact with both network management and physical layers for network services operation.

DCN Elastic intra-DC connectivity
Interactions with network management and physical planes will be required for providing elastic provisioning of connectivity.

Integration with external connectivity services (inter-DC)
Both physical and management planes will be required to interact with this capability for network services integration.

Scalability
Scalability will have to interact with DC network services virtualization and physical layers.

3.5 Applicability Scenarios for Data Centres

Section 3.3 describes three possible deployment models and motivates the choice of the centralized model for the COSIGN DCN Control Plane, which is deployed at the DC operator domain with a single SDN controller. However, the adoption of this principle at the DC operator level, does not prevent its customers to deploy their own SDN controllers for operating their own virtual infrastructures. This option has been briefly mentioned for the VDC use case in Section 3.2, and it is further explained in the pictures below.

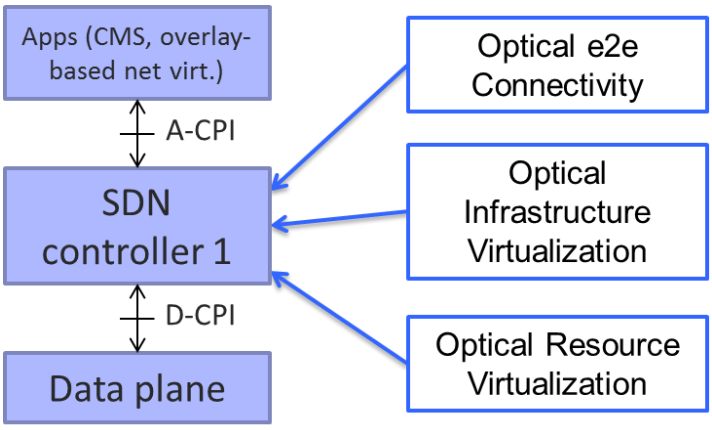


Figure 9 – Deployment for the use case on provisioning of overlay virtual networks

Figure 9 shows a typical deployment for the use case dedicated to the provisioning of overlay virtual networks. In this use case, the customer is interested only in a suitable environment to run its applications, independently on the specific topology of the virtual network infrastructures or the connectivity that allows the communication between the VMs. For this reason, the only SDN controller involved in this use case is the one which operates the DCN (SDN controller 1 in the picture). This controller is managed by the DC operator and provides the whole set of COSIGN Control Plane services: optical resource virtualization, optical infrastructure virtualization and

provisioning of end-to-end connectivity, while upper layer SDN applications implement the overlay-based network virtualization which expose its services to the customer.

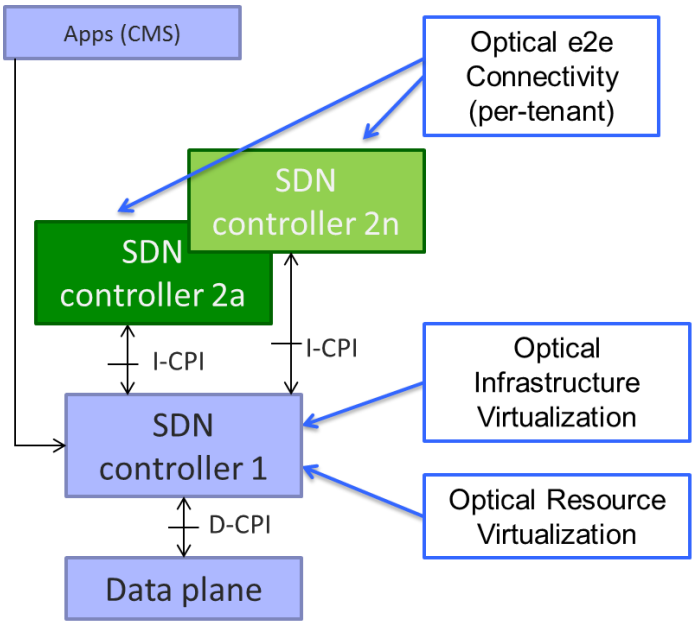


Figure 10 – Deployment for the VDC use case

On the other hand, in the VDC use case the customers are interested in a programmable virtual infrastructure which may be operated and configured using a dedicated control plane, external from the one which runs over the physical DCN. A possible solution to control this virtual infrastructure is to use again an SDN controller, resulting in a hierarchy with two levels of SDN controllers (see *Figure 10*).

In this scenario, we can identify multiple trust domains. The first one is associated to the DC operator and corresponds to the SDN controller at the bottom (*SDN controller 1* in the picture) which operates the DCN physical infrastructure. It implements the optical resource virtualization and offer the service for the on-demand provisioning of programmable virtual optical infrastructures. On top of that, *N* trust domains are associated to the tenants renting the virtual network slices. Each tenant deploys its own SDN controller (represented in green in the picture) to manage its network slice, for example to establish a customized optical network connectivity between their VMs. These controllers perform indirect operations over a subset of the DCN resources, which are mediated through the virtual functions implemented in the DC operator's controller.

Another aspect to be considered is the possible co-existence between the COSIGN DCN, managed through its SDN-based Control Plane, and other portion of the DCN based on legacy devices with their own control and management tools. This is particularly relevant in the short and medium term, as an intermediate step towards the migration to full COSIGN DCN solutions (see *Figure 11*), where the compatibility with existing technologies is fundamental for the acceptance of this novel approach.

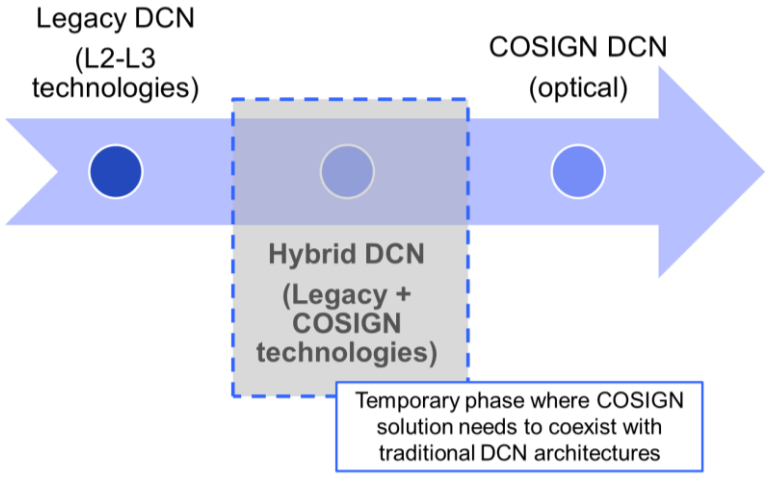


Figure 11 – Steps to migrate towards COSIGN optical DCN

Several potential scenarios could be foreseen, despite the fact to the date there are no SDN standard east-west mechanisms to connect with legacy non-COSIGN network domains.

Orchestrated alternative

A first alternative would consist of the coordination of different control planes (SDN COSIGN likely and legacy ones) from the orchestrator layer. This control plane orchestration would coordinate in a centralized way the different control plane frameworks while providing network connectivity services which involve devices from different nature. In this scenario, control plane NBI (Northbound Interfaces) should facilitate to the orchestration layer the required information to handle all control plane platforms present in the DC. This orchestrated alternative is currently being proposed in the context of GEANT3+ project [GN3p]. In this particular case, the orchestrated approach aims to make use of Network to Service Interface (NSI) agents and protocol to ensure the E2E connectivity traversing different domain types (that in the case of COSIGN would be legacy and COSIGN likely domains).

Distribute alternative

A second alternative may consist of different types of control plane being coordinated in a distributed way making use of East-West interfaces. This approach does not entail a centralized management of the different control plane systems, but the control plane signalling information would be forwarded among the involved SDN controllers and legacy control systems. To the date, there are no standard East-West interfaces defined, but some attempts and proposals have been done recently. This is the case of the Juniper Contrail SDN solution [Contrail] which proposes a East-West interface between SDN controllers to interchange standard BGP information. Nevertheless, the solution does not include legacy control systems. Also, the SDNi IETF draft [SDNi] proposes SDN interconnectivity and message exchange among multiple SDN controller instances within the scope of a single administrative domain making use of ALTO protocol [RFC7285]. To the date, the draft has not yet been categorized as standard.

Thus, it will be matter of COSIGN to monitor and propose potential requirements and solutions that could solve the interconnection of COSIGN and legacy network control systems in DCs either in an centralized manner (by means of COSIGN orchestrator) or in a distribute way, leaning on East-West interfaces.

3.6 Control Plane Requirements Matching

This section analyses the COSIGN Control Plane functional requirements defined in WP1, in deliverables D1.1 and D1.3 ([1] and [2]), and extracts the CP functions to meet these requirements. These functions are also mapped on CP functional components and interfaces (northbound or southbound) with the external layers in COSIGN architecture (see Table 2).

Table 2 – Mapping of CP requirements in functional components and interfaces

Requirement	CP functions	CP functional components	CP interface
Automated provisioning of intra-DC connectivity	Computation and provisioning of optical paths over the DCN	Optical provisioning manager Path computation	A-CPI for connectivity requests
Support for customizable network services	Computation and provisioning of optical paths with user-constraints (QoS, service resiliency, scheduling)	Optical provisioning manager with scheduling and recovery support Path computation for scheduled or disjoint paths Topology Inventory with calendar extensions Monitoring and fault manager	A-CPI for connectivity requests with user-constraints specification D-CPI for notification of data plane failures
Multiple automated connectivity paradigms	Computation and provisioning of point-to-point, point-to-multipoint or anycast connectivity	Optical provisioning manager Path computation with support for P2MP and anycast paths	A-CPI for P2MP and anycast requests
Multi-layer DCN operation	Operation of multiple optical technologies with different granularities	Agents and south-bound plugins for each data plane technology Service Abstraction Layer for optical resource abstraction Optical resource virtualization service Optical provisioning manager with multi-layer capabilities, including re-adaptation of the virtual server layer Path computation with multi-layer capabilities	D-CPI for data plane configuration
DCN resource usage optimization	DCN operation with operator-level constraints	Virtual Infrastructure manager Optical provisioning manager with re-planning features Path computation with Global Concurrent Optimization algorithms Policy manager	A-CPI for configuration of operator policies D-CPI for collection of statistics about network performance
Elastic intra-DC connectivity	Dynamic modification of established connectivity or Virtual	Virtual Infrastructure manager with modification	A-CPI for modification requests

	Infrastructures on-demand or based on automatic elasticity rules.	capabilities Optical provisioning manager with modification capabilities Monitoring and fault manager	A-CPI for specification of elasticity rules D-CPI for collection of statistics or failure notifications
Network monitoring	Service for monitoring and statistics	Monitoring and fault manager	D-CPI for collection of statistics or failure notifications A-CPI to expose monitoring information
Programmable APIs	Programmable APIs to request provisioning of network connectivity, even over virtual slices	Policy manager AAA for authorization	A-CPI
Network service monitoring and accounting	Monitoring and accounting for provisioning of virtual slices and connectivity	Monitoring and fault manager AAA for accounting	A-CPI to expose monitoring and accounting information D-CPI for collection of statistics or failure notifications
Scheduled network connectivity	Computation and provisioning of scheduled connectivity or virtual slices.	Optical provisioning manager with scheduling support Path computation for scheduled paths Topology manager with calendar extensions	A-CPI for requests with timing constraints
Service resiliency	Automated procedures for network service recovery	Optical provisioning manager with recovery support Path computation for disjoint paths Monitoring and fault manager	A-CPI for failure notifications D-CPI for notification of data plane failures
Multi-tenant isolation	Provisioning of isolated multi-tenant virtual slices	Virtual Infrastructure manager	--
SLA enforcement and verification	Provisioning of SLA-based services (connectivity and virtual slices), detection of SLA violations and automated reaction	SLA manager Monitoring	A-CPI for SLA specification D-CPI for collection of statistics or failure notifications

Interoperability with cloud platforms	Integration with cloud management platforms for NaaS provisioning	Virtual Infrastructure manager	A-CPI for the interaction with the cloud orchestrator
Integration with external connectivity services	Support of traffic for inter-DC communications	Optical provisioning manager	A-CPI in case on multi-domain interactions coordinated at the orchestrator level
DCN reconfiguration for optimization strategies	Automated re-planning of connectivity service or virtual slices	Virtual Infrastructure manager with re-planning features Optical provisioning manager with re-planning features Path computation with Global Concurrent Optimization (GCO) algorithms Monitoring	A-CPI for configuration of operator policies D-CPI for collection of statistics about network performance

4 Functional Components of the SDN Control Framework

The control plane architecture in COSIGN is designed around the Software Defined Networking (SDN) principles of a logically centralized controller. SDN is defined as a control framework that supports programmability of network functions and protocols by decoupling the data plane and the control plane, disrupting their vertical integration as it is for most of the legacy network equipment currently available in the market. On the other end, the separation of control and data planes makes SDN a preferable solution for an integrated control framework supporting heterogeneous technologies deployed in different layers (e.g. optical layer, Ethernet, and IP layer) and within several network segments, from access to transport, including data centres.

An SDN based control plane is an attractive solution for control and management of data centre networks since it opens the opportunity to support a wide set of features and functionalities. First, SDN can abstract the heterogeneous technologies employed in the data plane and represent and model them in a unified way. Therefore, it can be used to build a unified control plane for hierarchies of network technologies, such as optical technologies, layer2 and layer 3 within data centres. To implement and enable this abstraction and unification of data plane capabilities, proper vendor and technology agnostic protocols and interfaces are needed for the communications between the SDN controller and the data plane devices. To this purpose, OpenFlow (OF) is a suitable open standard protocol for the realization of SDN, since it is based on flow switching with the capability to execute software and user-defined flow based routing, control and management in an SDN controller located outside the data path. In addition, there are lot of standard, mature and widely deployed network protocols and interfaces, such as PCEP, NETCONF, OVSDB that can be easily deployed and integrated by means of an SDN approach in a data centre environment, as a way to interact and co-exist with further legacy tools and control entities supporting such protocols. Moreover, an SDN control plane deploys a generic controller, equipped with open, flexible and extendable interfaces which can be used to host any proprietary application, aiming at network programmability at the application level.

Furthermore, today's data centres are becoming increasingly multi-tenant, adopting a service model where each tenant is provided with its own virtual data centre infrastructure. In such an environment network virtualization is a key feature and when is combined with virtualization at the servers side, it allows data centre operators to create and offer to its customers multiple co-existing isolated virtual infrastructures. Here, SDN principles allow data centre operators to provision and control logical partitions of the network and create multiple network slices (virtual networks) for each tenant.

The COSIGN SDN Control Plane is presented in *Figure 12* in its functional split. It is composed around three main layers of functionalities that have been defined starting from the bottom side:

- *Abstraction layer*, to create and provide a unified and protocol-independent set of commands and notifications, expressed through a common DCN information model
- *Infrastructure control layer*, with a set of basic network functions for virtualization of optical resources and connectivity programming. This layer implements the functions to offer the three main services of the COSIGN Control Plane: optical resource virtualization, provisioning of programmable virtual optical infrastructures and provisioning of optical connectivity
- *Network service virtualization layer*, which implements the highest level of network virtualization following the overlay-based model with traffic encapsulated at the end points and tunnelled across the DCN.

