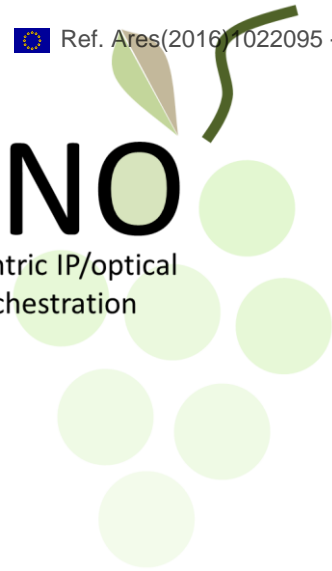


ACINO

Application-Centric IP/optical
Network Orchestration



ACINO / D3.1

The framework for the application-centric network orchestrator

Dissemination level	PU
Version	1
Due date	31.01.2016
Version date	29.02.2016



Document information

Authors

Domenico Siracusa – CREATE-NET

Federico Pederzoli – CREATE-NET

Michele Santuari – CREATE-NET

Roberto Doriguzzi – CREATE-NET

Victor Lopez – TID

Jose Manuel Gran Josa – TID

Ori Gerstel – SEDONA

Thomas Szyrkowiec – ADVA

Mohit Chamanian – ADVA

Editor

Domenico Siracusa – CREATE-NET

Project funding

645127 — ACINO — H2020-ICT-2014

Legal Disclaimer

The information in this document is provided 'as is', and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law.

Revision and history chart

Version	Date	Author	Comment
0.1	10/08/2015	All	MS14
0.2	21/12/2015	Federico Pederzolli (CREATE-NET)	Integrated last plenary requirements
0.3	24/12/2015	Domenico Siracusa (CREATE-NET) Federico Pederzolli (CREATE-NET)	First draft of Multi-Layer models description
0.4	11/01/2016	Federico Pederzolli (CREATE-NET)	Review of existing text
0.5	15/01/2016	Domenico Siracusa (CREATE-NET) Federico Pederzolli (CREATE-NET)	Improved Arch. overview and description
0.6	20/01/2016	Federico Pederzolli (CREATE-NET)	Added additional details on the ONOS arch. and how it relates to the ACINO architecture
0.7	21/01/2016	Victor Lopez (TID), Jose Manuel Gran Josa (TID), Federico Pederzolli (CREATE-NET)	Added description of the ABNO architecture and available implementation
0.8	26/01/2016	Michele Santuari (CREATE-NET), Federico Pederzolli (CREATE-NET)	Added description of the ONOS model
0.9	11/02/2016	Federico Pederzolli (CREATE-NET), Michele Santuari (CREATE-NET)	Implemented most comments from Plenary Meeting, added first draft of examples of Orchestrator operations, added additional detail on ONOS model and architecture
0.91	12/02/2016	Roberto Doriguzzi (CREATE-NET), Federico Pederzolli (CREATE-NET)	Document review
0.92	16/02/2016	Domenico Siracusa (CREATE-NET), Federico Pederzolli (CREATE-NET)	Document Review
1	29/02/2016	Domenico Siracusa (CREATE-NET) Mohit Chamanian (ADVA), Victor Lopez (TID), Federico Pederzolli (CREATE-NET)	Final Document Review

Table of contents

1	Introduction	8
2	ACINO Architecture Overview	10
3	ACINO Orchestrator Requirements	13
3.1	Requirements of the Northbound Interface	13
3.1.1	Dynamic Intent-driven Service Management Interface	13
3.1.2	Multi-layer Topology and Planning Interface	14
3.2	Requirements of the Southbound Interface	14
3.2.1	IP Controller Interface	14
3.2.2	Optical Controller Interface	15
3.3	Requirements of the ACINO Orchestrator	16
4	State of the Art of IP and Optical Control	18
4.1	Overview of SDN frameworks	18
4.1.1	ABNO	19
4.1.2	OpenDayLight	21
4.1.3	ONOS	22
4.1.4	ODENOS	23
4.1.5	Summary of the most promising SDN Frameworks	24
4.1.6	Selected framework for the development of the ACINO Orchestrator	26
4.2	Optical Controllers	27
4.2.1	North-bound Protocols for Optical Controllers	27
4.2.2	ADVA Optical Network Hypervisor	29
4.3	IP Controllers	31
4.3.1	North-bound Protocols for IP/MPLS Controllers	32
4.3.2	Active Stateful PCE / ABNO	32
4.3.3	ONOS-IP	33
5	Multi-layer Network Models	34
5.1	ONF Core Information Model	34
5.1.1	Transport-API	35

5.2	IETF Topology Models	36
5.2.1	OpenConfig	38
5.3	ONOS Model	39
5.4	Selected Model for the ACINO Orchestrator	40
6	ACINO Orchestrator Architecture	42
6.1	Detailed Internal Architecture	43
6.1.1	North-Bound Interface Module	43
6.1.2	Intent/Primitive Framework Module	43
6.1.3	Network State and Inventory DB and Manager Module (NSDB)	44
6.1.4	Multi-Layer Provisioning Module	44
6.1.5	Online planning Tool Module	44
6.1.6	Transaction Handler Module	45
6.1.7	Abstraction Layer Modules	45
6.1.8	Southbound Interface Modules	45
6.2	Examples of typical Orchestrator Operations	46
6.2.1	Intent-based Dynamic Connection Provisioning	46
6.2.2	What-If Scenario	47
6.2.3	Global Re-Optimization	47
6.2.4	Restoration Scenario	48
7	Conclusions	49

List of figures

Fig. 1: Simplified schema of the ACINO orchestrator.....	11
Fig. 2 : Generic ABNO Architecture	20
Fig. 3: High-level Architecture of OpenDayLight [ODL].....	22
Fig. 4: High-level Architecture of ONOS [ONOS].	22
Fig. 5: High-level Architecture of ODNOS [ODNOS].....	23
Fig. 6: Control Stack of an optical network running an SDN controller/orchestrator and the ADVA hypervisor.....	30
Fig. 7: Possible integration of the ADVA Optical Network Hypervisor with the ACINO Orchestrator.	31
Fig. 8: ONF Core Information Model schema [ONFCIM].	35
Fig. 9: IETF Model Hierarchy.....	36
Fig. 10: IETF model topology hierarchy example [CLETOPO].	37
Fig. 11: Proposed architecture for the ACINO orchestrator.....	42

List of tables

Table 1: Overview of most known SDN frameworks (source: [KRE14]).....	18
Table 2: Summary of the features of the three most promising SDN frameworks: OpenDayLight, ONOS and ODNOS.....	24
Table 3: Summary of the ONF Network Model.	34
Table 4: Summary of the IETF Network Model.	38
Table 5: Summary of the ONOS Network Model.	40

Summary

This deliverable describes the requirements and proposed architecture of the ACINO orchestrator. It gives an overview of the main elements of the architecture, and then the detailed requirements for each of these components. It provides an overview of existing SDN frameworks with the intent of identifying candidates for controlling the IP/MPLS network, the optical network, or realizing the ACINO orchestrator. The document also surveys several multi-layer network models, proposed in multiple standardization bodies, and identifies a potential candidate for the ACINO orchestrator along with an initial list of extensions required. The document identifies ONOS as the candidate framework for developing the ACINO architecture, outlines the mapping of the elements of the ACINO architecture on existing ONOS components, and presents the new components that would be developed in the ACINO project.

1 Introduction

The primary goal of the ACINO architecture is to guarantee application demands/requirements in a multi-layer IP-over optical network. A key component in such an architecture is the ability to “understand” capabilities and configure the underlying packet and optical infrastructure. However, vendors of IP and optical equipment typically ship their hardware bundled with custom, proprietary control plane solutions. This is especially true in the optical domain, where the analog nature of the media and complexities associated with physical layer impairments has led to the proliferation of vendor-specific control solutions. As a result, it is realistic to assume that multi-layer resource management operations will have to interact with technology-specific and sometimes vendor-specific control interfaces, and a central “Network Orchestrator” will coordinate operations across the multiple technology and vendor domains.

Furthermore, the applications that drive Internet traffic have evolved beyond simple requirements that can be easily and cheaply met with a best-effort network and a “reliable” transport protocol. Today’s applications can have stringent requirements in terms of latency and jitter (e.g. Audio over IP), bandwidth (e.g., Video over IP), reliability, security, etc. Despite these diverse requirements, their traffic is ultimately routed over the same optical connections in the optical layer, barring some clever Traffic Engineering efforts in the Multi Protocol label Switching (MPLS) layer typically used in transport networks. This “blind” approach makes it difficult to provide adequate services to some applications, whose requirements are often implicit and not known with enough detail to the control plane of the transport network. This results in a mostly “blind” allocation of traffic to MPLS and optical paths, which is not guaranteed to actually satisfy the service requirements for that traffic. While it is theoretically possible to offer the best possible service characteristics all the way down to the optical transport layer (e.g. using a fully meshed optical bypass virtual topology), it would be prohibitively expensive. Therefore, a smarter approach is needed.