The main role of the COSIGN Control Plane is to manage the entire DCN and expose its capabilities as a set of primitives to the upper layers (e.g., applications for overlay-based network virtualization or cloud orchestration platforms), exploiting the SDN benefits of network programmability and a global and unified view of the underlying network resources. This SDN functional split is particularly suitable to the COSIGN environment, where the dynamic and highly variable profiles of the data centre traffic require quick reconfigurations at the data plane, over network topologies interconnecting

an ever increasing number of servers. Moreover, the network programmability offered by SDN is crucial to achieve an efficient integration between network and cloud environments, where the data centre connectivity needs to be tightly coupled with the evolution of the VM lifecycles with high degrees of automation.

The COSIGN Control Plane architecture is aligned with the SDN architecture 1.0 defined by ONF [ONF-arch], that is an enhanced version of the preliminary ONF specification in [ONF-arch-0], mostly in terms of SDN principles, architectural components, control functions and interactions. However, the COSIGN SDN Control Plane evolves and modifies the ONF solution to apply the SDN principles and control functions to data centre environments, since ONF mostly targets transport networks. On the other end, COSIGN fully adopts the ONF SDN principle of abstracting network resources towards external applications via a northbound interface (i.e. the A-CPI) to enable deep programmability of DCN and virtual slices deployed on top of that. Moreover, COSIGN and ONF solutions both consider virtualization as a crucial service in SDN: the controller is indeed designed in both specifications to operate on abstracted and virtualized resources (enabled by multiple levels of abstraction and virtualisation) which ultimately encompass underlying heterogeneous physical network resources. The peculiarity of COSIN is that these physical network resources are optical technologies deployed inside the Data Centre.

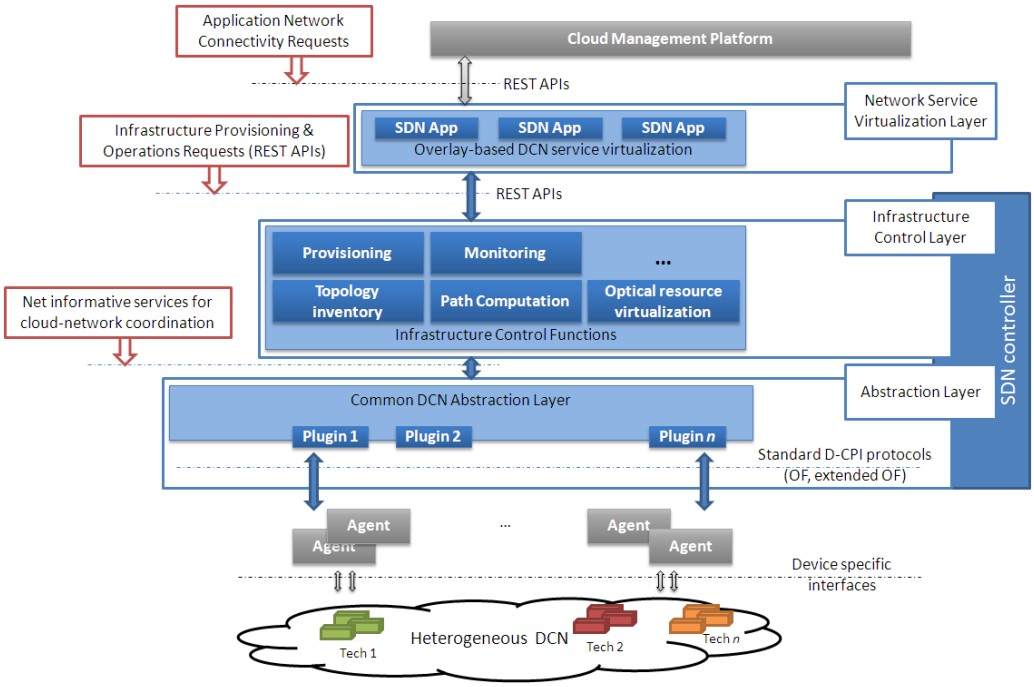


Figure 12 – COSIGN Control Plane architecture and functional components

The following sub sections provide a description of the functional components of the COSIGN SDN Control Plane, starting from the bottom side of *Figure 12*, i.e., the one closer to the data plane optical devices.

4.1 Components

4.1.1 Technology Agents

The communication between the SDN control plane and the optical devices in the data centre fabric is enabled by the technology agents, which act as bridging entities deployed on top of each data plane device to hide their specific technology and management/control interface details. In brief, a technology agent operates on top of each optical data plane device (with a 1:1 relationship) and takes the responsibility to translate the control messages coming from the SDN controller into the actual configuration commands for the given device. Moreover, it maintains a node-wide uniform resource description model for each device and provides capabilities and availabilities up to the controller through the proper control messages.

In COSIGN, the control protocol used at the SDN southbound interface will be OpenFlow. OpenFlow is considered the first SDN standard and defines the open communications protocol that enables a controller to interact with the forwarding layers in the SDN architecture. OpenFlow allows direct access to and manipulation of the forwarding plane of network devices such as switches and routers, either physical or virtual (hypervisor-based, like Open vSwitch). Therefore, to have SDN and OpenFlow enabled data plane optical devices, an OpenFlow agent for each device is required.

In COSIGN, these OpenFlow agents will provide the following control functions:

- Technology Specific Mapping and Translation: this part enables the communication between the OpenFlow agent and data plane devices. Different protocols and technologies (NE's management interface like SNMP, TL1, Vendor API, etc.) may be used to implement this interface, according to the given interface exposed by each optical device
- Uniform Resource Model: this resource model exposes a technology independent abstraction of the different optical technologies with the aim of simplifying the controller functions by hiding the hardware differences
- Extended OpenFlow API: The API is responsible for establishing and maintaining the communication tunnel between COSIGN SDN controller and the OpenFlow agent. Specific OpenFlow protocol extensions will be defined at the southbound interface to support the heterogeneous optical technologies developed in COSIGN.

4.1.2 SDN Drivers

The SDN drivers are the lowest level components in the COSIGN SDN controller. They are specific plugins needed at the southbound reference point of the controller to handle the different protocols adopted at the D-CPI. In principle, the SDN architecture is generic enough to support and deploy different types of drivers at the southbound interface, namely to let the controller interact at the same time with devices and network equipment exposing different protocols and interfaces, such as OpenFlow, NETCONF, OVSDB, PCEP.

In COSIGN, the SDN drivers will mostly implement and support the extended version of the OpenFlow protocol to communicate with the OpenFlow agents deployed on top of each optical device. In addition, a dedicated SDN driver for the communication with Open vSwitch instances running in the data centre servers might be needed in COSIGN, implementing the OVSDB protocol. For each type of COSIGN SDN driver, a 1:N relationship with the technology agents and data plane devices in general will occur.

4.1.3 Abstraction Layer

The abstraction layer provides a common and uniform access to the different optical devices deployed in the COSIGN hybrid data centre fabric, by means of a uniform DCN information model. Therefore, it creates and exposes a unified and protocol-independent set of commands and primitives to manage and configure the variety of optical technologies deployed in the data plane. This layer is implemented leveraging on the protocol-specific SDN drivers described above, each providing a per-technology and device-wide abstraction. On top of this first level of abstraction, a common DCN information model is built by this abstraction layer to fully expose to other SDN internal components (i.e. the core network functions) capabilities of heterogeneous optical technologies and devices to describe their features, actions supported and peer information (to build the topology).

In particular, the COSIGN SDN Control Plane addresses the heterogeneity at the hybrid optical data plane following a two-fold approach. On the one end, it introduces extensions at the D-CPI protocols to exploit the specific capabilities and granularities of the different optical resources. On the other end, powerful abstraction models simplify the management of these resources within the SDN controller, using a unified information model for the whole DCN. This common DCN abstraction layer is a key component of the COSING SDN controller, and the main enabler of the SDN principles for deep programmability of network functionalities from the upper layers.

4.1.4 Core Network Functions

The core network functions are those components of the COSIGN SDN controller which provide the basic internal functions to manage DCN resources and offer the COSIGN Control Plane services discussed in Section 3.2. The whole set of these network functions composes the infrastructure control layer of the COSIGN Control Plane architecture: it includes elementary blocks like topology inventory, path computation, reliable provisioning and monitoring. In order to support the enhanced features required by COSIGN use cases (see Section 3.6 for a mapping between CP functions and components), these blocks are properly extended or new ones are introduced as internal modules of the COSIGN SDN controller. They cooperate together to establish connection paths and compose isolated virtual infrastructures coexisting over the same DCN physical topology, following the network virtualization model based on the on DCN fabric programming. Typically, these functions are implemented as internal modules of the SDN controller and export REST API towards upper layer applications.

The following table lists the COSIGN SDN controller core network functions and a brief description is provided for each one.

Table 3 – COSIGN SDN controller: functional components

Core network function	Description
Optical provisioning manager	This core function provides the network connectivity intelligence in the COSIGN SDN controller, by handling the actual configuration of the optical devices as exposed by the common DCN abstraction layer, with QoS guarantees. It exposes internal SDN primitives to be used by other services which need to provision and configure the optical devices. In the case of OpenFlow, it provides the flow programming APIs to be used by other core network functions (e.g. the optical resource virtualization) to properly provision the hybrid optical data plane. It sits on top of the common DCN abstraction layer and it uses the abstracted view of the DCN resources. Also, it exposes dedicated external primitives and REST APIs at the A-CPI for QoS enabled network connectivity in support of DC management procedures and services, such as VM migration during management operation, or overlay-based DCN service virtualization. The provisioning manager of existing SDN controllers needs to be extended to support specific COSIGN requirements, in particular related to the multi-layer nature of the COSIGN data plane. Moreover, it needs to implement additional features, like provisioning of scheduled connections, re-planning for DCN optimization, recovery mechanisms triggered by data plane failures.
Topology Inventory	The Topology Inventory maintains an updated view of the underlying DCN optical resources, combining and composing the different entities (i.e. devices) in a topology format. It is directly interfaced with the common DCN abstraction layer for topology discovery and learning purposes. Also, it provides topology information to other internal SDN core network functions, such as the Provisioning and the Path Computation. It also exposes topology views at the A-CPI reference point for usage by the SDN network applications, possibly with different degrees of details according to the given consumer.
Monitoring and fault manager	The Monitoring function is responsible for handling all the notifications about status and performances of the DCN optical devices (e.g. alarms, failures, number of dropped packets, congestion status, etc.). It directly

	interacts with the abstraction layer to gather all the monitoring information through the D-CPI interface. It also exports, through the A-CPI, these information to those network applications running on top of the SDN controller that need to react upon failure events (or degradation of performances) for dynamic and flexible control of connectivity inside the DCN.
Optical Resource Virtualization Manager and Virtual Infrastructure Manager	The Optical Resource Virtualization Manager and the Virtual Infrastructure Manager are key functions in the COSIGN SDN Control Plane architecture. They implement the two associated Control Plane services (see Sections 3.2.1 and 3.2.2) and their combined action enables the multi-tenant virtual networks provisioning. In particular, the Virtual Infrastructure Manager exposes APIs used by external SDN applications to create their own virtual topologies on demand and with QoS guarantees. The main aim is to provision isolated virtual network slices on top of the optical DC fabric, possibly belonging to multiple customers and users of the DC infrastructure. Dedicated routing policies and network configurations are enforced on top of the abstracted DCN resources (as exposed by the abstraction layer) for traffic isolation purposes. The Virtual Infrastructure Manager exposes through the A-CPI a logical plane where virtual switches and virtual links are created: external applications can design and deploy any desired virtual network without dealing and caring of the different optical technologies constraints and physical topologies. It is an internal task of the SDN controller to take care of the actual mapping of the virtual slices and topologies into the underlying optical DCN resources.
Path Computation	The Path Computation is conceived to provide some basic and simple routing functions inside the COSIGN SDN controller. It basically implements computation algorithms on top of the topology views exposed by the Topology Inventory in support of the provisioning of QoS enabled connectivity within the optical DCN. Dedicated computation functions for virtual networks and slices creation are also provided, in support of the Virtual Infrastructure Manager. The aim in this case is to exploit the heterogeneous capabilities of the optical technologies in the COSIGN DCN and match for each virtual slice the requirements of given application or customer using it. More complex and elaborated path computation functions, implementing specific algorithms or objective functions (e.g. for disjoint paths, P2MP paths or GCO), may be deployed as an extension of the SDN controller Path Computation component or externally as an SDN application which can be invoked by the SDN controller itself. In this case, the usage of standard protocols, such as the PCEP, may be needed to interact with these path computation entities, e.g. by means of dedicated SDN drivers.

4.1.5 Network Applications

The COSIGN control plane architecture described here follows the SDN principle of keeping the SDN controller itself as a generalized container of basic network functions which aim to provide deep programmability of optical resources in the DCN for SDN applications and services running above the controller. The core network functions presented above provides all the tools and the primitives, i.e. through the A-CPI, to manage, control and operate the COSIGN optical DCN in a programmable and flexible way.

A set of SDN applications are implemented over the COSIGN SDN controller, both at the COSIGN Control Plane (WP3) and at the cloud orchestrator level (WP4). These applications cover more complex control procedures, often involving also IT resources (e.g. joint provisioning of virtual network and IT infrastructures at the cloud orchestrator), or management functions that are needed to match the global COSIGN requirements. Example of these applications include the overlay-based network and service virtualization, the orchestration of cloud services (WP4) and management functions like policy configuration, accounting and billing, authentication and authorization or SLA management (see Table 4).

Table 4 – SDN applications in COSIGN architecture

SDN application	Description
Overlay-based service virtualization	This is a key SDN application in the COSIGN architecture and it implements the highest level of network virtualization following the overlay-based model with traffic encapsulated at the end points and tunneled across the DCN. In particular, it does not need to have a direct view of the DCN topology and capabilities, but it can just rely on the abstract connection services provided by the infrastructure control layer, i.e. by the SDN controller A-CPI APIs. This approach simplifies the service virtualization approach, since it operates just at the network edges. On the other hand, the cooperation with the infrastructure control functions introduces a certain degree of awareness and mutual adaptation between the two layers, with the possibility to overcome the limitations of the pure overlay-based approach. For example, bandwidth constraints between two end points can be guaranteed through the proper configuration of optical connection paths at the infrastructure control layer, upon specific request of the service virtualization layer. Connection recovery can be also managed in a transparent and policy-driven way at the infrastructure control layer inside the SDN controller and, in case of unrecoverable failures, suitable notifications can be delivered towards the upper layers. Finally, a partial cross-layer visibility allows to re-program the whole underlying virtual paths taking into account the dynamicity of the overlay traffic, while informative services disclosing a suitable level of monitoring information about the DCN status and structure can be considered to take more efficient and network-aware decisions about the whole cloud service deployment.
Cloud Service Orchestration (WP4)	The Cloud Service Orchestration service, implemented in a cloud management platform like OpenStack, is the key entity for the integration between network and IT resources. The orchestrator manages the whole set of resources available in a Data Centre and relies on the DCN Control Plane for the configuration of the DCN and the provisioning of virtual network slices. The interaction between the cloud orchestrator and the DCN Control Plane is based on the REST API exposed at the A-CPI of the SDN controller and, optionally, can make use of REST APIs developed by SDN applications running on top of the controller itself. An example is the case of the Overlay-based Service Virtualization, which is implemented as an SDN controller and can be invoked by the cloud orchestrator to establish application-driven virtual overlay networks. However, the interaction between cloud orchestrator and DCN Control Plane in COSIGN is not only used to request actions of the DCN, but

	also to collect information about the DCN status or the existing network services already established on the DCN. This allows to bring a certain level of network awareness at the orchestrator level, that may be used to take more efficient decisions about IT resource allocations.
Policy manager	The Policy manager is a management application which allows the DCN administrator to configure and activate the policies that will be used within the SDN controller. These policies may regulate various aspects of the services offered by the SDN Control Plane, e.g. the level of disclosure that can exposed towards external entities regarding confidential information like topology, network performance, failures, etc. or the matching between abstract QoS parameters or application requirements and actual network configurations.
AAA	The AAA component offers functions for: - Authentication and authorization of SDN applications and tenants which operates on the DCN resources, directly or through virtual slices - Accounting for the usage of the DCN resources and the provisioning of virtual slices and optical connectivity. Accounting needs to support different charging and billing models, e.g. flat or pay-per-use. Accounting makes use of the services provided by the monitoring and statistics modules of the SDN controller.
SLA manager	The SLA manager is the entity in charge of configuring, enforcing and validating the SLAs established with the different tenants. It offers a management interface on the northbound to allow the DC administrator to configure the SLA and retrieve reports about SLA verifications. At runtime, it interacts with the provisioning manager for SLA enforcement and with the monitoring service for SLA assessment. It implements functions for detection of SLA violations and may also include algorithms for early SLA violation prediction or mechanisms to actively react in case of SLA failures.

4.1.6 Initial Mapping on Software Architecture

Considering the analysis reported in Section 2.4, OpenDaylight offers an SDN controller platform which is suitable to be extended with the new modules and functions of the COSIGN controller. The COSIGN **abstraction layer** (Section 4.1.3) can be mapped on the OpenDaylight Model-Driven Service Abstraction Layer (MD-SAL), which enables the abstraction of the services implemented by different types of OpenDaylight plugins. These services are described using the YANG language [RFC6020], providing an abstract and implementation-independent specification of its interface. In particular, following the YANG models, a service can be characterized with data structures describing the resources that will be manipulated from the services itself, remote procedure calls which define the actions that a service consumer can invoke on these resources and notifications that the service will trigger to inform its listeners about asynchronous events. OpenDaylight's internal framework is able to automatically translate the YANG models in both internal java APIs and REST APIs, based on the RESTCONF protocol [RESTCONF]. This approach allows for easily extending the controller with new internal modules (called plugins) or external applications which act as consumers of each OpenDaylight service.

The internal implementation of the services exposed by the MD-SAL is managed by the OpenDaylight plugins. Two main types of plugins are available in OpenDaylight: the southbound plugins, which are responsible of the interaction with the data plane, and the application plugins, which implement the high-level logic and makes use of the primitives offered by the southbound plugins to translate their directives into actual network configurations. The difference between these plugins is just conceptual,

while their software design is exactly the same and follows the principle of the OpenDaylight MD-SAL.

The COSIGN **SDN drivers** (Section 4.1.2) can be implemented as OpenDaylight southbound plugins, possibly extending the already existing OpenFlow plugin ([ODL-OFP], [ODL-OFL]) with the optical extensions required to operate the COSIGN DCN optical data plane. On the other hand, the COSIGN **core network functions** (Section 4.1.4) can be implemented as OpenDaylight application plugins. Also in this case, a large number of functions is already present (e.g., a topology service, a statistics service, etc.) and can be easily extended to meet the new COSIGN requirements by extending the associated YANG models with the YANG augment mechanisms (see [RFC6020] for details). Only where needed, new OpenDaylight plugins will be defined.

Finally, the COSIGN **network applications** (Section 4.1.5) can be developed either as external entities that will make use of the OpenDaylight RESTCONF APIs (e.g., the service orchestrator in WP4 will rely on OpenStack, where the Neutron service interacts with OpenDaylight through REST APIs) or application plugins. Also in this case, COSIGN software design will try to re-use as much as possible the components already available in the OpenDaylight framework. Example of applications that be used as baseline for COSIGN extensions are the OpenDaylight AAA application for authorization and accounting [ODL-AAA], the applications for network virtualizations ([ODL-DOVE] or [ODL-VTN], or the project for application-centric policy models [ODL-POL].

The detailed selection of OpenDaylight plugins to be used as starting point for COSIGN development will be performed in the next period, as part of the WP3 software design activities.

4.2 Information Models

Different information models exist within the COSIGN DCN Control Plane. They reside in different layers and reflect different levels of resources. Each of them defines its own main elements and parameters specified together with the actions that can be performed from external entities over the resources.

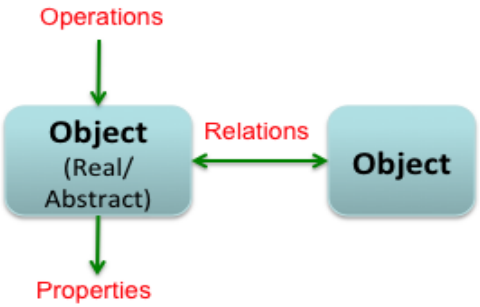


Figure 13 – General diagram describing the main entities in COSIGN information models

The main information models adopted in COSIGN SDN controller are associated to the high-level services offered by the COSIGN Control Plane: multi-technology optical resource virtualization, provisioning and management of virtual optical slices and provisioning and management of optical connections. These information models, expressed in the format of YANG data models, actually define the "contract APIs" of these services, defining the structures that can be manipulated from the service consumers and the commands and notifications that can be exchanged between service consumer and provider.

Beyond these main "external" information models, that can be easily associated to the interfaces exposed towards the upper layer entities (e.g. the cloud orchestrator), other internal information models may need to be defined to regulate the interaction between the internal components of the SDN controller. They can be considered as models which provides an intermediate level of abstraction (e.g. hiding protocol or some level of technology details) to facilitate the operation of some services on the COSIGN DCN.

Finally, the lowest level of information model at the COSIGN Control Plane is used to describe the physical resources and it is used at the D-CPI interface through the adoption of specific protocols, like

OpenFlow. This information model will include the highest level of technology details about the variety devices deployed at the DCN and will include:

- Data structures for features, characteristics, capabilities and constraints of DCN physical devices
- Methods for synchronous operations, triggered from the controller, to configure the optical devices (e.g. to make or destroy a cross-connection)
- Methods or notifications for resource synchronization, to allow the controller to retrieve information about the status and the capabilities of the devices under its control
- Notifications for failure events related to devices' resources
- Methods or notifications to collect performance and metric measurements from the data plane.

The specification of the information model for the COSIGN physical resources depends on WP2 outcomes and the COSIGN D-CPI will be defined in cooperation with this work-package.

The following sub-sections will briefly elaborate on the features and principles of the main information models in COSIGN Control Plane. The specification of the information model used at the external interfaces will be provided in deliverable D3.2, which will be focused on the SDN controller interfaces, while the internal information models will be defined through the YANG models of the associated OpenDaylight plugins during the software design phase.

4.2.1 Virtual Resources

The virtual resources information model is used to describe the optical resource virtualization service. This model includes the description of all the virtual resources that can be created inside a Data Centre, for example virtual nodes (network nodes (V Net Node) and IT nodes/VMs (VM)) and virtual links (V Link). The SDN based Control Plane will focus on the network related resources.

The optical resource virtualization service APIs can be expressed through the common CRUD actions to be invoked over a virtual resource, for example "create a virtual node with a given number of virtual ports", "delete the virtual link with a given ID", etc. Asynchronous events can be also modelled as modifications in the status of an existing resource (e.g. a node failure can change the operational state of that node resources). Therefore they can be retrieved through proper subscriptions associated to a given type of resources or attributes in the database. Moreover, further commands can define other specific operations on some resources, for example "connect port A with port B in node N".

Some key attributes for describing each element have been identified in *Figure 14*:

- Virtual DC topology
  - Topology (interconnections of the links and nodes)
  - Number of nodes (virtual network and IT)
  - Capacity (each node and each link)
- V Net Node
  - Number of ports
  - Bitrate per ports
  - Other advanced features and functionalities which abstract the technology capability
- VM
  - CPU requirement
  - Memory requirement
  - Capacity requirement
- V Link
  - Bandwidth
  - Latency
  - Other advanced features and functionalities which abstract the technology capability.

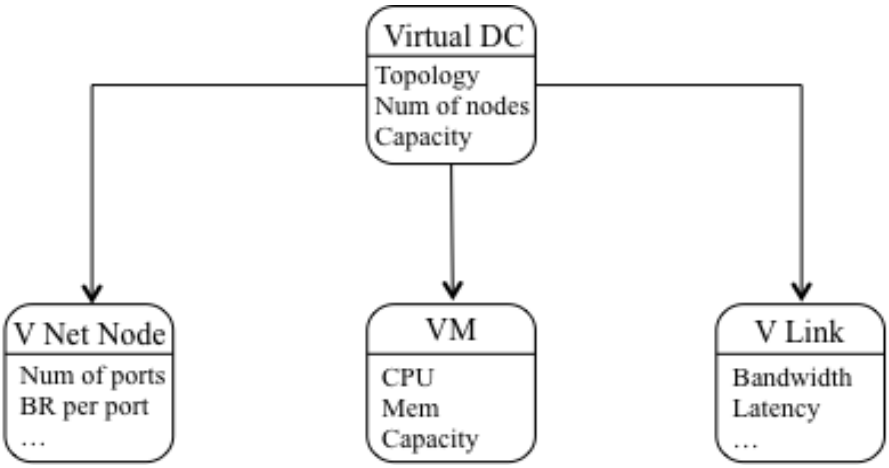


Figure 14 – Information model for virtual DC resources

Depending on the physical technology which is at the basis of the virtual resource, the advanced parameters which characterize virtual network nodes and links may be slightly different, however they will provide a certain level of abstraction and try to use common and unified concepts. This approach allows the other services which will use the basic virtual optical resources (e.g. to compose a virtual slice) to exploit the optical characteristics of the DCN, while using common mechanisms, procedures and algorithms.

4.2.2 DCN Infrastructure

The DCN infrastructure information model is an uniform model exposed by the abstraction layer for both internal SDN controller modules and external applications and it is used to provide information about the DCN capabilities.

The main objective of the abstraction layer is to provide an homogenous view of the DCN infrastructure to facilitate the management and control of the heterogeneous devices and technologies that compose the data plane. To achieve an information model that is generic enough to cope with all the technologies involved in the DCN, we need to simplify the elements and parameters to be used. In this context, at the most simplified view, the DCN can be seen as a set of nodes and links, which are deployed in a specific topology. Hence, in a first phase of the information model, the topology element defines the nodes and links composing the DCN. Then, the nodes provide an abstraction of the different switching devices that can be found in the data plane (e.g. ToRs, optical switches, etc.). Finally, the links connect the different nodes into the given topology. The number of nodes and links characterize the topology element. The nodes may be further divided into different classes according to the role they play in the DCN but, in a first stage, all of them need to keep common attributes such as the number and type of ports. For the links, bandwidth and occupancy attributes are needed.

The DCN information model is typically used in read-write mode from some of the internal modules of the SDN controller, since they may need to operate and configure somehow the DCN itself. However, the REST APIs towards external entities may reduce the access to read-only operations. This is typically the case of the cloud management platform, which may need to collect information related to the DCN capabilities for taking internal decisions about cloud service scheduling or VM locations. However, the cloud platform itself is not allowed to operate directly on the DCN (i.e., with writing access on DCN resources), but needs the mediation of other DCN Control Plane services. They will be responsible to configure the DCN while providing higher level APIs to create virtual network slices for IaaS provisioning or end-to-end intra-DC connections to enable DC management procedures.

4.2.2.1 DCN Services

The DCN should allow the following services:

- To expose a full resource view, including virtual network topology
- To directly expose the low-level information model

- To expose abstracted views of its resources. This should be done with a specialization of the same information model that exposes full detail.

Furthermore, the DCN should allow the following functions for client's applications:

- To allow an application to set and query any attribute or state within the scope of its control
- To allow an application to control traffic forwarding: to select traffic according to a set of criteria, to modify or adapt the traffic if necessary, to forward it to a given set of egress points. Selection criteria may be as simple as an ingress port or as complex as a vector that includes match fields up to and including layer 7, potentially spanning multiple packets of a flow. The type of applicable criteria depends on the node technology, i.e. an OVS instance running in a server, a ToR, an optical switch
- To allow an application to propose a traffic-forwarding construct that requires the use of new or existing resources according to specified figures of merit, and to receive one or more offers from the controller. The client may propose and accept in a single request, or may review the offers and accept zero or one of them
- To allow an application to invoke and control standardized functions such as STP, MAC learning, ICMP, BFD/802.1ag, 802.1X, etc.
- To allow an application to subscribe to notifications of faults, attribute value changes, state changes and threshold crossing alerts (TCAs)
- To allow an application to configure performance monitoring (PM) collection points, with thresholds, and to retrieve current and recent results
- To allow an application to invoke and control traffic-processing functions through exchanges of opaque data blocks.

4.2.3 Information Models for COSIGN Control Plane Services at the A-CPI

The COSIGN control plane offers two main services at its northbound interface: the provisioning and management of Virtual Infrastructure and the provisioning and management of end-to-end connectivity. This section briefly presents the type of information in the models describing the APIs for these services and discusses a possible deployment for the A-CPI.

4.2.3.1 Virtual Infrastructures

The Virtual Infrastructure information model describes the service for the provisioning and management of virtual network slices, which is implemented by the *Virtual Infrastructure Manager* component of the SDN controller (see Table 3). The information model specifies the entities composing a virtual network which can be handled as part of a cloud service offer. Depending on the request type, the level of detail in the request can specify topology constraints or even abstract optical capabilities (e.g. a WDM link with two wavelengths). In this case, the virtual infrastructure information model will be very similar to the DCN information model, but dealing with virtual resources rather than with physical ones. Moreover, the request may even specify virtual network functions to be traversed by some traffic flows described through classifiers.

Otherwise, the request can describe a virtual network infrastructure using the higher level concepts which are currently in use in cloud services. For example, the Neutron component in OpenStack defines virtual resources like networks, subnets, ports, routers, interfaces, security groups and rule, metering labels. COSIGN offers a service to enable CRUD operations on this type of resources through the SDN application for *Overlay-based network service virtualization* (see Table 4). The REST APIs offered by this application at the SDN controller's A-CPI are used for the interaction with the Neutron component of the cloud platform for the on-demand provisioning of overlay virtual networks.

4.2.3.2 End-to-End Intra-DC Connectivity

The information model for the intra-DC connectivity describes the service to establish network connections between the edge nodes of a DC. The model will be built around the concept of "end-to-end connection", as it would be requested from upper layer applications that are fully agnostic about the specific nature of the COSIGN optical DCN. For example, the following parameters may be specified for a given connections:

- Connection type (point-to-point, multi-point, anycast)
- End-points: source and destination, which may corresponds to multiple endpoints in case of P2MP or anycast connections
- QoS parameters, like bandwidth or delay
- Guarantees for service reliability (e.g., protection)
- Monitoring information to be received
- Start-time and duration for scheduled connections.

This "external" model does not include any notion about the usage of the optical technologies available at the DCN, the multi-layer nature of the resulting connections or the data path established between source and destination. This type of information is handled internally at the provider of this service, i.e. the *Optical Provisioning Manager* components listed in Table 3, and will be included in the YANG models of the internal SDN controller services consumed by the Optical Provisioning Manager itself to compute and configure the end-to-end connection.

4.2.3.3 A-CPI Deployment

From an architectural perspective, the entities involved at the A-CPI comprise entities that belong to the Infrastructure Control Layer and external entities that interact with the control functions (see *Figure 15*).