The ACINO project proposes to realize such an application-centric, multi-layer network orchestrator, capable of: (i) interfacing directly with large applications, exposing a North-Bound Interface (NBI) through which to allow them to submit service requests with explicit requirements, expressed as high-level, friendly “intents” rather than directly using low-level network configuration commands (other than strictly necessary ingress/egress port and traffic selection parameters), (ii) learning the state of and controlling both the packet and optical layers of a transport network domain through an appropriate South-Bound Interface (SBI), and (iii) translating, through appropriate dynamic and planning algorithms, such high-level intents into optimized implementable network configuration.

There are many aspects to be taken into account during the design and implementation of the ACINO orchestrator. The first is its scope: the orchestrator will operate over a single carrier scenario¹, where a network operator owns an infrastructure composed of packet and optical domains, with devices produced by multiple vendors². Secondly, the functional requirements that targeted applications impose on the orches-

¹ This can be relaxed in the future, but it is not among the priorities of the project.

² In practical terms, for the purpose of demonstration optical equipment from ADVA and IP routers from Juniper are currently committed to the project.

trator, and which it, in turn, imposes on the underlying control planes (be them SDN controllers or more traditional control interfaces) must be explored. Thirdly, in order to realize multi-layer optimization, the underlying technologies must expose an abstract yet appropriately rich multi-layer topology, which can be used by the orchestrator to compute optimized assignments of network resources to incoming demands. Components and algorithms performing multi-layer optimizations (to be designed in tasks T3.3 and T3.4) must also take into account customer applications demands, which will be expressed directly by the applications using an “intent-based” interface. This interface will facilitate the definition of application requirements without delving into network configuration, and will be translated into network-aware service requests by the ACINO orchestrator. This process will be studied in task T3.2.

With respect to the framework for implementing the orchestrator, the position of the consortium is to leverage and improve an existing solution, according to the objectives and use-cases of the project, which drive the requirements for the orchestrator. Multiple alternatives are reviewed in this document, which differ in terms of architectures, capabilities, interfaces, etc. The candidate solution that will be selected as a starting framework must also satisfy an important requirement related with innovation: it must be an open-source framework with a strong developer community supporting it.

Overall, this deliverable describes the structure, requirements, interfaces and internal architecture of the application-centric SDN orchestrator that will implement the ACINO concept. The selected framework and the described architecture will feed the WP4 activities and the development process.

The document is structured as follows: first, Section 2 gives a high-level overview of the architecture of the ACINO orchestrator and its interactions with the surrounding actors (high-level applications, lower level controllers/control planes). Then, Section 3 outlines the requirements for the application-facing interface and the underlying IP/MPLS and optical controllers according to the ACINO Orchestrator directives. Section 4 contains an overview of existing SDN frameworks, and describes in greater detail both which are suitable as starting points for the implementation of the ACINO Orchestrator, and which can be used as underlying IP/MPLS or optical controllers, including an analysis of the possible protocols which could be used to implement the interface between them. Section 5 describes multiple multi-layer network models being developed in several standardization bodies, which can be used as a basis for the internal multi-layer network representation used by the Orchestrator, and related south-bound API models for the ACINO network-facing interface. Lastly, section 6 gives a detailed description of the proposed internal architecture of the ACINO orchestrator and examples of how the proposed modules coordinate to perform the orchestration functions, while section 7 concludes and summarizes the document.

2 ACINO Architecture Overview

The ACINO concept allows applications to explicitly specify requirements for the network services they need, in terms of high-level parameters, such as maximum latency or reliability, without delving into the details of the underlying technologies. Such parameters have been described in literature as primitives or intents; in this document, we use the following definitions:

- Primitive: a parameter that expresses a service feature, e.g. network object or constraint;
- Intent: a combination of desired primitives that corresponds to the desired characteristics of a useful service.

In other words, a “primitive” is a feature of the underlying network(s) that the ACINO orchestrator is able to handle, and an “intent” a combination of primitives that a client application selects to describe the desired characteristics of a service.

At a high level, the ACINO architecture is a logically centralized **Orchestrator**, outlined in Fig. 1, with a primary function of translating the application-level service requirements into appropriate configuration requests for both the underlying IP/MPLS and Optical network layers. This involves several sub-processes, including: maintaining a multi-layer model of the controlled network, using it to determine which resources are available to serve new connections and selecting resources that satisfy application-level constraints (expressed by means of “intents”), which include instantiating additional connections in the supporting optical layer if needed. Furthermore, the orchestrator houses an online planning tool, which allows the simulation of the effect of various changes in the network, such as failures, new services and network re-optimizations.

The orchestrator interfaces with the outside world through two main interfaces. A **North-Bound Interface** (NBI) towards applications gives them the possibility to request network services with specific requirements, such as latency, reliability and capacity. Special applications, such as a Network Management System (NMS), may even interface with the online planning functionalities exposed by the orchestrator. Application intents provided over the NBI are then translated into a multi-layer service configuration utilizing both IP/MPLS and optical resources, which is pushed down to the underlying network layers through the use of **South-Bound Interfaces** (SBI) exposed by the control planes of the optical and the IP networks. The SBIs are used to push configuration and pull network state or relevant alarm states from the networks being controlled.

The rest of this section briefly outlines the major components of this architecture, as seen from outside; a more detailed representation of the internal modules is presented in Section 6.

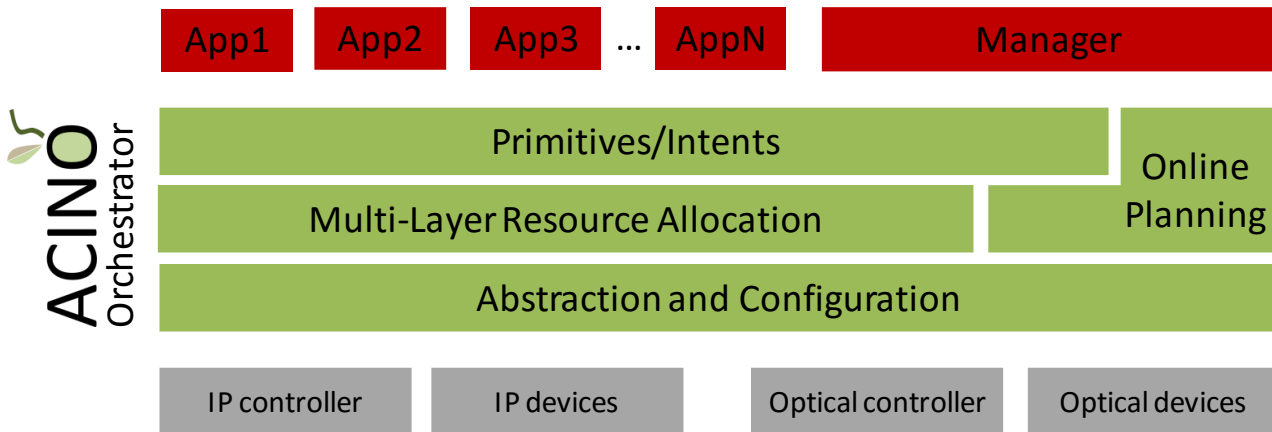


Fig. 1: Simplified schema of the ACINO orchestrator.

Orchestrator

The ACINO orchestrator is a platform that performs the following functions:

- Gather information on the underlying network layers using the SBI and stores it into a multi-layer network model;
- Receive service requests from applications through the NBI in the form of “intents”, maps these intents into installable configuration for the underlying networks, optimizing it with respect to both the explicit service requirements (which must be satisfied, if at all possible) and existing configuration (re-using pre-established lightpaths and Traffic Engineering tunnels whenever possible), and pushes such configuration down to the underlying network control planes via the SBI;
- Receive planning requests from one or more Network Management System class application through the NBI, computes the effects of the proposed changes and replies back to the requestor with the outcomes of the computation; optionally, it may also push the resulting configuration to the underlying network layer via the SBI.

North-Bound Interface (NBI)

The North-Bound intent-based Interface is split into two main functions: firstly, it allows a “network-aware” application to request a network service (i.e., connectivity) by explicitly specifying both standard parameters, such as ingress and egress points, and also several metrics that affect the quality and type of service it perceives, capacity (bandwidth), reliability, maximum latency and more.