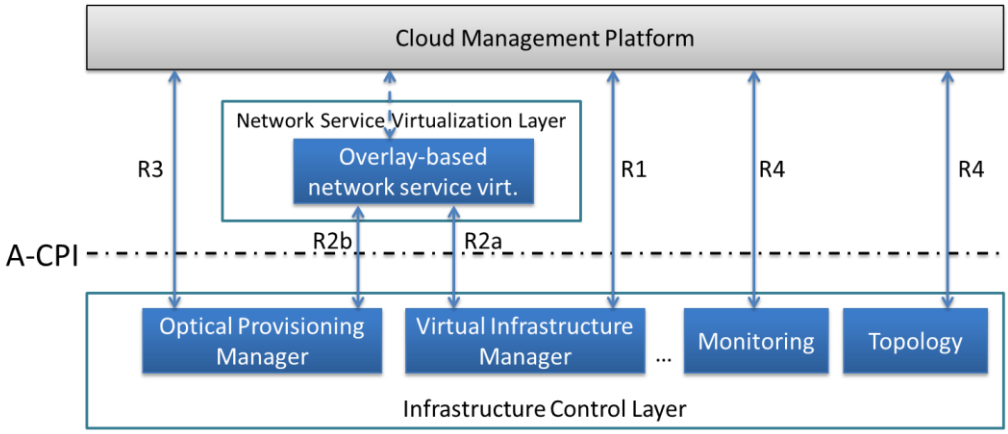


Figure 15 – A-CPI deployment

In particular, there are two external entities that interact with the control functions: the cloud management platform (i.e. the *orchestrator* developed in WP4) and the *DC Network (DCN) service virtualization layer*. The deployment models discussed herein assume there are only one orchestrator and one DCN service virtualization module.

The main SDN controller entities involved at the A-CPI, which offer services to uppers layer entities, are:

1. The *Virtual Infrastructure Manager*, for provisioning of virtual optical infrastructures

   The orchestrator provides IaaS (e.g. Virtual Data Centres) to customers. A customer can have several VDCs (1:N relation). A VDC relies on the existence of a virtual network infrastructure. The Virtual Infrastructure Manager provides such virtual infrastructures to the orchestrator (R1 in *Figure 15*). The relationship between a VDC and a virtual network infrastructure is 1:1.

The DCN service virtualization module offers services for provisioning of virtual service networks (virtual networks that are customized according to customer preferences). To do this, the DCN service virtualization module also requests virtual network infrastructures (R2a in *Figure 15*). There is a 1:1 interaction model between a virtual service network and a virtual network infrastructure. The virtual service network maps as an overlay over the virtual network infrastructure.

2. The *Optical Provisioning Manager*, for provisioning of end-to-end optical connectivity

The services offered by this module are consumed by the orchestrator and the DCN service virtualization application. For the orchestrator, there is a 1:N relationship between a tenant network (or a VDC) and the end-to-end optical paths, meaning that several optical paths can be mapped onto a tenant network (R3 in *Figure 15*). Similarly, there is a 1:N relation between a virtual service network and the end-to-end optical paths, meaning that the connectivity existing in the overlay (the virtual service network) is supported by multiple end-to-end optical connections in the underlay DCN (R2b in *Figure 15*).

3. *Monitoring* and *Topology* modules, which offer informative services (network-related information)

The services offered by these modules may be consumed by the orchestrator in order to make optimal decisions for coordination of IT and network resources. The orchestrator can request information regarding different aspects of the network. However, it is not necessary to assign resources to different entities, in this process. The module for informative services may create data objects that contain the required information, which are passed to the orchestrator (R4 in *Figure 15*). In this context there is a 1:N relationship between a tenant network and the data objects with network information, meaning that there can be several data objects conveying information about a tenant network to the orchestrator. On the other hand, the information encapsulated in one data object refers to one tenant network.

4.3 Internal Workflows

This section defines the internal workflows that regulate the interaction between the internal functional components of the COSIGN Control Plane (see Section 4.1) for the two services exposed towards external layers, i.e., the provisioning and management of virtual optical infrastructures (see Section 3.2.2) and the provisioning and management of intra-DC optical connectivity (See section 3.2.3).

4.3.1 Workflows for Provisioning and Management of Virtual Optical Infrastructures

The workflow shown in *Figure 16* shows the procedures to create and provision a programmable virtual optical infrastructure (i.e. a virtual network slice), when requested from the upper layer cloud service orchestrator.

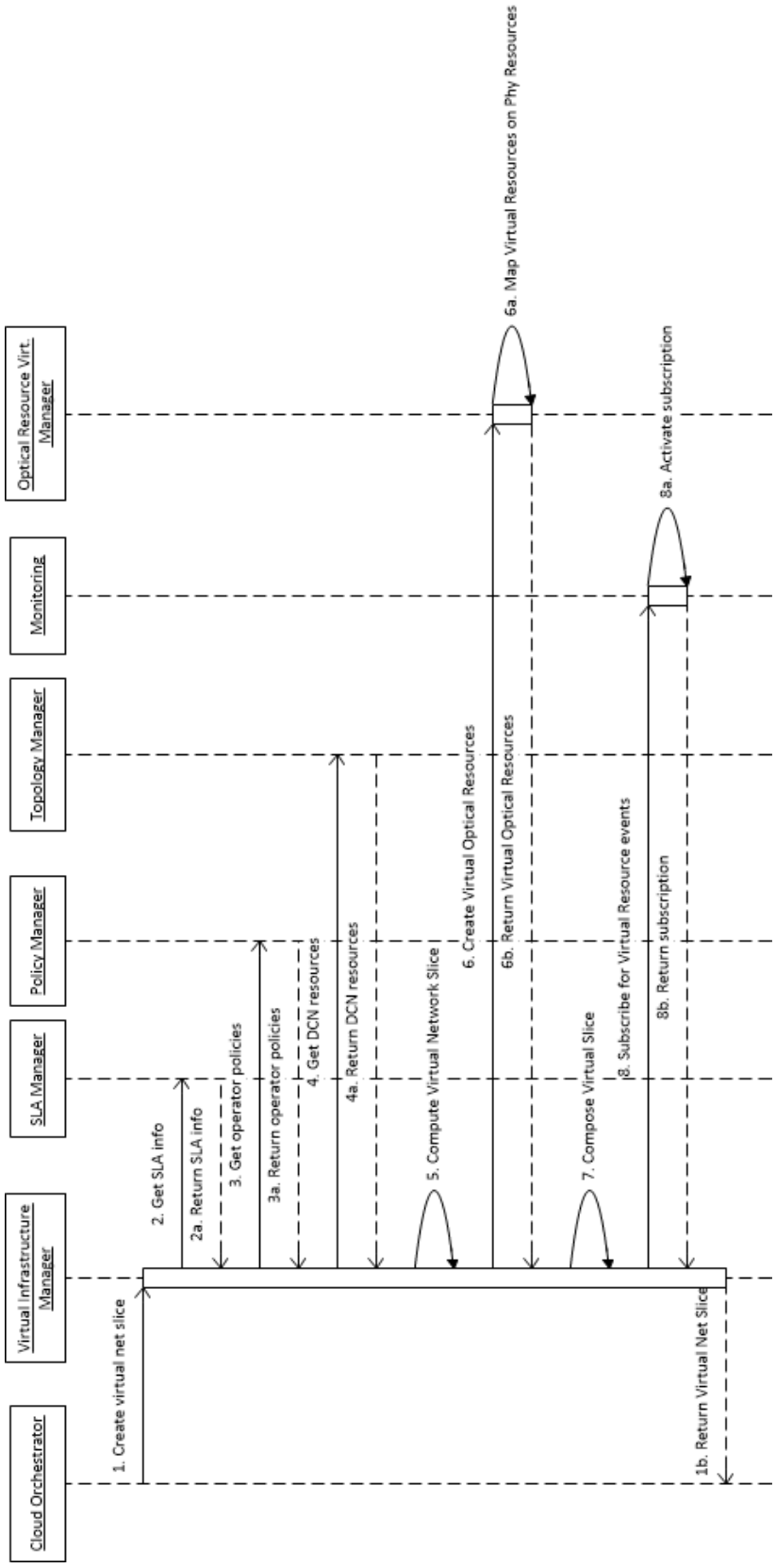


Figure 16 – Provisioning of a Virtual Optical Infrastructure

The request is processed by the *Virtual Infrastructure Manager*, which retrieves information from the *SLA Manager* and the *Policy Manager* (steps 2 and 3) to complete the specification of the virtual slice (e.g. QoS parameters, DCN usage constraints, etc.) depending on the business agreement with the customer who originated the request and the operator's policies. The virtual infrastructure is

elaborated taking into account the capabilities of the DCN (steps 4 and 5). Following this design, the SDN controller creates all the virtual resources required to compose the whole virtual infrastructure (steps 6-7). Since the *Virtual Infrastructure Manager* will be responsible for the entire lifecycle of the slice, it creates a subscription with the *Monitoring* component (step 8) to receive all the asynchronous events associated to the virtual infrastructure resources.

This approach allows the *Virtual Infrastructure Manager* to react in case of failures at the DCN data plane which have an impact on the virtual slice, following the workflow depicted in *Figure 17*. A data plane failure (e.g. a port failure) is notified from the affected device to the SDN controller using the specific protocol used at the D-CPI, for example OpenFlow (step 1). The notification is handled by the *SDN driver* in charge of operating the failed device and forwarded to all the components which have registered to receive asynchronous notification about data plane events (step 2), in our case the *Topology Manager* and the *Optical Resource Virtualization Manager*. The former uses this information to update its view of the DCN status (step 3), while the latter is interested in the failures which affect the virtual resources built upon the given device (step 4) and generates the corresponding alerts for the *Monitoring* service (step 5). As consequence of the subscription performed in step 8 of the previous workflow (*Figure 16*), the *Virtual Infrastructure Manager* is informed about the failure which involves one of its virtual slices (steps 6 and 7). In order to take a decision about how to react, it interacts again with the *SLA Manager* and the *Policy Manager* (steps 8 and 9). Since the virtual slice needs to be recovered, it computes a new slice following the same procedures of the provisioning case but avoiding the failed nodes (steps 11-13) and removing the old virtual resources (step 12) following a make-before-break approach. Finally, the *cloud orchestrator* is informed about the modifications in the virtual slice (step 14).

Depending on the use case (for example in the VDC use case), the tenant may desire to maintain a certain level of programmability over the rented virtual infrastructure. This programmability is enabled through some APIs offered by the Virtual Infrastructure Manager, which is responsible to regulate the actions triggered by the tenant through its own applications and to map them over the corresponding virtual resources, guaranteeing the isolation between the different slices.

An example of workflow for virtual slice programming is shown in *Figure 18*. Here we suppose that the tenant has already requested and obtained a virtual slice, with some virtual nodes and links characterized by optical capabilities. The tenant is interested in creating customized optical paths over this infrastructure, computed using algorithms running in its own applications. The same application interacts with the *Virtual Infrastructure Manager* to request the creation of a cross-connection (XC) between two ports of a given virtual node X, belonging to the tenant's network slice (step 1). The *Virtual Infrastructure Manager* verifies the authorization for the requested operation (step 2) and resolve the local virtual resource managed at the SDN controller level (step 3). The cross-connection on the virtual node is mediated by the *Optical Resource Virtualization Manager* (steps 4-5), which handles the mapping between virtual and physical resources and triggers a command to configure the data plane node (step 6) on the *SDN driver*.