Secondly, it allows the interfacing of an application such as a Network Management System (NMS) with an internal online planning tool. This tool allows to perform what-if analyses of possible changes in the configuration of the underlying networks in a dynamic fashion, and to re-optimize the current configuration in order to, e.g., minimize a cost or performance metric. These features are only given to an NMS application because it can be assumed to be owned and operated by the same stakeholders who own the underlying networks being managed.

Additional details of the syntax and processes implementing the intent processing will be included in D3.2.

South-Bound Interfaces (SBIs)

The south-bound interfaces are used by the orchestrator to interface with the control planes of the underlying IP/MPLS and Optical layers. These could take several forms:

- Software Defined Networking (SDN) Controllers exposing a custom REST interface or standard protocol and model (e.g. OpenFlow or NETCONF/YANG);
- Direct interfaces to configure each device (e.g. via CLI, OpenFlow or NETCONF/YANG);
- Service interfaces in each device (e.g. PCEP, UNI), which, unlike an SDN controller, are exposed by each border device (and can typically only instantiate services starting from that device).

Irrespective of the exact technology used, the purpose of this interface is to enable both fetching topological information from the underlying layers, and sending configuration commands to (eventually) the underlying devices.

3 ACINO Orchestrator Requirements

This section describes the requirements on the three main elements (i.e., Orchestrator, SBI and NBI) of the ACINO architecture. Additionally, it maps primitive/intent requirements with the Case Studies defined in deliverable D2.1 [ACID21] (using the notation CS4.X, where X is the index of the case study in D2.1)³.

3.1 Requirements of the Northbound Interface

The North-Bound Interface is split in two logically different set of services: the high-level dynamic intent-driven service management interface and the more detailed management interface towards a multi-layer planning tool. The former allows applications to provision and manage connectivity services using high-level service descriptors, while the latter exposes additional topological details (such as information from the optical layer) and to interface with an integrated online planning tool.

3.1.1 Dynamic Intent-driven Service Management Interface

[NB-INT-1] The ACINO orchestrator shall allow an application to request connectivity for a particular host from a specified network ingress point.

[NB-INT-2] The ACINO orchestrator shall allow applications to specify any combination of the following constraints:

- a. Bandwidth (capacity), potentially variable [CS 4.1, 4.2, 4.3, 4.5, 4.6];
- b. Maximum latency, or latency class [CS 4.1, 4.2, 4.3, 4.5];
- c. Cost (monetary) [CS 4.1, 4.5];
- d. Security (some services may be encrypted) [UC 4.4, potentially UC 4.1, 4.2, 4.5];
- e. Reliability, expressed as survivable downtime: none, low, high, don't care (possibly a numeric threshold) [CS 4.3, potentially UC 4.1, 4.2, 4.5].
- f. Calendaring, expressed as time of activation, deactivation and potentially recurrence [CS 4.1, 4.2, 4.6, potentially CS 4.5]. Optionally, in conjunction with a Cost limit, this option could be interpreted as "provide a service when the cost is below a certain threshold".

[NB-INT-3] An application must be able to learn or be told the network ingress (and possibly egress) points of the network(s) managed by the orchestrator.

[NB-INT-4] An application will be able to poll the status (e.g. active, scheduled, terminated, refused) of a service it has requested.

[NB-INT-5] An application will be able to change some parameters of active/scheduled services, such as bandwidth. The orchestrator may reject the changes if resources are insufficient to enact them.

³ Please note that these requirements may be modified during the implementation phase.

[NB-INT-6] An application will be able to terminate or un-schedule services it has requested.

3.1.2 Multi-layer Topology and Planning Interface

[NB-PLAN-1] The orchestrator shall allow an external application/user to request the built-in online planning tool to:

- Simulate the effect of failures, i.e., observe the effect of a node or link failure on existing traffic and check which application requirements would no longer be fulfilled;
- Simulate the effect of changes to the physical (optical) or virtual (IP/MPLS) topology, i.e. check how traffic would be rerouted by installing new fibers or lightpaths; for the latter, the tool shall also be able to install the new configuration into the network.
- Compute a re-optimized configuration of the network, and later install it.

3.2 Requirements of the Southbound Interface

The South-Bound Interface has two instances: one for the control plane of the IP/MPLS network, and one for that of the Optical network. Both are split in three sub-functions: (1) Provisioning, (2) Topology Discovery, and (3) Monitoring. Furthermore, these requirements generally apply to both Wavelength Division Multiplexing (WDM)⁴ and Optical Transport Network (OTN) systems, with the obvious omission of feasibility limitations that only apply to the WDM layer. Requirements shared by both instances are outlined here, while those specific to either the IP or Optical interface are described in the following subsections.

[SB-1] Optionally, an underlying controller may inform the ACINO orchestrator that it can provide a different connection to the one requested.

3.2.1 IP Controller Interface

3.2.1.1 Provisioning Requirements

[IP-PROV-1] The IP/MPLS layer Control Plane shall enable to configure an IP link by defining the salient attributes for the link and the router ports at both ends (e.g., IP address, IGP metric, link bundling properties).

[IP-PROV-2] The IP/MPLS layer Control Plane must be able to configure explicit tunnels (e.g., Label Switched Paths).

[IP-PROV-3] The IP/MPLS layer Control Plane shall provide a list of active core ports for each router and the remote peer for each connection between routers.

[IP-PROV-4] The IP/MPLS layer Control Plane shall provide a list of spare core router ports: inactive core ports for each router that are available for use but not currently connected to a peer.

⁴ Actually, any transparent optical switching layer, including flexi-grid.

3.2.1.2 Topology Discovery Requirements

- [IP-TD-1] The IP/MPLS layer Control Plane shall provide the IP layer topology, including the set of core and edge routers (with their names and loopback addresses), existing adjacencies between routers in the core and edge of the network, and, possibly, their association with geographical sites.
- [IP-TD-2] The IP/MPLS layer Control Plane shall provide the Interior Gateway protocol (IGP, e.g. OSPF or IS-IS) metric characterizing the existing adjacencies.
- [IP-TD-3] The IP/MPLS layer Control Plane shall provide the MPLS configuration and Label Switched Path (LSP) tunnels routing information.
- [IP-TD-4] The IP/MPLS layer Control Plane shall collate and provide the End-To-End (E2E) IP/MPLS layer traffic matrix across the core (from edge router to edge router), including traffic per LSP tunnel.

3.2.1.3 Monitoring and maintenance Requirements

- [IP-MON-1] The IP/MPLS Control Plane must propagate to the ACINO orchestrator alarms regarding link and equipment failures.
- [IP-MON-2] The IP/MPLS Control Plane must allow the ACINO orchestrator to modify the cost of a link (either as a specialized operation or by changing its IGP metric).
- [IP-MON-3] The IP/MPLS Control Plane should support multiple parallel requests, at least at the interface level.

3.2.2 Optical Controller Interface

3.2.2.1 Provisioning Requirements

- [Op-PROV-1] The optical Control Plane shall be able to set up an optical path by defining its end points, facility type (Ethernet, OTN, etc.) and the desired bandwidth if the transceivers are not fixed.
- [Op-PROV-2] The optical Control Plane shall be able to set up a loose/strict explicit optical path between two transceivers, either through an Explicit-Route-Object (ERO) like structure, or via an Exclude list of Shared Risk Link Groups (SRLGs) or Shared Risk Node Groups (SRNGs) to avoid.
- [Op-PROV-3] If a setup fails, the Control Plane shall return the reason for the failure.
- [Op-PROV-4] The optical Control Plane shall be able to provision a restoration path for a given working path. Like the working path, this path can also be either explicit, e.g. using an ERO, or implicit with constraints, e.g. using an eXclude Route Object (XRO). The optical layer will check feasibility of the restoration path and return an error if it is infeasible. The capacity of the restoration path may be shared with other restoration paths as long as their respective working paths are diverse.

3.2.2.2 Topology Discovery Requirements

- [OP-TD-1] The optical Control Plane shall expose the set of optical nodes, the fiber links between them and the latency (using time or distance) to cross them.

- [OP-TD-2] The optical Control Plane shall expose the type of each optical node (e.g. ROADM, transceiver, regenerator, etc.).
- [OP-TD-3] Common SRLGs in a Reconfigurable Optical Add Drop Multiplexer (ROADM) must also be provided.
- [OP-TD-4] The optical connections and their properties must be exposed. The properties must contain the routing of each optical connection in the optical topology, its wavelength/frequency slots, and regenerators used for it (if any).
- [OP-TD-5] The optical Control Plane shall either offer a service to perform optical validations of potential new optical circuits, or expose to the ACINO orchestrator all the necessary information to do so in its stead.
- [OP-TD-6] The optical Control Plane shall expose the latency or length of the links within its topology.
- [OP-TD-7] Whenever an optical circuit is subjected to Optical Restoration, the optical control plane should expose the actual re-routed path of restored optical connections to the ACINO Orchestrator.