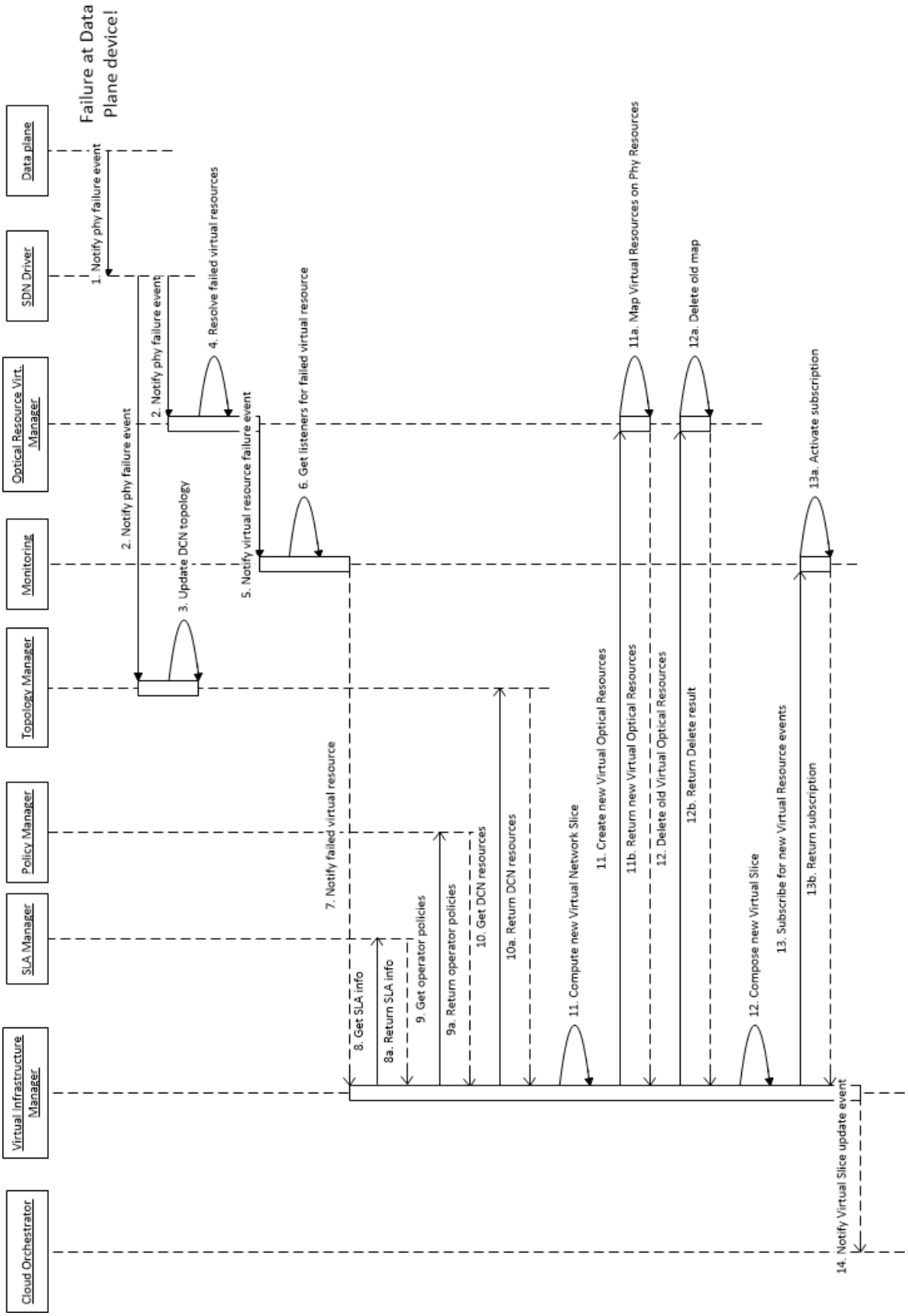


Figure 17 – Restoration of a Virtual Optical Infrastructure

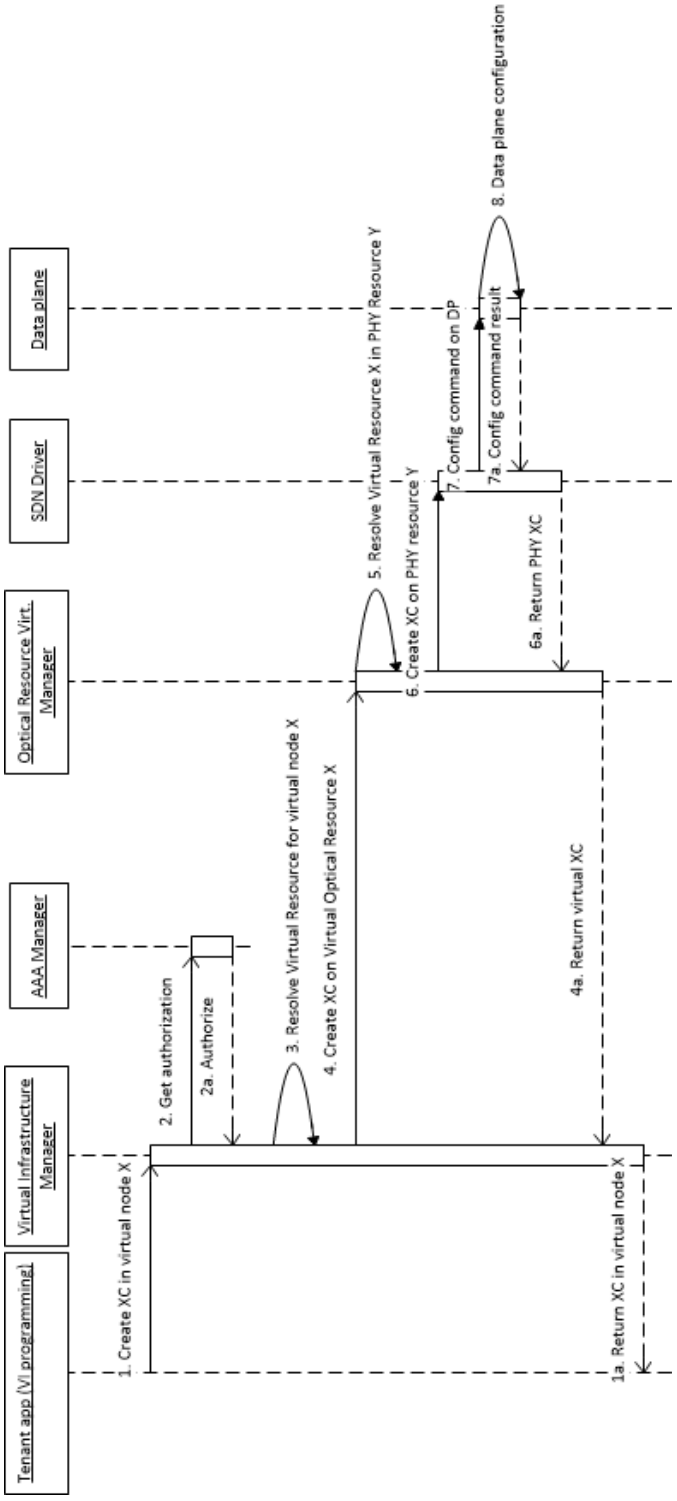


Figure 18 – Programming of a Virtual Optical Infrastructure

4.3.2 Workflows for Provisioning and Management of Intra-DC Optical Connectivity

The on-demand provisioning of intra-DC connections can be requested by the cloud orchestrator to dynamically create optical path between the nodes of the Data Centre dedicated to management traffic.

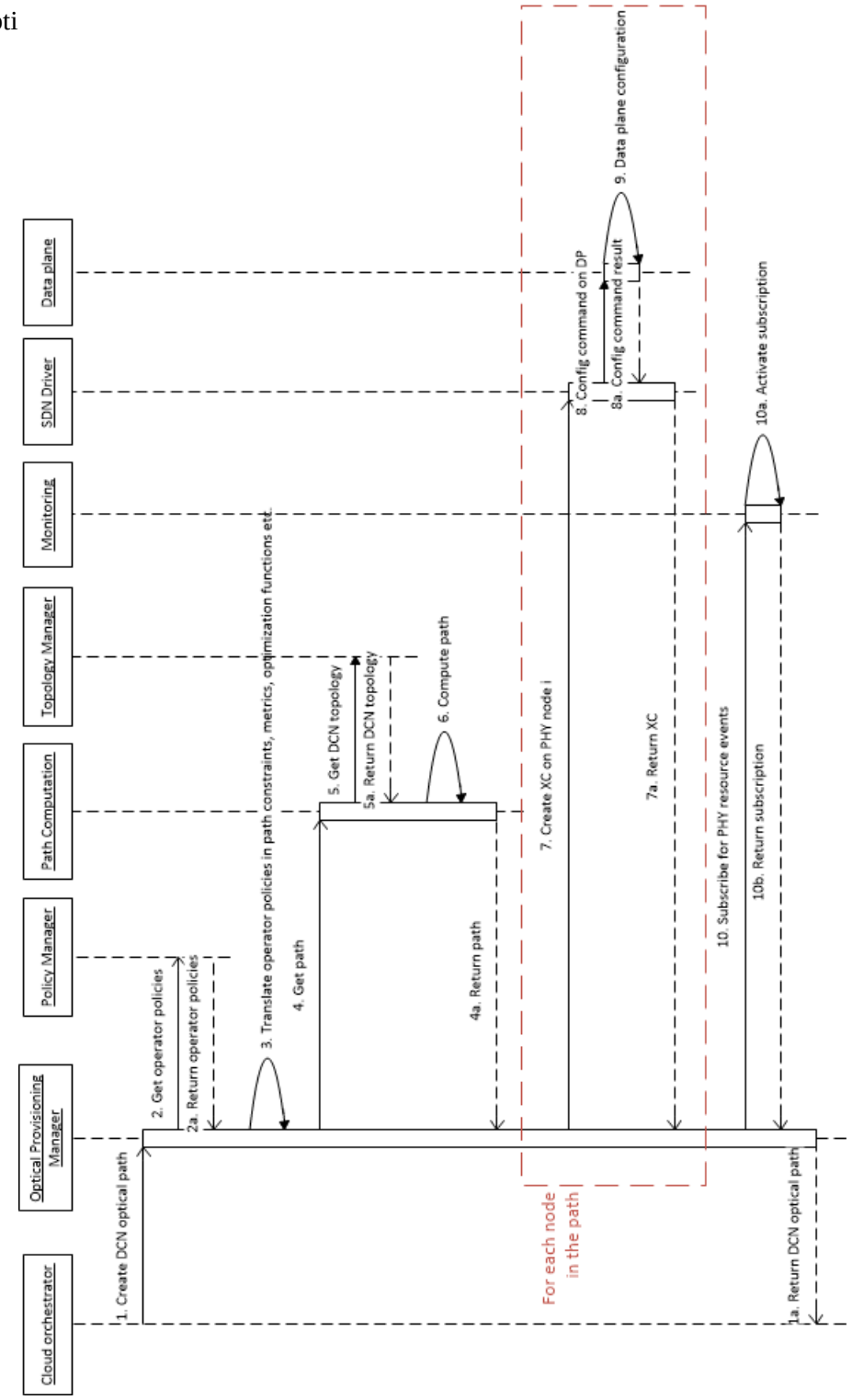


Figure 19 – Provisioning of intra-DC optical connectivity

Figure 19 shows the workflow to create an optical path over the DCN. The request is processed by the *Optical Provisioning Manager* component (step 1), which uses the operator's policies (step 2) to define the path characteristics in terms of constraints, metrics, optimization functions etc. (step 3). The elaboration of the optical path is performed through the algorithms of the *Path Computation* component (step 4 and 6), which runs over a network graph derived from the DCN topology view maintained by the *Topology Manager* (step 5). The Optical Provisioning Manager is responsible to coordinate the configuration of the DCN resources along the whole path, following the centralized concept of the SDN controller. Therefore, for each hop specified in the returned path, it creates a cross-connection on the associated physical node making use of the methods exposed by the SDN driver (step 7), which in turn configure the data plane (steps 8 and 9). When the whole path is

established, the Optical Provisioning Manager registers itself with the Monitoring service, in order to be notified in case of failures at the devices along the route (step 10).

5 Conclusions

This deliverable constitutes the first output of COSIGN WP3, providing the functional architecture of the COSIGN DCN Control Plane and a complete analysis of available open-source SDN controllers, used to select the reference software platform for the development of the COSIGN controller.

These outcomes will be taken as input for the following WP3 activities in the second year of the project, which will be focused on two main aspects:

- The definition of the COSIGN Control Plane external interfaces with the data plane (WP2) and the cloud orchestrator (WP4). These interfaces will be documented in deliverable *D3.2 "SDN framework north-bound and south-bound interfaces specification*"
- The software design of the COSIGN Control Plane, which will be based on the re-use, adaptation and extension of some components already available in the OpenDaylight controller, the reference software platform for COSIGN controller prototype.

6 References

[AWS-EC2] Amazon EC2 web page: http://aws.amazon.com/ec2/

[AWS-S3] Amazon S3 web page: http://aws.amazon.com/s3/

[CloudStack] CloudStack web page: http://cloudstack.apache.org/

[Contrail] OpenContrail web page: http://www.opencontrail.org/

[GN3p] GÉANT 3 project web page: http://geant3.archive.geant.net/

[NFV] "Network Functions Virtualization. An Introduction, Benefits, Enablers, Challenges & Call for Action", SDN and OpenFlow World Congress, Darmstadt, Germany, October 2012

[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-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-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

[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

[OpenIRIS] OpenIRIS web page: http://openiris.etri.re.kr/

[OpenNebula] OpenNebula web page: http://opennebula.org/

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

[PCESTAT] E. Crabbe, I. Minei, J. Medved, R. Varga, "PCEP Extensions for Stateful PCE", IETF draft, work in progress, October 2014

[PCEINIT] E. Crabbe, I. Minei, S. Sivabalan, R. Varga, "PCEP Extensions for PCE-initiated LSP Setup in a Stateful PCE model", IETF draft, work in progress, October 2014

[QFabric] Juniper QFabric web page:

http://www.juniper.net/us/en/products-services/switching/qfx-series/

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

[RFC4271] Y. Rekhter, T. Li, S. Hares, “A Border Gateway Protocol 4 (BGP-4)”, IETF RFC 4271, January 2006

[RFC5440] J.P. Vasseur, J.L. Le Roux, “Path Computation Element (PCE) Communication Protocol (PCEP)”, IETF RFC 5440, March 2009

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

[RFC7285] R. Alimi, R. Penno, Y. Yang, S. Kiesel, S. Previdi, W. Roome, S. Shalunov, R. Woundy, “Application-Layer Traffic Optimization (ALTO) protocol”, IETF RFC 7285, September 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

[UCS] Cisco Unified Computing System web page

http://www.cisco.com/c/en/us/products/servers-unified-computing/index.html

Appendix: Software Architecture

Following the comments from the first year technical review, this appendix has been added to describe the software modules that will be developed in the prototype of the COSIGN control layer. The objective is to highlight the components that will be implemented in the OpenDaylight framework as new OpenDaylight plugins developed from scratch or as extended versions of already existing OpenDaylight plugins. In this latest case, this section will provide details about the modifications and enhancements that will be needed to meet COSIGN requirements. Moreover, some components related to the composition and delivery of virtual optical slices will be implemented as OpenVirteX plugins and will require the extension of the OpenVirteX platform in support of optical resources, as documented in the next tables.

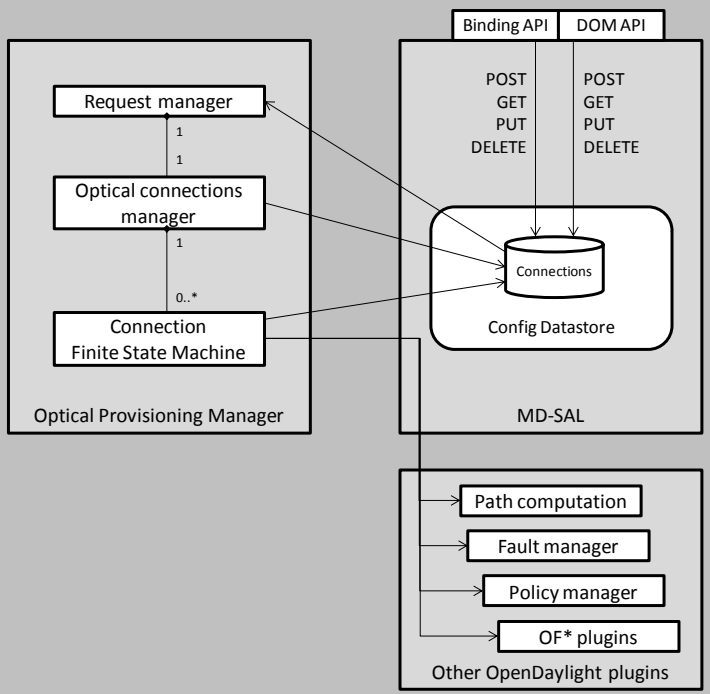
The outcomes documented in this section are the result of the software design activities carried out in T3.1, T3.2 and T3.3 from M13 to M15. These activities have taken as input the functional architecture documented in the previous sections of this document and, for each functional component defined in Section 4.1, have defined a set of software modules which will implement the associated control plane functions (functions derived from the requirements, as detailed in Table 2). The list of the resulting software modules is summarized in Table 5, which also specifies the prototype release(s) where each module will be included (i.e., preliminary release in *D3.3 "SDN controller for DC optical network virtualization and provisioning software prototypes: Preliminary release"* at M24 and/or final release in *D3.4 "SDN controller for DC optical network virtualization and provisioning software prototypes: Final release"* at M30). The following tables describe in details each software module, defining the software architecture which marks the milestone MS14 (Intra-DC control plane high-level architecture), planned at M15.