3.2.2.3 Monitoring Requirements

- [OP-MON-1] The optical Control Plane shall be able to provide alarms for link, node or optical connection failure to the ACINO orchestrator.
- [OP-MON-2] The optical controller should support multiple parallel requests, at least at the interface level.

3.3 Requirements of the ACINO Orchestrator

At a high level, the ACINO orchestrator must implement the functionalities exposed by the north-bound interface, by leveraging the services and information exposed by the south-bound interface. More in detail:

- [ORC-1]The orchestrator shall be able to compute multi-layer solutions to service requests received through the NBI, whose constraints are expressed in the form of intents. This includes:
- a. Computing new tunnels in the IP/MPLS layer, including backup routes.
 - b. Computing new circuits (optionally protected or restorable) in the Optical layer, taking feasibility, SRLGs, and available resources into account.
- [ORC-2]The orchestrator shall be able to re-optimize the underlying networks according to some multi-layer optimization metric.
- [ORC-3]The orchestrator shall be able to simulate, over the topology and traffic matrix learned from the SBI, the impact of a failure or routing change.
- [ORC-4]The orchestrator shall ensure consistency between multiple requests to the underlying controller(s); if one or more requests from a batch cannot be served, the orchestrator shall remove the successfully instantiated connections since the overall demand is not satisfied (multiple attempts

may internally be made, but the end result must be binary: either all are accepted or all are removed).

[ORC-5]The orchestrator shall be able to handle concurrent requests.

[ORC-6]Optionally, the orchestrator shall support subjecting incoming requests to policy rules filtering allowed operations on a per-user basis.

4 State of the Art of IP and Optical Control

This chapter gives an overview of existing SDN frameworks and describes the current State of the Art for controllers of Optical and IP/MPLS Networks.

4.1 Overview of SDN frameworks

This section lists most known SDN frameworks, which can be used as a base to develop the ACINO orchestrator or the controllers for the IP/MPLS or Optical networks. To the best of our knowledge, the most comprehensive survey of available SDN frameworks is presented in [KRE14], upon which a large part of this section is based. According to [KRE14], there exist a large number of SDN frameworks, which differ in their support of different types of north- and south-bound APIs, different languages (C, Python, Java, etc.), support for consistency checks and fault tolerance, licensing scheme, and performances/scalability. Some of these details are recorded in Table 1, which is taken from [KRE14].

Table 1: Overview of most known SDN frameworks (source: [KRE14]).

Name	Architecture	Northbound API	Consistency	Faults	License	Prog. Language	Version
Beacon	centralized multi-threaded	ad-hoc API	No	no	GPLv2	Java	v1.0
DISCO	distributed	REST	—	yes	—	Java	v1.1
Fleet	distributed	ad-hoc	No	no	—	—	v1.0
Floodlight	centralized multi-threaded	RESTful API	No	no	Apache 2.0	Java	v1.1
HP VAN SDN	distributed	RESTful API	weak	yes	—	Java	v1.0
HyperFlow	distributed	—	weak	yes	—	C++	v1.0
Kandoo	hierarchically distributed	—	no	no	—	C, C++, Python	v1.0
Onix	distributed	NVP NBAPI	weak, strong	yes	commercial	Python, C	v1.0
Maestro	centralized multi-threaded	ad-hoc API	no	no	LGPLv2.1	Java	v1.0
Meridian	centralized multi-threaded	extensible API	no	no	—	Java	v1.0
MobileFlow	—	SDMN API	—	—	—	—	v1.2
MuL	centralized multi-threaded	multi-level interface	no	no	GPLv2	C	v1.0
NOX	centralized	ad-hoc API	no	no	GPLv3	C++	v1.0
NOX-MT	centralized multi-threaded	ad-hoc API	no	no	GPLv3	C++	v1.0
NVP Controller	distributed	—	—	—	commercial	—	—
OpenContrail	—	REST API	no	no	Apache 2.0	Python,	v1.0

Name	Architecture	Northbound API	Consistency	Faults	License	Prog. Language	Version
						C++, Java	
OpenDaylight	distributed	REST, RESTCONF, ad-hoc API	weak	no	EPL v1.0	Java	v1.0
ONOS	distributed	RESTful API, ad-hoc API	weak, strong	yes	—	Java	v1.0
PANE	distributed	PANE API	yes	—	—	—	—
POX	centralized	ad-hoc API	no	no	GPLv3	Python	v1.0
ProgrammableFlow	centralized	—	—	—	—	C	v1.3
Rosemary	centralized	ad-hoc	—	—	—	—	v1.0
Ryu NOS	centralized multi-threaded	REST, ad-hoc API	no	no	Apache 2.0	Python	v1.0
SMaRtLight	distributed	RESTful API	no	no	Apache	Java	v1.0
SNAC	centralized	ad-hoc API	no	no	GPL	C++	v1.0
Trema	centralized multi-threaded	ad-hoc API	no	no	GPLv2	C, Ruby	v1.0
Unified Controller	—	REST API	—	—	commercial	—	v1.0
yanc	distributed	file system	—	—	—	—	—

Since ACINO is a research project, scalability and performances are a concern but not the primary drivers in choosing the most appropriate framework on top of which to develop the ACINO orchestrator. Greater weight must be given to available features, ease of extensions, adherence to available standards, open-source license. Finally, in order to maximize the impact of the ACINO project, the overall preferences of the target research and vendor communities are also taken into consideration.

Based on this analysis, three of these frameworks emerge as the most promising for supporting the ACINO concept: OpenDayLight, ONOS and ODENOS. Additionally, the ABNO architecture, while mixing SDN with older (e.g., PCE) concepts, deserves further consideration.

4.1.1 ABNO

The Netphony ABNO [ABNO] implementation is based on the IETF Application-Based Network Operations (ABNO) architecture. The purpose of this architecture is to facilitate multiple different implementations of key components, while offering interoperability between them, and simple interaction with the applications and with the network devices. An implementation of this architecture may make several important decisions about the functional components. Multiple functional components may be grouped together into one software component. For example, an Active, Stateful PCE could be implemented as a single server combining the ABNO components of the PCE, the Traffic Engineering Database, the Label Switched Path Database, and the Provisioning Manager. Alternatively, these components could be distributed across separate processes, or there could even be multiple ABNO Controllers, each supporting different classes of applications or application services.

To better understand the relationships between these components, Fig. 2 presents some of their common interactions.

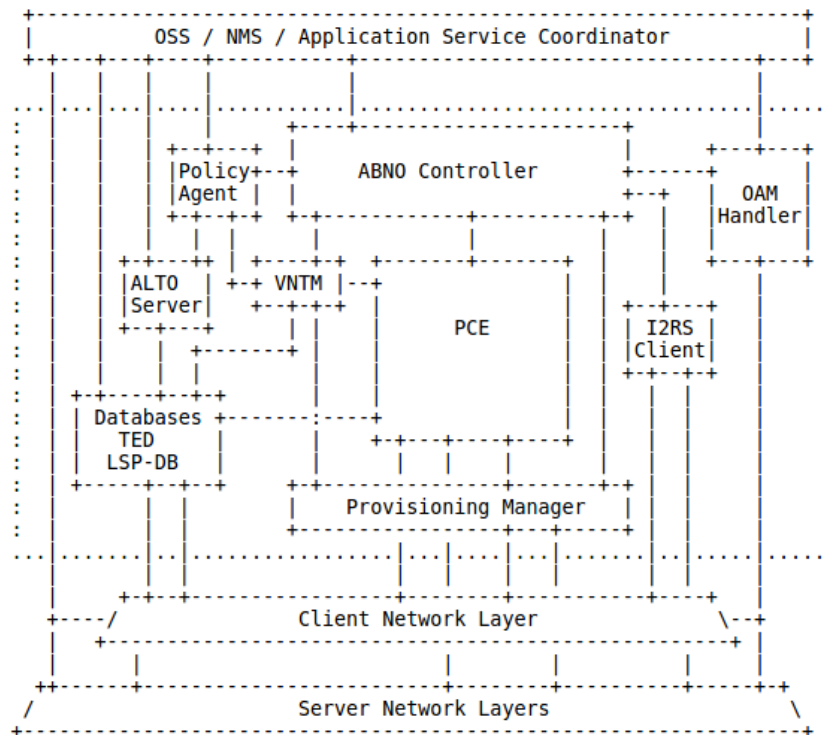


Fig. 2 : Generic ABNO Architecture

A Network Management System (NMS) or an Operations Support System (OSS) can be used to control, operate, and manage a network. For example, if the operator wants to provision an End-to-End path in the network that the ABNO orchestrator controls, a PCEP communication between ABNO and a PCE takes place. With the information obtained from the PCE, the ABNO controller requests to the Provisioning Manager (PM) to perform the provisioning in the client network layer domains.