Table 5 – COSIGN control layer: software modules

Functional component	Software module (s)	Type of development	Release
Optical provisioning manager	Optical provisioning manager (Table 6)	OpenDaylight plugin from scratch	D3.3 & D3.4
Topology & inventory	Topology manager (Table 7Table 7) COSIGN DCN Multi-layer Topology Manager (Table 8)	OpenDaylight extension	D3.3 & D3.4
Monitoring and fault manager	Monitoring and fault manager (Table 9)	OpenDaylight extension plus OpenDaylight plugin from scratch	D3.3 & D3.4
Optical resource virtualization manager	Optical resource virtualization manager (Table 10)	OpenVirteX plugin from scratch	D3.3 & D3.4
Virtual infrastructure manager	Virtual infrastructure manager (Table 11)	OpenVirteX extensions	D3.3 & D3.4
Path computation	Path Computation Manager (Table 12)	OpenDaylight plugin from scratch	D3.3 & D3.4

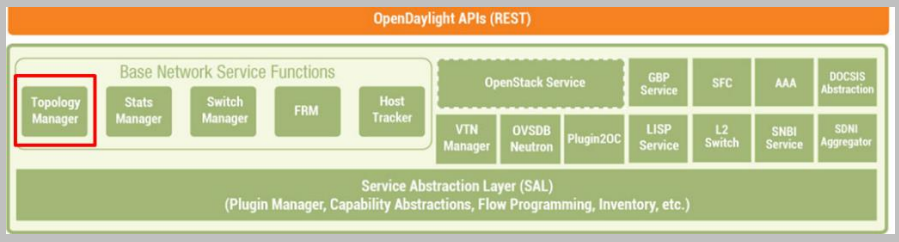
Overlay-based service virtualization	Overlay-Based Service Virtualization (Table 13)	OpenDaylight extension	D3.3 & D3.4
Policy Manager	Policy Manager (Table 14)	OpenDaylight plugin from scratch	D3.3
AAA	AAA (Table 16)	OpenDaylight extension	D3.3
SLA Manager	SLA Manager (Table 15)	OpenDaylight extension	D3.3
OF drivers	OF java library OF plugin (Table 17)	OpenDaylight extension	D3.3 & D3.4
OF agents	OF agent for ToR OF agent for optical ToR OF agent for Polatis large scale Switch OF agent for optical NIC OF agent for OXS (see deliverable D3.2 [3] for OF agents software design)	Software from scratch	D3.3 & D3.4
Abstraction Layer	DCN controller abstraction mechanisms and engine (Table 18)	OpenDaylight extension	D3.3 & D3.4

Table 6 – Optical Provisioning Manager

Software module	Optical Provisioning Manager
Functional component	Optical provisioning manager
D3.3 features	Provisioning of P2P optical paths Provisioning P2P multi-layer paths Support for QoS constraints Support for path modification requests
D3.4 features	Path protection and restoration Path scheduling Provisioning of P2MP and anycast paths Automated reconfiguration of virtual server layer in MLN scenarios Automated re-planning of flows for global optimization Automated adoption of operator policies for mice/elephant flows mapping on different physical technologies Integration with AuthN/AuthZ features
Software baseline	The Optical Provisioning Manager will be developed from scratch as a new OpenDaylight plugin operating over the MD-SAL.
High-level software design	The software architecture of the Optical Provisioning Manager is shown in *Figure 20*. The APIs of the service allows to perform CRUD actions on "connection" resources (see Deliverable D3.2 [3], Section 3.2.3, for the detailed information model). Both binding APIs for other internal services and RESTConf APIs for external applications are supported.  *Figure 20 – Optical Provisioning Manager* Connections resources are stored in the Config Datastore of the Open-

	Daylight MD-SAL, so that they can be manipulated and configured by external entities. In the Optical Provisioning Manager module, the "Request Manager" handles the requests for creation, modification and deletion of connections, received through the subscriptions to the data store. The "Optical Connections Manager" manages the whole set of active connections, through the instantiation of a "Connection Finite State Machine" when a new connection is created. This entity manages the whole lifecycle of a connection: it triggers the interaction with the other Open-Daylight components (using the mediation of the MD-SAL mechanisms) to compute, establish, modify, restore, re-plan or remove the path and update the Data Store consequently (e.g. to change the status of a connection).
Service API	North-bound API for the optical connectivity service, as specified in Deliverable D3.2 [3], Section 3.2.3. These APIs are based on RESTConf protocol, so that the service can be accessed by external components, and are modelled through a YANG model.
License	Eclipse Public License 1.0

Table 7 – Topology Manager

Software module	Topology manager
Functional component	Topology & inventory
D3.3 features	Multiple topology instances for different DCN views: technology domains topologies and inter-technology topologies. Optical characteristics and constraints extensions for COSIGN devices. Traffic engineering extensions.
D3.4 features	Calendar extensions in support of scheduling. Metric extensions (e.g. related to energy consumption) for customized operator policies.
Software baseline	OpenDaylight MD-SAL Topology Manager, included in the OpenDaylight Controller project (https://wiki.opendaylight.org/view/OpenDaylight_Controller:Main)
High-level software design	The Topology Manager is a core module within the OpenDaylight software architecture, as depicted below. It is a base network service function, and it is implemented as a common MD-SAL plugin where topologies are handled through dedicated data stores. This means that the interaction with the Topology Manager follows the principles of MD-SAL provider/consumer approach. Other internal OpenDaylight software modules can interact with the Topology Manager through read/write transactions or listening operations on the related MD-SAL data stores, while external applications can retrieve topology information through RESTCONF APIs.  *Figure 21 – The Topology Manager within OpenDaylight* The Topology Manager data model and APIs are defined by dedicated YANG models. The OpenDaylight Controller project already provides a YANG model for a base network topology, based on the related IETF draft (https://git.opendaylight.org/gerrit/gitweb?p=yangtools.git;f=model/ietf/ietf-topology/src/main/yang/network-topology@2013-10-21.yang;a=blob). This model defines the basic structure of a network topology that is built by the composition and interconnection of nodes, links and termination points. OpenDaylight allows the extension of this basic topology model by defining additional YANG models which enhance the base one with further specific attributes associated to the three elementary resources. This enhancement can be modelled through the standard augmentation mechanisms defined in the YANG language specification [RFC6020]. In particular, COSIGN will augment and extend the OpenDaylight base network topology in support of the technologies, devices and switching paradigm implemented in the project. Thus, as depicted below, COSIGN will extend the base network topology YANG model to describe the

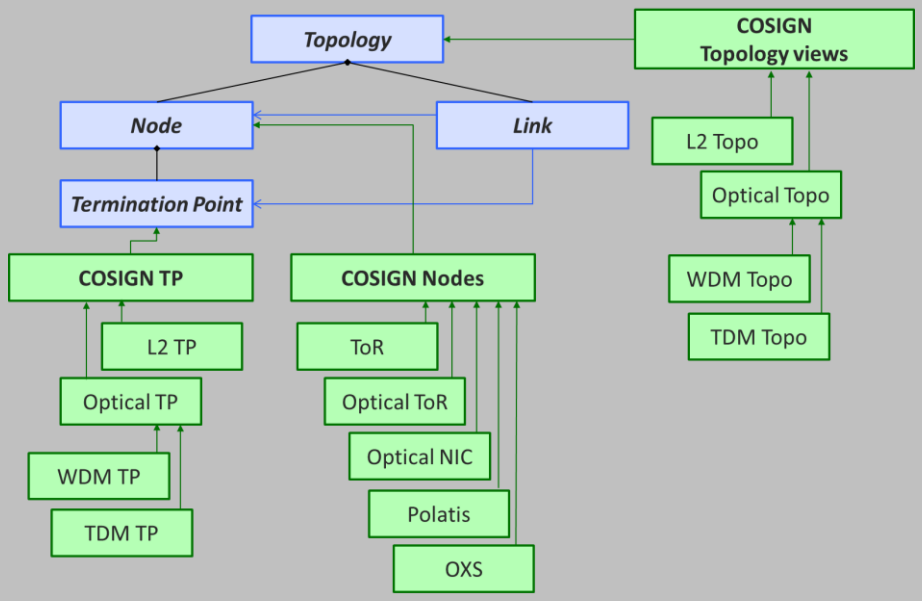
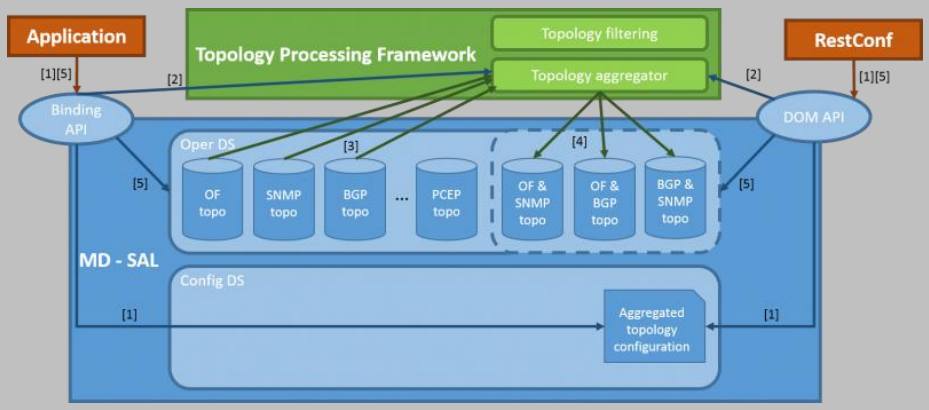
	capabilities and constraints of the different optical and Ethernet devices.  *Figure 22 – COSIGN network topology model*
Service API	The Topology Manager will expose a set of java and RESTCONF APIs derived from the extended YANG models described above for the COSIGN network topologies. In particular these APIs will provide *read* operations on the YANG resources, namely nodes, links and termination points.
License	Eclipse Public License 1.0

Table 8 – COSIGN DCN Multi-layer Topology Manager

Software module	COSIGN DCN Multi-layer Topology Manager
Functional component	Topology & inventory
D3.3 features	Creation and management of the multi-layer COSIGN DCN topology.
D3.4 features	As in D3.4, but operating with additional underlay topologies and aggregation or filtering criteria, depending on the technologies available at the different stages of the COSIGN data plane.
Software baseline	Topology Processing Framework (OpenDaylight project: https://wiki.opendaylight.org/view/Topology_Processing_Framework:Main) Note: this project is still in a preliminary stage and the plugin is not yet included in the latest official release of OpenDaylight (Helium), but just planned for the next release (Lithium). In COSIGN we will evaluate the level of maturity of the code available when starting the development and we will implement the fixes needed for a stable version.
High-level software design	This module will extend the corresponding OpenDaylight component in order to build a logical upper-level topology view of the entire COSIGN DC multi-layer network, including all the different nodes and technologies, from OVS instances and NICs to ToRs and optical switches. Figure 23 shows the software architecture of the original module already available in OpenDaylight. COSIGN prototype will extend the underlay topologies to include all the instances associated to COSIGN data plane devices (e.g. optical OF topology and OF topology) and will introduce new filtering and aggregation criteria to request, build and access the multi-layer topology of COSIGN DCN. The multi-layer and multi-technology topology will be extended through the augmentation of the associated YANG model, adding a representation of the technical constraints of COSIGN resources. Similarly, the filtering and aggregation criteria will be specified through the augmentation of the YANG model which defines the correlations between entities belonging to different underlay topologies.  *Figure 23 – Topology processing framework – original software design*
Service API	This service is designed to be consumed by other internal OpenDaylight plugins (e.g., the internal path computation component) through the binding API or by

	external applications (e.g. an external PCE server) through the RESTConf API. The implementation of both these APIs is auto-generated through the OpenDaylight tools starting from the definition of the YANG model. The APIs will enable two main actions: • Request to build a new multi-layer topology, providing as input the desired filtering and aggregation criteria. • Retrieve the resulting multi-layer topology or receive notifications about its modifications.
License	Eclipse Public License 1.0

Table 9 – Monitoring and fault manager

Software module	Monitoring and Fault Manager
Functional component	Monitoring and fault manager
D3.3 features	• Compilation of Monitoring parameters • Compilation of fault management alarms • Provisioning to the orchestration layer of the monitoring and fault management events. • Extensions to the Statistics base network functionality for COSIGN devices
D3.4 features	• Support for subscriptions to specific monitoring parameters from the upper layers. • Support for subscriptions to specific fault alarms from the upper layers. • Full module integration with the DCN control plane
Software baseline	OpenDaylight includes a module (the Statistics Manager) which collects the OpenFlow counters about nodes, ports, flows and queues (e.g. received and sent packets, received and sent bytes, etc.). This component will be extended to support the new counters for optical ports defined in deliverable D3.2 [3]. The Monitoring and Fault Manager will be implemented from scratch as a consumer of this extended Statistics Manager and, starting from the elementary information collected from this module and other entities (e.g. the Inventory Manager or directly from some OF drivers), it will elaborate performance and detect failures in the overall network.
High-level software design	The goal of the Monitoring and Fault Manager is to enable to the SDN controller with the typical performance monitoring management and fault management functionalities. More specifically the Monitoring and fault manager will carry out the following operations: • By means of the D-CPI interface, gather the monitoring information and failure notifications. • Handle the notifications and performance of DCN optical devices: Alarms, failures, dropped packets, congestion, etc.). • Forward to the network application layer running on top of the COSIGN controller all this information by means of the A-CPI interface. • By means of this same A-CPI, it should be enabled a mechanism to allow an application to configure performance monitoring, collection points, thresholds, or define the type of fault management alarms that the application desires to configure.

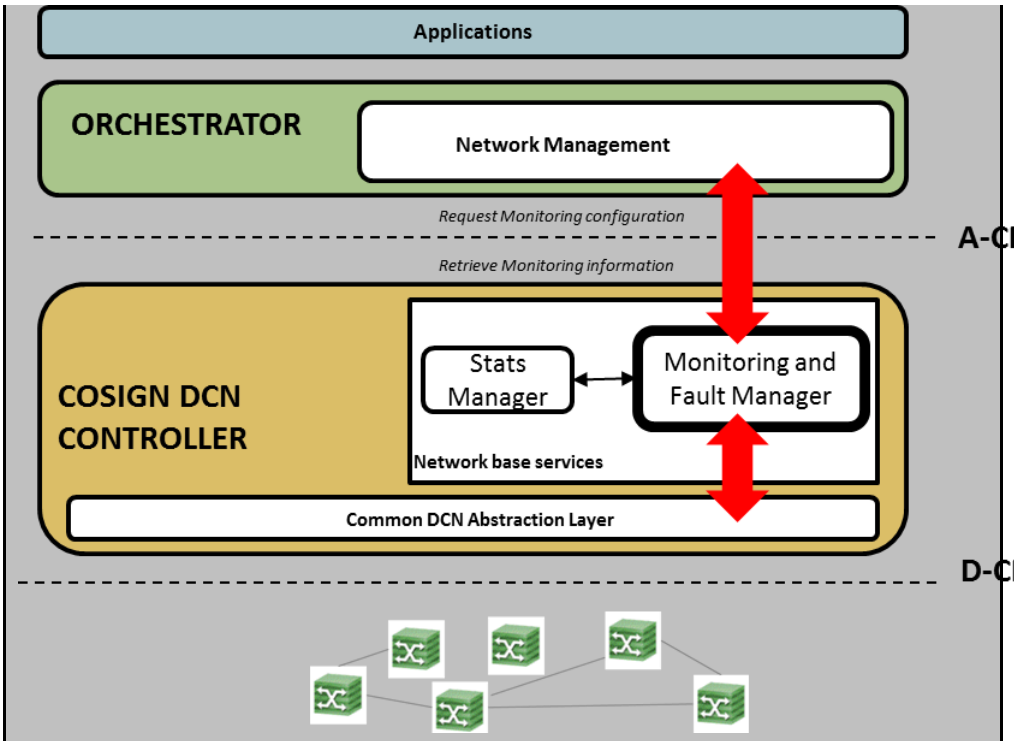
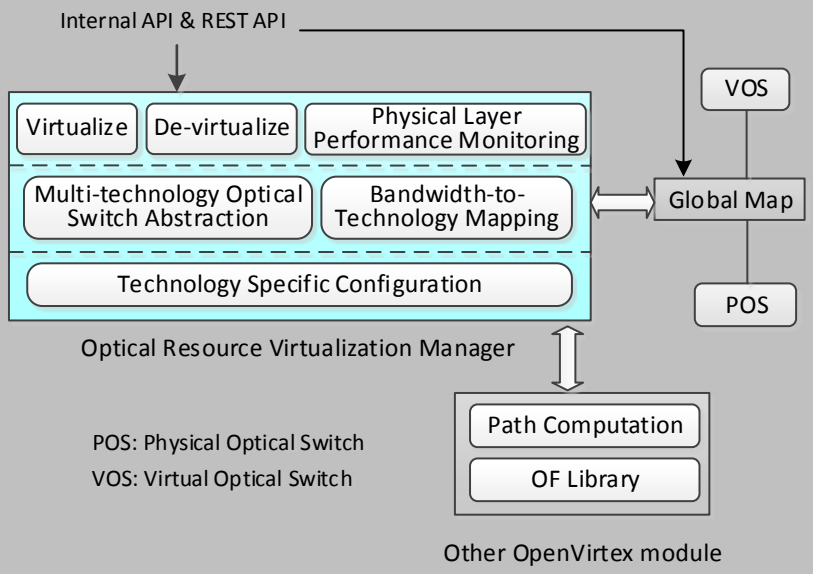


Figure 24 – Monitoring and Fault Manager in COSIGN DCN framework

Service API	The monitoring and fault management service will be consumed by external applications to the controller making use of the information provided by means of the A-DCPI interface. The service API should enable the following options: • Request the configuration (the orchestrator may request this in order to take optimal decisions on the coordination of IT and Network resources) of the desired performance monitoring parameters and statistics from the optical devices and network paths. Same configuration request applies to the fault management aspects. • Retrieve the requested information and notifications from the DCN data plane by means of the DCN monitoring and fault manager to be consumed by the orchestrator.
License	Eclipse Public License 1.0

Table 10 – Optical resource virtualization manager

Software module	Optical resource virtualization manager
Functional component	Optical Resource Virtualization Manager
D3.3 features	Provisioning of multi-technology optical switch abstraction Support for optical switch aggregation and disaggregation/slicing capability Provisioning of SLA-based virtualized optical network
D3.4 features	Automated SLA-based virtualized optical network mapping Support for virtual resource modification requests Monitor and manage virtualized optical network resources
Software baseline	The Optical Resource Virtualization Manager will be developed from scratch as a new OpenVirteX plugin.
High-level software design	The software architecture of the Optical Resource Virtualization Manager is shown in *Figure 25*. The APIs of the service allow performing CRUD actions on resources (see Deliverable D3.2 [3], Section 3.2.2, for the detailed information model). Both internal APIs for other internal services and RESTConf APIs for external applications are supported.  *Figure 25 – Optical Resource Virtualization Manager* In COSIGN data plane, different optical technologies will be employed and they will need to be abstracted to expose their features to the external entities (through *multi-technology optical switch abstraction*) and configured by technology specific message/interface (through *bandwidth-to-technology mapping* and *technology specific configuration*). The physical-virtual split will be implemented by the extended OpenFlow message *virtualization/de-virtualization* process in OpenVirteX, which references the *Global Map* (where the whole set of active VONs and their physical-virtual mappings are stored). *Performance Monitoring* will be used to monitor the optical channel quality (e.g., power) and to expose this information to external or internal service performance monitoring application (e.g., to react to performance degradation).

	In addition, the *Optical Resource Virtualization Manager* will need to communicate with or to extend some existing OpenVirteX modules to implement the VON mapping properly (e.g., calling the path computation to map optical virtual link to a physical path, and extending the OF library with optical features).
Service API	North-bound API for the virtual optical slice provisioning service, as specified in Deliverable D3.2 [3], Section 3.2.2. These APIs are based on RESTConf protocol, so that the service can be accessed by external components.
License	Eclipse Public License 1.0

Table 11 – Virtual Infrastructure Manager

Software module	Virtual Infrastructure Manager
Functional component	Virtual Infrastructure Manager
D3.3 features	Provisioning of SLA-based virtual network slices, including virtualized optical resources and electronic resources Support for virtual network modification requests
D3.4 features	Automated virtualized infrastructure mapping, including dynamic mapping algorithm Support for virtual infrastructure modification requests Monitor and manage virtualized infrastructure, in compliance with the established SLAs
Software baseline	OpenVirteX. (http://ovx.onlab.us/documentation/architecture/overview/) Note: to support multi-domain virtual infrastructure provisioning (e.g., belonging to different infrastructure provider), an Orchestrator may need to be developed which sits between users and domain virtual infrastructure manager.
High-level software design	The Virtual Infrastructure Manager will support address virtualization, topology virtualization and control function virtualization, which will be implemented based on virtualization and de-virtualization approach in OpenVirteX. The service will manage and configure the virtual optical infrastructure through the optical resource virtualization manager. Moreover, the software will expose APIs used by external SDN applications (or multi-domain Virtual Infrastructure Orchestrator) to create their own virtual topologies on demand without dealing and caring of the different technologies constraints.
Service API	The APIs of the service allow performing CRUD actions on resources. Both internal APIs for other internal services and RESTConf APIs for external applications are supported.
License	Eclipse Public License 1.0

Table 12 – Path Computation Manager

Software module	Path Computation Manager
Functional component	Path Computation
D3.3 features	Path computation for point-to-point multi-layer paths with TE metrics and optical constraints.
D3.4 features	Path computation for scheduled paths. Path computation with constraints about nodes/links to be excluded. Path computation for anycast and P2MP paths. Path computation with objective functions related to more advanced operator policies (e.g. energy consumption, global resource utilization, etc.)
Software baseline	OpenDaylight implements the Dijkstra routing algorithms over a network graph built over the network view provided by the Topology Manager. However, this basic implementation does not allow to support complex constraints or objective functions like the ones required in COSIGN. For this reason we will define a more flexible component, written from scratch, which implements different types of routing algorithms and is able to select dynamically the most suitable one taking into account the characteristics and constraints of the requested path.
High-level software design	The Path Computation Manager is developed as a new OpenDaylight internal plugin which acts as a consumer of the Topology Manager (in particular the multi-layer topology manager) and provides a service to compute different kinds of network paths in the DCN networks. This service is typically consumed by other OpenDaylight modules (e.g. the Optical Provisioning Manager) through java APIs. The high-level software architecture of the Path Computation Manager is shown in *Figure 26*. The component operates in stateful mode and offers APIs (both java and RESTConf APIs) to request the computation of a new path, its release, or the resolution of a path key for a path previously computed. The *Path Request Manager* is the entity in charge of processing the incoming requests. Each request is associated to one or more paths, which are logical entities managed through the Paths Manager. Each path has a lifecycle according to a Finite State Machine and some characteristics like end-points, metrics, objective functions (e.g. minimization of hops or energy consumption), and constraints (e.g. time duration, exclusion of some resources). Depending on these constraints, the module invoke dynamically the more suitable routing algorithms from the available set. Each routing algorithm computes paths following a common procedure: it retrieves the current network view from the Topology Manager, builds the associated network graph depending on the path characteristics and elaborate the list of hops are resources along the path. This information is stored in the MD-SAL operational data store so that the results can be retrieved for the overall duration of the path.

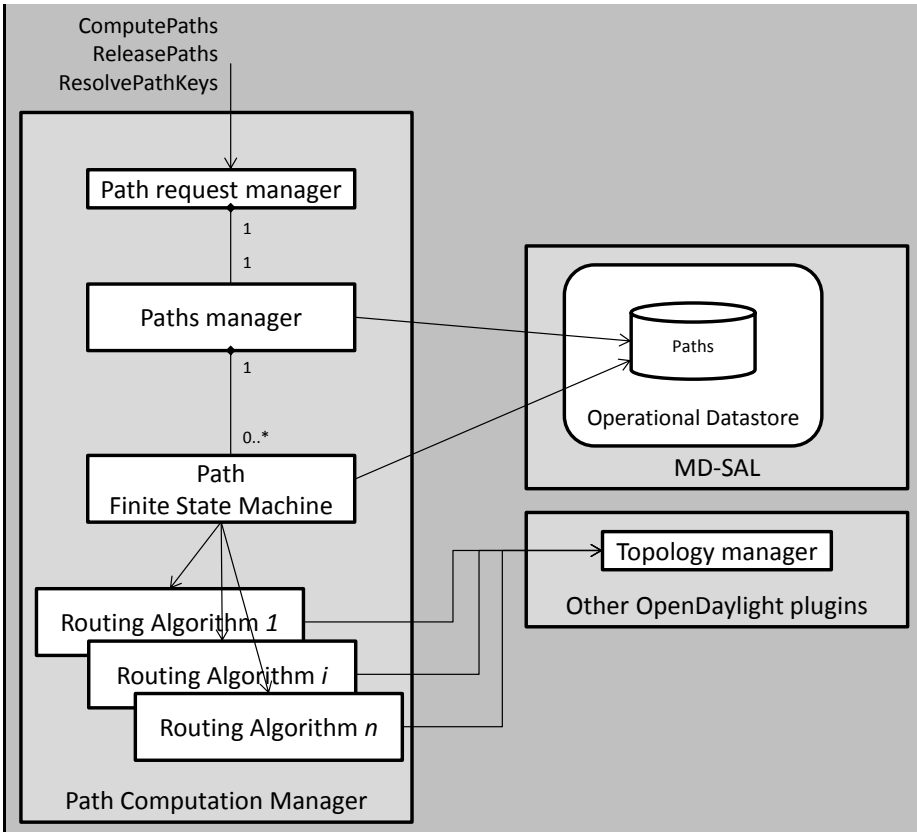


Figure 26 – Path Computation Manager

Service API	The Path Computation Manager implements both java APIs and RESTConf APIs, which are defined through a YANG model following the same approach of the other OpenDaylight MD-SAL plugins. However, since the service is invoked only by internal OpenDaylight modules, only the java APIs are mostly used. The YANG model defines the main data structures for the *path* resource and three remote procedure calls, as follows: • ComputePaths: request to compute one or more new paths, with a given set of constraints and characteristics • ReleasePaths: request to remove one or more paths previously computed • ResolvePathKeys: request to return the explicit route of one or more active paths previously computed, given their unique identifier (i.e. their path key).
License	Eclipse Public License 1.0

Table 13 – Overlay-Based Service Virtualization

Software module	Overlay-Based Service Virtualization
Functional component	Overlay-Based Service Virtualization
Software baseline	The Overlay-Based Service Virtualization will be developed based on openDOVE overlay technology, which is an OpenDaylight plugin.
High-level software design	The software architecture of the Overlay-Based Service Virtualization is shown below.

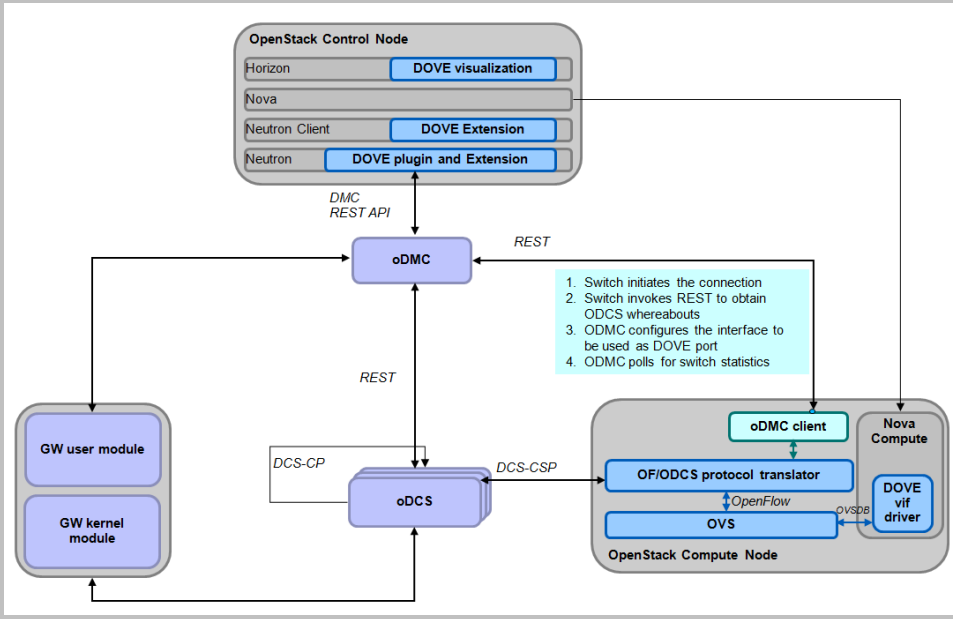


Figure 27 – DOVE

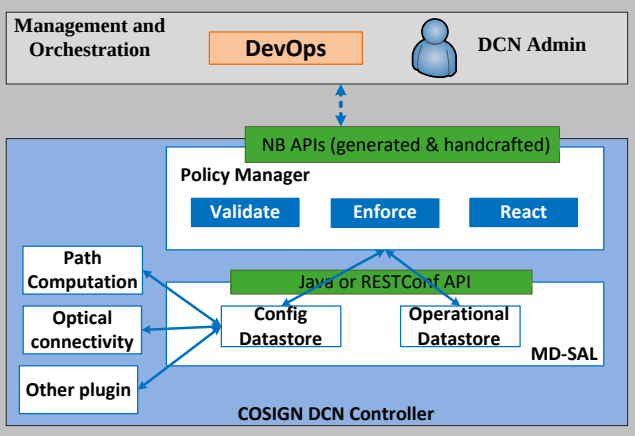
DOVE is a distributed overlay virtual network software solution that provides a virtualized form of a physical network without requiring any changes to the real physical network. DOVE architecture abstracts the underlying network for the virtual environment and presents it as a service or as an infrastructure in the form of an overlay network, which creates a more flexible network by creating a virtualized network for virtual machines, without requiring any type of multicast to be enabled on the physical network. This virtual network is de-coupled and isolated from the physical network much like a virtual machine is de-coupled and isolated from its host server hardware. DOVE takes a host-based overlay approach, which achieves advanced network abstraction that enables application-level network services in large-scale multi-tenant environments. It provides a multi-hypervisor, server-centric solution that is composed of multiple components that overlay virtual networks onto any physical network that provides IP connectivity.

Regardless of the hypervisor platform that is used in the cloud, DOVE components (see *Figure 27*) represent one of the building blocks of the virtualized network environment. These four software components work in combination to provide effective host-based network virtualization.

Open DOVE Management Console (oDMC): A management console is the centralized point of control for configuring DOVE. It configures each virtual network, controls policies, and disseminates policies to the virtual switches.