The main components of the ABNO architecture, as prescribed in [RFC7491], are the following:

- The ABNO controller is the main component of the architecture and is responsible for orchestrating the workflows and invokes the necessary components in the right order. ABNO stores a repository of workflows with operations in the network (MPLS provisioning, L3VPNs, etc.).
- The PCE is the unit that handles the path computation across the network graph. If the PCE is active, it can directly operate in the nodes to create LSPs. Coordination between multiple PCEs operating on different TEDs is also required to perform path computation in multidomain (for example, inter-AS) or multilayer networks.
- The virtual network topology manager (VNTM) is a key element for multilayer scenarios. The VNTM is in charge of maintaining the topology of the upper layer by connections in the lower layer. This entity simplifies the upper-layer routing and traffic engineering decisions as it hides the optical connections set up by the LSP.

- The topology module (TM) has multiple databases: a view of each layer, an interlayer relation, and an inventory DB with the configuration parameters of each of the resources. One part of the module is devoted to obtaining and maintaining up to date the topology from the available sources (e.g., routing protocols, OF controllers). The other part of the module is devoted to providing the information to the requesting parties such as the provisioning manager (PM), PCE, or VTNM.
- The PM is the unit in charge of configuring the network elements. It can do so by configuring the resources in each node (CLI, OF, or NetConf) or by triggering a set of actions to the control plane via PCEP.

The ABNO architecture was designed to provide the following services to applications: optimization of traffic flows between applications to create an overlay network, remote control of network components allowing coordinated programming of network resources, interconnection of Content Delivery Networks, network resource coordination to automate provisioning and to facilitate traffic grooming and regrooming, and Virtual Private Network (VPN) planning.

4.1.2 OpenDayLight

OpenDayLight (ODL) [ODL] is “an open platform for network programmability to enable SDN and create a solid foundation for NFV for networks at any size and scale”. It is one of the, if not the, largest open-source SDN controller projects, drawing support from a large number of companies (including Cisco, Intel, and ADVA), which all together make up the bulk of the important names in ICT and networking.

According to [ODL], ODL supports features such as providing on-demand services that may be controlled by the end user or the service provider, agile service delivery on cloud infrastructure, dynamically optimizing the network based on load and state, highly agile automation of networks for universities, cities and metropolitan areas, and centralized administration of the network and/or multiple controllers.

On the technical side, ODL is a Java-OSGI based framework, whose architecture (see Fig. 3) is built around a Service Abstraction Layer (SAL). Below the SAL, multiple plugins implement a host of southbound protocols (OpenFlow, BGP-LS, LSIP, SNMP, NETCONF, etc.⁵) and translate them to an internal model that is suitable for consumption by a number of inter-dependent OSGI modules running on top of the SAL, which implement the logic behind REST (and, optionally, Java or RESTCONF) APIs exposed through the controller’s north-bound interface.

The main use case for which standard ODL is being developed in controlling a “traditional” layer 2 SDN packet network made up of OpenFlow switches. This is the functionality supported by the main modules included with the framework source code. These may or may not be useful for developing the ACINO orchestrator, however ODL also provides facilities to write custom models for the SAL (therefore expanding the internal model) and implement custom modules, as well as providing a number of pre-implemented southbound protocols, including HTTP for ease of interface with other controllers implementing a REST north-bound interface.

⁵ Not all of them are fully implemented yet.

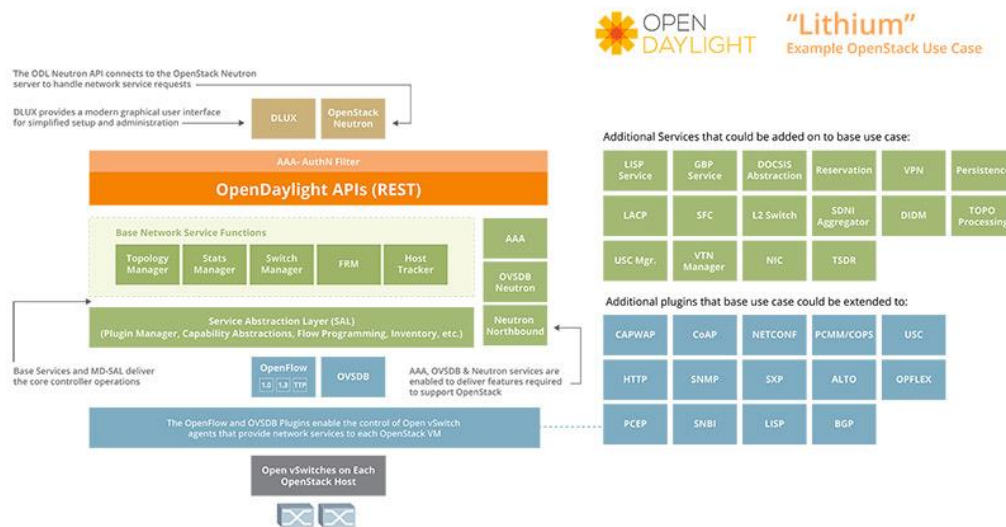


Fig. 3: High-level Architecture of OpenDayLight [ODL].

For the above reasons, ODL is a promising platform on top of which to develop the ACINO orchestrator.

4.1.3 ONOS

ONOS [ONOS], or Open Networking Operating System, is another large open-source effort to develop an SDN controller supported by a number of large companies, including Cisco, Huawei and Ciena.

Like ODL, it is based on a Java-OSGI framework, with network-facing south-bound modules abstracting away unnecessary details and presenting a uniform view to higher layers, as depicted in Fig. 4. Its code is designed to be modular, configurable, protocol-agnostic, and expandable.

Unlike ODL, ONOS southbound “providers” translate to a very specific internal representation, whose goal is to eventually become an all-encompassing network model. Like ODL, the current model is mainly focused on L2 switches running OpenFlow, but some support for IP and very simple Optical nodes is already present. Like ODL, multiple south-bound protocols can be used, with several already implemented and the possibility to include more.

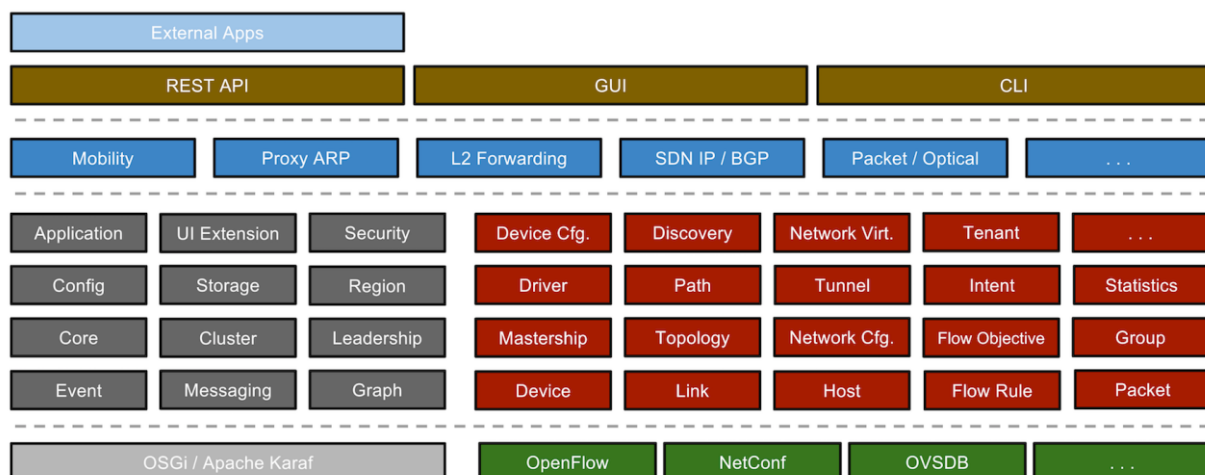


Fig. 4: High-level Architecture of ONOS [ONOS].

ONOS is, furthermore, built as a distributed SDN operating system, which allows greater fault-tolerance, scalability and resilience. Specifically, it has built-in support for the replication and cooperative handling of its internal model of the controlled network.