	It also helps administrators manage individual virtual networks. The software is on a server as a virtual appliance. **Open Distributed Connectivity Service (oDCS):** A connectivity service disseminates VM addresses to the virtual switches participating in a DOVE virtual network. The connectivity server configures the virtual network, controls policies, and disseminates policies to the virtual switches. The oDCS software can be deployed as a cluster of virtual appliances, and offers more scalability and reliability of the traditional network control plane. **DOVE Gateways (DGWs):** DOVE Gateways are specialized appliances that connect the DOVE overlay network environment with a non-DOVE environment. **DOVE Client**: DOVE agent process runs on each individual hypervisor and listens for direction from the Distributed Connectivity Service. The DOVE agents are also responsible for creating the individual bridged interfaces that ultimately show up in the Virtual Machine Manager. The agents provide network virtualization over a UDP VXLAN overlay, and implements the data path of the virtual network.
Service API	North-bound API for the Overlay-Based Service Virtualization, as specified in Deliverable D3.2 [3], Section 3.2.1. These APIs are based OpenStack networking API v2.0. OpenDaylight SDN controller is now merged onto OpenStack Icehouse and therefore should support these REST APIs.
License	Eclipse Public License 1.0

Table 14 – Policy Manager

Software module	COSIGN DCN Policy Manager
Functional component	Policy Manager
D3.3 features	Configuration of policies to control the behaviour of various components residing in the control plane. Continuous verification of policies for validity. Trigger reaction (notify administrator, automatically respond by reinforcing the policy) when policy becomes invalid.
D3.4 features	-
Software baseline	There is no module in ODL that matches the functionality of the Policy manager therefore the module (Policy manager) will be implemented from scratch. It is however possible to reuse elements from the Group-based Policy [ODL-POL] plugin and the Network Intent Composition plugin [ODL-NIC].
High-level software design	The goal of the Policy manager is to allow the administrator of the DCN to configure and manage the behaviour of the infrastructure control components. *Figure 28* illustrates the high level architecture of the Policy manager. The Policy manager interacts with the *config* and *operational* datastores in MD-SAL. More specifically, the policies are stored in the config datastore. The Policy manager uses information from the operational datastore (various statistics about the network, usage of resources, etc.) to validate the policies and to continuously verify if policies are valid. The interaction between the Policy manager and the MD-SAL is enabled through the APIs (Java or RESTConf) automatically generated from the YANG models. The Policy manager performs three main operations: 1) Validates new policies by using the information available in both config and operations datastores. 2) Enforces the validated policies by configuring a set of elements in the config datastore (to adjust the behaviour of other components such as path computation, etc.). 3) Continuously monitors the state of the network and the state of the control plane components by monitoring the information in the datastores. 4) When a policy becomes invalid due to network changes, the Policy manager reacts by trying to reinforce the policy or notify the admin.  *Figure 28 – Policy Manager architecture*

Service API	The service API consists of automatically generated Java and RESTConf APIs. Apart from the generated APIs the Policy manager module exposes a set of handcrafted APIs for receiving policies from external entities. These handcrafted APIs may serve as an adaptation layer between external entities (applications, DevOps tools) and the MD-SAL generated APIs. The service API allows CRUD operations for the policies.
License	Eclipse Public License 1.0

Table 15 – SLA Manager

Software module	COSIGN DCN SLA Manager
Functional component	SLA Manager
D3.3 features	Configuration of SLAs. Automatic enforcing of SLAs. Monitoring the metrics to ensure the validity of SLAs.
D3.4 features	-
Software baseline	The SLA manager will be based on two existing ODL projects: 1) Group-based Policy [ODL-POL]: it allows specifying policies between endpoints or groups of endpoints. Such a policy comprises a request for resources with specific characteristics. The policy may thus include aspects related to the SLA between the requestor (customer) and the DC provider. 2) Network Intent Composition [ODL-NIC]: This project allows applications to dynamically configure intents for the DC infrastructure. Such intents may comprise connectivity, QoS, chaining of network functions, etc. The constraints and use cases included in [ODL-NIC] so far, do not exactly match the requirements for the SLA manager. Nevertheless, SLAs can be extracted from the intents and further configured using the NB API exposed by the SLA manager.
High-level software design	The architecture of the SLA manager (*Figure 29*) resembles the architecture of the Policy manager. The difference lies in the purpose of the SLAs when compared to policies. An SLA captures the contractual terms between a customer and the DC provider. Such terms may include QoS, service availability, actions to take in case the measured service performance metrics do not comply with the agreed thresholds, etc. The SLA for a service request may be conveyed together with the request or separately. It can be specially tailored to customer's needs or selected from a predefined set, or built from predefined elements. The orchestrator understands and extracts the SLA, and further configures the SLA using the NB APIs exposed by the SLA Manager. The SLA manager performs the following operations: 1) Validates the SLAs: it verifies the resources availability in the operational datastore and possible conflicts other configured SLAs in the config datastore. 2) Enforces the SLAs. 3) Monitors the performance metrics and reacts in case the metrics do not correspond to agreed values.

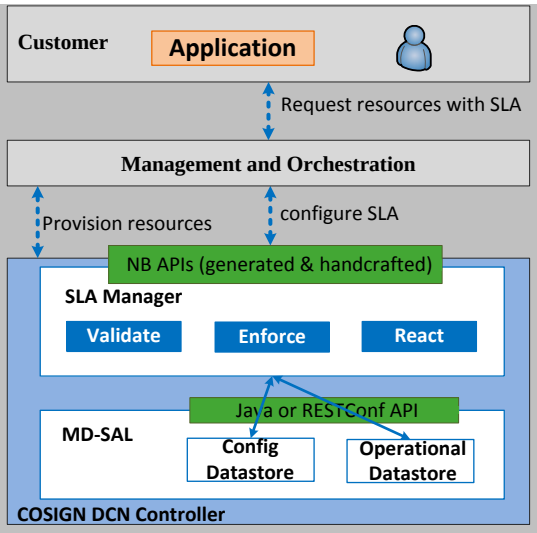


Figure 29 – SLA Manager architecture

Service API	The service API, which is described in deliverable D3.2, allows CRUD operations for SLAs.
License	Eclipse Public License 1.0

Table 16 – AAA

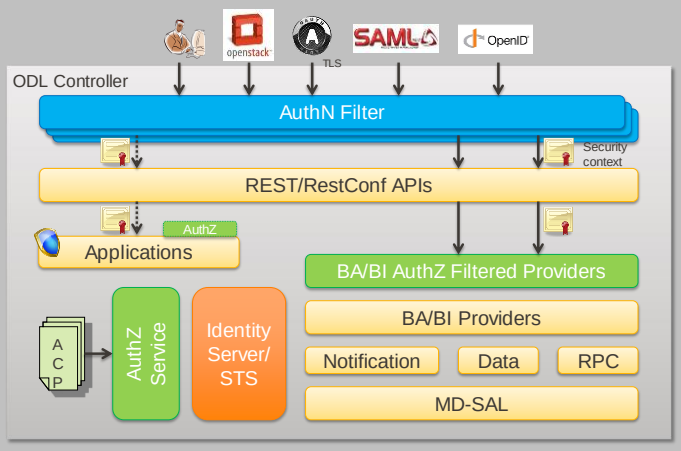
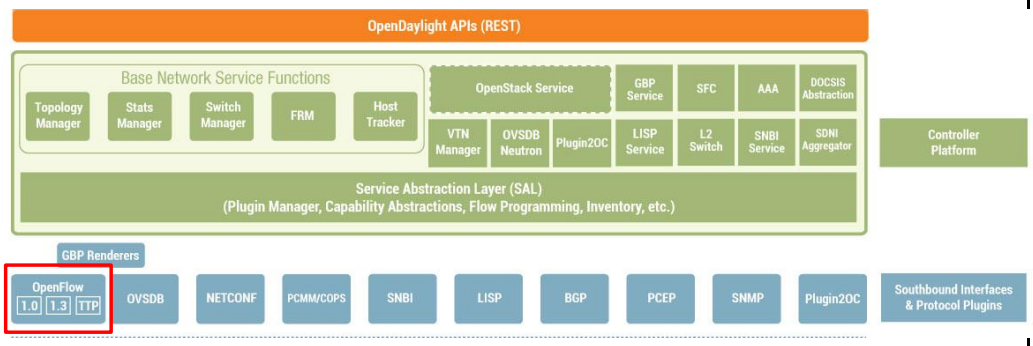
Software module	COSIGN DCN AAA
Functional component	AAA engine
D3.3 features	Token-based authentication using password credentials. Authentication based on users, domains and roles. A domain represents a set of resources that the user is allowed to access depending on the role that user has on that domain. Federated authentication (using Identity providers (IdP)).
D3.4 features	-
Software baseline	This module is based on the AAA plugin provided by ODL [ODL-AAA].
High-level software design	*Figure 30* depicts the architecture of the ODL AAA plugin. The authentication can work with a number of frameworks such as SAML, OpenID, of Keystone tokens (from Openstack). For a user to be able to access the RESTConf API it must first be authenticated thus it must pass through an authentication filter. The user is then given a token based on which the user is authorized to access specific data in MD-SAL datastores.  *Figure 30 – AAA engine architecture [ODL-AAA]*
Service API	The NB APIs exposed by the AAA service allow users to authenticate using password credentials. For users with *admin* roles, the API allows more operations such as creating domains, roles and association of user, roles and domain.
License	Eclipse Public License 1.0

Table 17 – OF drivers

Software module	OpenFlow plugin and OpenFlow java library
Functional component	OF drivers
D3.3 features	Suppport for OpenFlow extensions, as defined in the COSIGN south-bound interface specification in deliverable D3.2 [3].
D3.4 features	Suppport for OpenFlow extensions, as defined in the COSIGN south-bound interface specification in deliverable D3.2 [3] (support for additional COSIGN devices).
Software baseline	COSIGN OpenFlow drivers will be developed as extensions of the corresponding OpenDaylight functional components (see).  *Figure 31 – OpenFlow driver within OpenDaylight* In particular, in OpenDaylight the OpenFlow driver is implemented through different interdependent plugins: • OpenFlow protocol java library: implements the OpenFlow protocol to allow the communication channel between the OpenDaylight controller and the OF devices. (OpenDaylight project: https://wiki.opendaylight.org/view/Openflow_Protocol_Library:Main) • OpenFlow plugin: the MD-SAL south-bound plugin supporting OpenFlow 1.0 and OpenFlow 1.3.x through the integration of the OpenFlow protocol java library. (OpenDaylight project: https://wiki.opendaylight.org/view/OpenDaylight_OpenFlow_Plugin:Main) • Table Type Pattern: provides the YANG models for the Table Type Pattern (TTP), the standard defined by the ONF Forwarding Abstraction WG to allow the negotiation of functionalities between OF controller and switches to manage the variety of OpenFlow versions. (OpenDaylight project: https://wiki.opendaylight.org/view/Table_Type_Patterns:Main)
High-level software design	In COSIGN the OpenFlow driver already available in OpenDaylight will be modified to implement the protocol extensions defined in the COSIGN south-bound interface specification in deliverable D3.2 [3]. In particular, the modifications are related to the OpenFlow protocol java library, which provides the YANG models for the OpenFlow structures (e.g. OF actions, OF matches, etc.) and messages, as well as their internal implementation. In detail, the implementation of an OF extension requires the following steps: • Augmentation of the YANG model for the associated OpenFlow structure or message.

	• Creation of new serializers and deserializers as new classes extending OFSerializer and OFDeserializer respectively. • Registration of the new serializers/deserializers in the library.
Service API	The OpenFlow drivers provide java APIs to allow other internal components of OpenDaylight to exchange (extended) OpenFlow messages with the data plane devices.
License	Eclipse Public License 1.0

Table 18 – Abstraction layer component

Software module	DCN controller abstraction mechanisms and engine
Functional component	Abstraction layer
D3.3 features	• Implementation of the APIs which enable service to request for resources, i.e. implementation of the interface towards the SDN controller network service functions (I). For example: o With regards the Optical Provisioning Manager, this module sits on top of the common DCN abstraction layer and it uses the abstracted view of the DCN resources. o The Topology & Inventory module is directly interfaced with the common DCN abstraction layer for topology discovery and learning purposes. o The Monitoring and Fault Manager directly interacts with the abstraction layer to gather all the monitoring information through the D-CPI interface. • Implementation of extensions and D-CPI plugins to exploit the specific capabilities and granularities of the different optical resources (II).
D3.4 features	• Implementation of extensions and D-CPI plugins to exploit the specific capabilities and granularities of the different optical resources (II) – extended to additional data plane technologies.
Software baseline	This software module will rely on the abstraction mechanisms provided by the ODL reference control plane framework in order to define COSIGN own models and define new abstraction mechanisms to adopt the underlying Optical DP resources and expose the service abstraction towards the upper Orchestration and Customer layers of the COSIGN architecture. OpenDayLight offers an SDN controller platform which is suitable to be extended with the new modules and functions of the COSIGN controller. The COSIGN abstraction layer (Section 4.1.3) can be mapped on the OpenDayLight Model-Driven Service Abstraction Layer (MD-SAL), which enables the abstraction of the services implemented by different types of OpenDayLight plugins. COSIGN development will not focus on the internal functionalities of the MD-SAL, which will be simply re-used in the COSIGN controller. Instead, COSIGN development activities will implement new information models that will define the abstract version of the DCN resources and infrastructure, as well as the plugins which will provide the translation towards these abstract models. Following the MD-SAL approach, each information model will be expressed in the YANG language [RFC6020] and, where possible, will be built through the re-usage, composition and augmentation of elements from the standard YANG models defined in the IETF Netmod WG.
High-level software design	The main objective of the abstraction layer is to provide a homogenous view of the DCN infrastructure to facilitate the management and control of the heterogeneous devices and technologies that compose the data plane. The service abstraction layer acts like a large registry of services advertised by various modules and binds them to the applications that require them. Modules providing services, or producers, can register their APIs with the registry. When an application, or a consumer, requests a service via a generic API, the service abstraction layer is responsible for assembling the request by binding producer and consumer into a

contract, brokered and serviced by the abstraction layer.

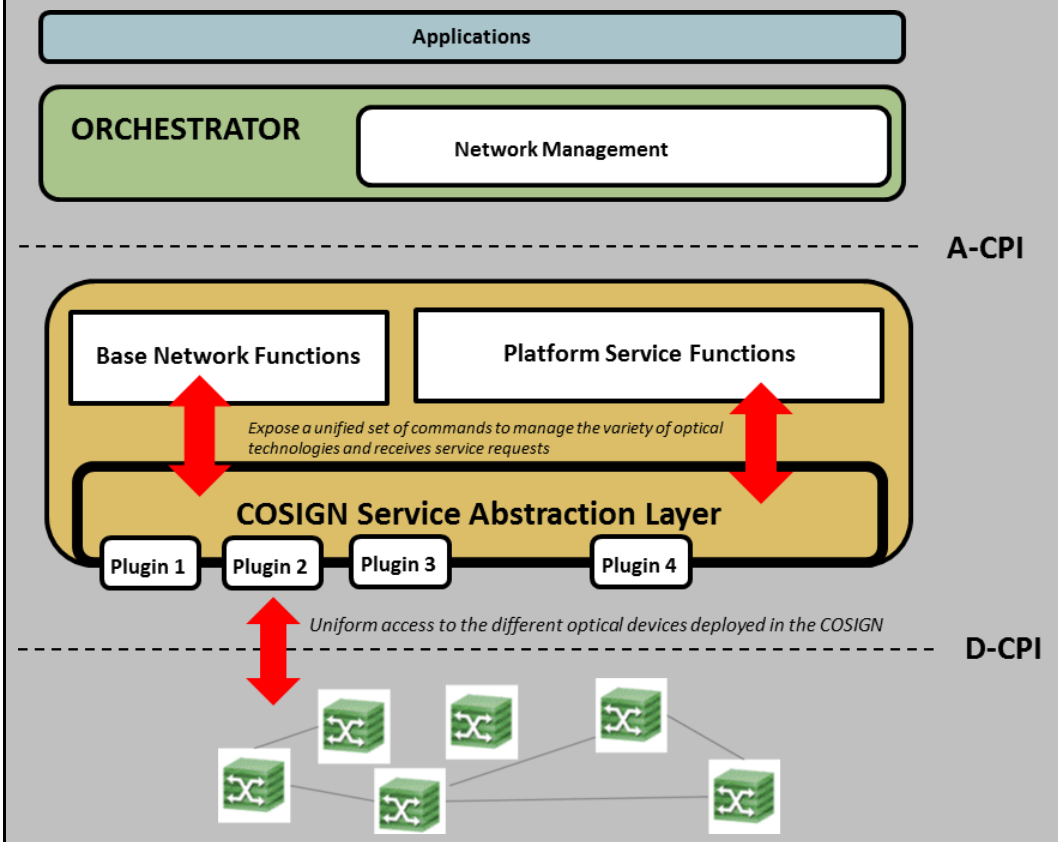


Figure 32 – COSIGN Service Abstraction Layer (based on OpenDaylight MD-SAL)

Service API	The abstraction layer service API consists of a common and uniform access to the different optical devices deployed in the COSIGN hybrid data centre fabric. The plugins are dynamically linked to the abstraction layer which determines how to fulfil the service requested (by applications). On the other hand, the abstraction layer exposes a unified and protocol-independent set of commands and primitives to manage and configure the variety of optical technologies deployed in the data plane.
License	Eclipse Public License v1.0