Additionally, ONOS already implements an Intent Framework, which allows controlling the network in form of policy rather than instruction. The inbound intents are translated via compilation into potentially multiple installable intents, representing network services, which in turn can consist of multiple installable low-level network operations (e.g., OpenFlow flows rules, device configuration, optical wavelength reservation...) on possibly multiple network elements. The framework is extensible by design, allowing adding new intents, compilers or installers, which could be used to implement the high-level primitives that characterize the ACINO application-centric approach.

4.1.4 ODENOS

ODENOS is a network orchestration framework developed within the O3 project [ODENOS], designed to control multi-layer, multi-domain or multi-vendor networks. It is based on an extensible object-oriented network model which includes nodes, ports, links, flows and packets, and abstract network operators such as slicing (multiplexing), aggregation (abstraction) and link-layerization (turning set of nodes into networks).

Public documentation for ODENOS is currently severely limited. Its high-level architecture is depicted in Fig. 5.

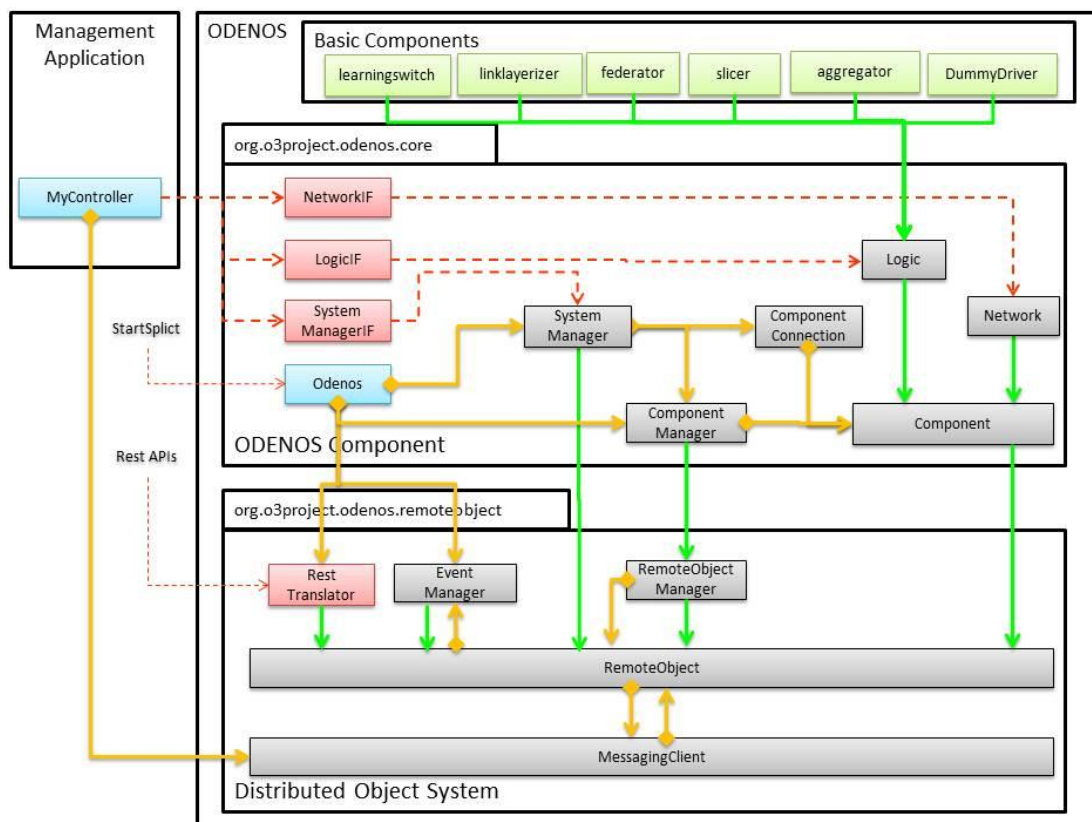


Fig. 5: High-level Architecture of ODENOS [ODENOS].

4.1.5 Summary of the most promising SDN Frameworks

Table 2: Summary of the features of the three most promising SDN frameworks: OpenDayLight, ONOS and ODENOS.

	Feature	OpenDayLight	ONOS	ODENOS	ABNO
Features relevant to ACINO	Intent-based northbound interface	No, just REST/RESTCONF APIs	Yes		No
	Multi-layer network model	No, to be developed	Yes (limited to WDM)	Yes	Yes
	Extensible southbound interface	Yes	Yes		Yes
	Code Documentation	Some, but mostly outdated	Good	Poor	Poor
	Extensibility of the framework	Somewhat complex, but allowed by design	Easy, by design		Modular
	Open Source	Yes	Yes	Yes	Yes
	Network emulation	Mininet, mainly OF switches	Mininet, support for BGP speaker, IP and Optical nodes	Mininet, mainly OF switches. It is possible to add Ryu-based optical nodes	Includes GMPLS emulator. Demonstrated with OF controllers
Architectural Features	Available southbound protocols	OpenFlow 1.0 and 1.3, OpenFlow TTP, OVSD, NETCONF, PCEP, BGP, OPFLEX, SNMP	OpenFlow 1.0 and 1.3, NetConf, OVSD, PCEP	OpenFlow	PCEP, GMPLS, interfaces to some OF controllers (RYU, NOX, POX, ODL), NETCONF
	Available basic functions	Topology manager, Inventory manager, Statistics manager, L2 switch, Host tracker, Flow Rules Manager, ...	topology, inventory, statistics, flow management, intent management		Topology, Traffic Engineering
	Available Northbound protocols	RESTCONF	REST, JAVA API		REST API and RESTCONF (COP)
	Standard northbound information models	YANG models (no standard currently available)			COP YANG models
	Support external/internal SDN applications	External apps via REST APIs, internal apps via Java APIs	External apps via REST APIs, internal apps via Java APIs	Via REST APIs	External apps via REST APIs, internal apps via Java APIs

	Feature	OpenDayLight	ONOS	ODENOS	ABNO
Software Characteristics	Deployment models		stand-alone, clustering	Multi-vendor SDN controller, Hierarchical SDN	Multi-vendor SDN controller, Hierarchical SDN, Multi-layer SDN controller
	Cloud Integration	Openstack Neutron	Openstack	No	Openstack Neutron Plugin
	Latest version	Lithium, June 2015	1.4.0, Dec. 18 2015	1.0 / Feb 19, 2015	Nov 4, 2015
	Controller Language	JAVA	JAVA	Java, Python and Ruby	JAVA
	Application Language	JAVA or over REST API	JAVA or over REST API		JAVA or any
	Controller License	Eclipse Public License	Apache 2.0		GNU AFFERO
	Application License	Eclipse Public License	Apache 2.0		GNU AFFERO
	Documentation	https://wiki.opendaylight.org/view/Main_Page	https://wiki.onosproject.org/display/ONOS/Wiki+Home	https://github.com/o3project/odenos/blob/develop/doc/api/index.md	https://github.com/telefonicaid/netphony-abno/wiki
	Source code distribution		https://github.com/opennetworkinglab/onos	https://github.com/o3project/odenos	https://github.com/telefonicaid/netphonyabno/wiki/Downloads
	Package distribution	https://www.opendaylight.org/downloads	https://wiki.onosproject.org/display/ONOS/Downloads		https://github.com/telefonicaid/netphonyabno/wiki/Downloads
	System requirements	https://github.com/opendaylight/integration	Stand-alone: 2CPUs, 4GB RAM, 20GB disk. Clustering: from 3 to N servers (can be virtual) with at least 2CPUs, 4GB RAM, 20GB disk		
	Main system technologies	maven, OSGi, Karaf	maven, OSGi, Karaf, JAVA8	maven	maven, JAVA7
	Membership required	No	No	No	No
	Community size and popularity	High	Small/medium community, high popularity	Small	Small
	Major industrial partners involved	Brocade, Cisco, Citrix, Dell, Ericsson, HP, Intel, RedHat, NEC	AT&T, Intel, NEC, Cisco, Ciena, Ericsson, Huawei, NTT, Fujitsu, China Unicom, SK telecom	NEC, NTT, Fujitsu, Hitachi	Telefonica

Feature	OpenDayLight	ONOS	ODENOS	ABNO
Members of the consortium that can act as a bridge towards the community		CREATE-NET		Telefonica
Adoption in other projects	LIGHTNESS, CONTENT, COSIGN, INSPACE, NetIDE, many others	NetIDE, GEANT in EU, several projects and use-cases in the US		Idealist, STRAUSS, XIFI, STRONGEST, 5GEx, XHAUL

4.1.6 Selected framework for the development of the ACINO Orchestrator

The previous sub-sections described several existing SDN frameworks. Among these, we have identified ONOS to be the most suitable framework for serving as the basis of the implementation of the ACINO Orchestrator, due to the following reasons.

Features: while all major SDN frameworks share a modular design (based on Java-OSGI, except for ABNO), the ONOS structure additionally already implements some functionalities that can be leveraged to implement the ACINO orchestrator, namely:

- an intent-based API and framework for connection requests;
- an abstract topology model which can support both packet and optical networks (at least basic cases);
- a logically centralized yet (optionally) physically distributed Control Plane, with built-in fault-tolerance and scalability features.

Community: like ODL, ONOS is supported by a vast community of research and industrial entities (ODENOS and ABNO are supported by comparatively smaller communities). This is important to give the ACINO project high visibility and impact. Based on initial discussions with the ONOS community, it would be possible for the ACINO project to contribute code developed as a part of the project within the mainline ONOS distribution.

Development community: unlike ODL, the ONOS development community, at least insofar as the Master code branch is concerned, is structured and makes use of procedures like code reviews to ensure that only high-quality and reasonably documented code that conforms to the overall project vision is accepted into the core. This makes working with existing code much simpler than with ODL, where the available documentation is severely lacking. ODENOS and ABNO, while open-source, do not appear to have large development communities backing them.

4.2 Optical Controllers

Based on application/network demands, optical controllers are required to provision optical paths in a network, with the option of enforcing additional configuration parameters such as strict/loose path definitions, Shared Risk Link Group (SLRG) constraints, and restoration or protection requirements, among others. This, in turn, mandates that optical controllers have the ability to discover, and possibly present, detailed topology information for the optical network. Optical controllers must also monitor network infrastructure and provisioned services, and provide a mechanism to expose this information.

A detailed list of requirements for an optical controller to interoperate with the ACINO orchestrator is presented in Section 3.2.2. Most optical networks and their controllers support these capabilities; however, unlike IP and Ethernet transport networks, where a single controller architecture can potentially manage infrastructure across multiple vendors, configuration of optical infrastructure is limited to vendor specific controllers due to the significantly higher configuration complexity associated with it. As a result, unlike other technologies, where “open” interfaces are available on network elements, optical network controllers expose such open interfaces as their northbound interface, and act as a mediation layer between requests coming in from SDN controllers and the complex optical infrastructure.

Therefore, the capabilities exposed by an optical controller depend significantly on the choice of the protocols used for the northbound interface. We now present a number of these protocols, focusing on the limitations that they impose on the control of the optical layer, before giving some details of the optical hypervisor from ADVA, which is expected to be the controller that will be used for implementation activities.

4.2.1 North-bound Protocols for Optical Controllers

4.2.1.1 *OpenFlow Protocol*

The simplicity of the OpenFlow (OF) protocol [OF] and its large-scale adoption make it an appealing choice for configuring multi-layer networks. However, the protocol in its original version was not suited for configuration of optical networks.

OpenFlow was designed to configure individual network elements on a per-flow basis, where a flow identifies a set of packets sharing certain header values. It does not have primitives to define services that can span multiple network elements. Topology representation and discovery mechanisms in OpenFlow are designed for packet networks, where every port can fundamentally route traffic to every other port in the same network element, which is not the case in optical networks. The lack of service constructs also limits access to other service lifecycle management operations such as protection, restoration and re-routing, which are all widely used at the optical layer.

Typically, configuration of optical networks via OpenFlow is facilitated by exposing an abstract network topology of the optical network via the optical controller. The optical controller intercepts discovery packets generated by the OpenFlow controller and generates appropriate responses to present this topology abstraction to the OpenFlow controller. In order to create a service, the OpenFlow controller installs flow entries on the entities in the abstract topology, and the optical controller implicitly identifies services based on the flows and configures the network accordingly.

Examples of typical topology abstractions used to expose optical networks over OpenFlow include a single switch model, where all “client” ports from all optical devices are presented as individual ports on a single switch, and a flow configuration between these ports is treated as an optical service between them in the real infrastructure. The limitations enforced by the optical infrastructure can however limit the number of transponder pairs between which a service may be established, and this information is not readily available in this abstraction. Virtual link abstractions are designed to partially address this problem: here, different network elements are represented as individual switches in the OpenFlow topology, and virtual links are created between every network element pair where a service can currently be created. The topology information provided clearly indicates the possible paths on which a single service may be established, but does not present details of the shared resources associated with these “virtual” links. As a result, the configuration of a single service can result in the destruction of multiple virtual links, which significantly limits the capabilities of computing and configuring multiple optical services in the network simultaneously.

The Open Networking Foundation, which maintains the OpenFlow standard specification, is working to address some of these limitations, with special focus on the port definitions to better suit the specifications for optical networks. Port specifications were limited to configuration of address parameters at layer 2 (Ethernet) and 3 (IP), and a primitive up/down status configuration. The port properties in OF1.4 are now being extended with limited capability for configuring details relevant to optical networks, such as transponder properties, channel information, impairment awareness etc. Some effort for including these specifications was already presented in the circuit switch extensions for OF1.0 [SAUOF], and starting with OF 1.4.0 new fields have been introduced:

- Optical port mod property `ofp_port_mod_prop_optical` to configure optical ports.
- Optical port stats property `ofp_port_stats_prop_optical` to monitor optical ports.
- Optical port description property `ofp_port_desc_prop_optical` to describe optical port capabilities.

For the purpose of ACINO, OpenFlow is limited in its capability of performing complex operations in optical networks.

4.2.1.2 *NETCONF / RESTCONF Protocols*

NETCONF and RESTCONF are part of the same family of network device management protocols. The difference is the underlying communication paradigm which is Remote Method Invocation (RMI) for NETCONF and REpresentational State Transfer (REST) for RESTCONF. Due to their flexible modelling based on the YANG language, they can be used to manage different types of network setups. The configuration and message exchange depends on the YANG model.

These protocols can be used as the interface between the optical controller and the optical devices, but also as the interface through which the ACINO orchestrator would control the optical network.

The limitations imposed by NETCONF/RESTCONF depend on which YANG model is used. An emerging open-source implementation in this regard is the Control Orchestration Protocol (COP) [COP] being developed within the STRAUSS European project [STRAEU]. COP consists of four YANG Models:

- `Service-topology.yang`: defines the format for exposing and sharing topology information

- `Service-path-computation.yang`: defines a Remote Procedure Call (RPC) request and reply format for path-computations. By using this API the computation of a path can be requested. The reply contains either the computed path (if successful) or no path (if not).
- `Service-call.yang`: defines the format for requesting new services and sharing information about running services and connections.
- `Service-virtual-network.yang`: defines the format for sharing information about virtual networks, i.e. abstracted views.

Some of the features needed for ACINO, like routing matrices, SRLGs, restoration, protection etc. are missing, but could be added by extending the model. Since these models are focused on optical connections, no information about the IP layer is exposed.

4.2.1.3 *PCEP with BGP-LS Protocols*

The IETF Application Based Network Operations (ABNO) architecture [RFC7491] proposes a set of architectural components and protocols that can address limited requirements for fault, configuration, accounting, performance and security (FCAPS) in optical networks. It leverages the Path Computation Element (PCE) infrastructure, learning the state of multiple networks using BGP-LS and instructing nodes to create services using PCEP.

The BGP-LS protocol is designed to expose link state (topology) information to other entities outside the IGP area, and extensions to the current draft specifications can facilitate the exchange of optical parameters associated with the topology via BGP-LS [FLOVIS].

The Path Computation Element Protocol (PCEP) [RFC5440] was designed to facilitate path computation, and extensive ongoing work on developing this protocol has led to the specification of standards to define a wide range of constraints, including capacity, delay, protection mechanisms, SRLGs uniqueness, loose or explicit path specification, node or link exclusion, etc. Extensions to the PCE architecture and protocol specification proposed in [CRAB13] provide the capability for the PCE server to communicate with a client to establish an LSP and modify LSP parameters as deemed necessary.

In the context of optical controllers, the BGP-LS protocol can be used to expose topology information to the SDN controller or orchestrator while the PCEP protocol can provide capabilities for path computation and provisioning. In terms of capabilities, both protocols can cover most requirements for provisioning and topology discovery in the context of optical controllers. The protocols however do not have built-in monitoring capabilities for network infrastructure and can only, potentially, provide state information for established LSPs.

In the context of ACINO, it is possible to use these protocols in conjunction with other protocols such as SNMP to provide monitoring information using established standards.

4.2.2 **ADVA Optical Network Hypervisor**

While the SBI of the ACINO orchestrator is designed to be interoperable with theoretically any controller, we focus on the integration with the ADVA optical network controller as a representative case of technologies employed by most vendors.

The architecture of the ADVA optical network controller is based on a hypervisor concept. It is a software tool that provides an abstraction layer between the hardware (corresponding to the physical resources in the network) and the operating system (mapped onto the concept of the SDN controller). Fig. 6 depicts three layers.

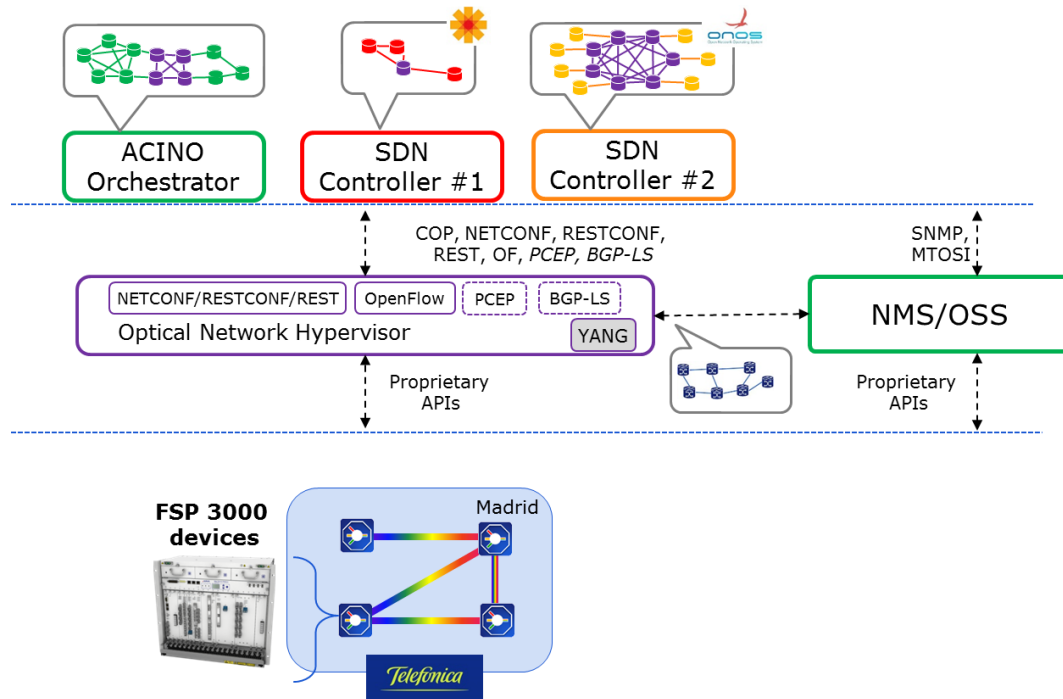


Fig. 6: Control Stack of an optical network running an SDN controller/orchestrator and the ADVA hypervisor.

The bottom layer represents physical resources, i.e. the optical transport network of an operator and the attached hardware of clients. Clients in Fig. 6 are represented by different controller colours. In each client network a subset of the client's network elements called customer edge devices is connected to the edge devices of the operator's transport network. Each client network maintains a number of access points and needs the transport network to interconnect them.

The layer in the middle provides necessary abstraction and isolation between the client's slices as well as well-known Operations, Administration, and Maintenance (OAM) functions, depending on one of two typical mechanisms: The first approach (on the right hand side) utilizes the traditional Network Management System (NMS, and the Operations Support System, OSS). The NMS and OSS perform FCAPS management (fault, configuration, accounting, performance, and security). They expose their functions over a Graphical User Interface (GUI) or Command Line Interface (CLI) as well as standardized interfaces like RESTful interfaces or a Multi-Technology Operations System Interface (MTOSI). The second mechanism is the Optical Network Hypervisor (ONH). It enhances the functionality offered by the NMS by opening interfaces to external client SDN controllers, presenting the abstract view of the topology to hide complexity. The ONH partitions the network in support of multiple tenants still leaving a supervisory view and the overall control to the network's operator. Both the NMS and ONH connect southbound to the physical resources over proprietary interfaces via the data communications network (DCN).

The layer represents the client (tenant) space in which he can use his own SDN controller to manage a slice of the operator’s network. This assumes the use of common and standards based interface between the client’s SDN controller and the ONH. OpenFlow (OF) is one important SDN protocol originally designed for configuring packet-based switches. BGP-LS (Border Gateway Protocol Link State) is an extension of BGP allowing topology exchange based on endpoints and links – both may be virtual due to abstraction. An alternative to BGP-LS is the GMPLS-ENNI protocol, which presents the transport network to the client as an overlay topology made of Virtual TE Links. PCEP (Path Computation Element Protocol) is used for the communication between a PCE (Path Computation Element) and a PCC (Path Computation Client) in the context of path computation requests between endpoints. Recently model-based interfaces (i.e. NETCONF, RESTCONF and various flavours of REST) are also gaining traction. They expose network functionality based on the corresponding YANG data model. This approach gives much more flexibility due to decoupling semantics of the data from the underlying transport protocol.. There are many examples of such models, one of which is COP. Other supported drafts and models include [LIUTE], [LIUAB], [CHETE], [GANTE] and the ODL “TED” and “topology” models.

4.2.2.1 ACINO integration

Two possible solutions for integrating the network hypervisor with the ACINO orchestrator are shown in Fig. 7. The left-most includes a middle man (e.g. an ODL instance) that is used to expose a predefined API. If the capabilities of this mediator layer are not needed a direct connection between the orchestrator and the network hypervisor can be used based on a model driven RESTCONF interface.

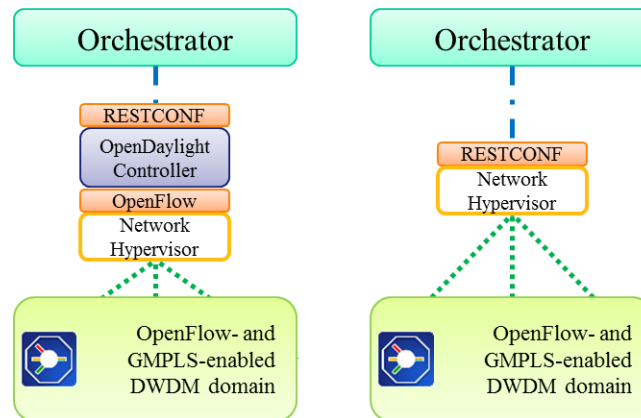


Fig. 7: Possible integration of the ADVA Optical Network Hypervisor with the ACINO Orchestrator.

A more detailed description on the choice of integration strategy and interface specifications between the ADVA controller and the ACINO orchestrator will be provided in D4.4.

4.3 IP Controllers

Given the extreme complexity of managing the distributed control of large IP networks, numerous management tools have been developed to simplify and centralize this task. These include “legacy” planning tools and Network Monitoring Systems, which, unlike SDN controllers, are designed to work with and on top of traditional IP/MPLS routers (using CLI or RSVP-TE to push configuration to the network) and lack programmatic interfaces.

The section presents an overview of the protocols that are available as the northbound interface of an IP/MPLS controller and their limitations. We also present possible solutions, based on SDN controllers or older designs, which can be used by the ACINO orchestrator to manage the IP layer.

4.3.1 North-bound Protocols for IP/MPLS Controllers

4.3.1.1 OpenFlow

The OF protocol, previously discussed in Section 4.2.1.1, was designed to install rules to route IP/Ethernet packets and, from version 1.1, also supports MPLS labels. However, it is generally implemented directly on the network elements, and is used as the south-bound interface of IP controllers. Based on the limited service level scope of the OpenFlow controller, its use as the north-bound protocol, would severely limit the usefulness of such controllers, as the entire network would need to be abstracted as a single switch, which is too coarse for the purposes of ACINO (except for particular implementations, such as in FlowVisor [FLOVIS], which expose multiple OF switches using a single IP address).

4.3.1.2 PCEP + IGP/BGP-LS

PCEP, already described in Section 4.2.1.3, is used, in conjunction with an IGP (OSPF or ISIS) or BGP-LS, to export information to the IP controller from the network elements and instruct them to create new tunnels. The same protocol can also be used to push requests down from the orchestrator and expose topology and potentially monitoring information to the controller.

4.3.1.3 NETCONF

NETCONF (and RESTCONF) was outline in Section 4.2.1.2, and can be used as a communication protocol between the IP/MPLS controller and ACINO orchestrator. However, an appropriate YANG model would be required to do so.

4.3.1.4 REST

Finally, a possible protocol to be used is a non-standard REST interface, as long as it fulfills all the requirements outlined in Section 3.2.1.

4.3.2 Active Stateful PCE / ABNO

Although it predates the SDN era, a stateful Path Computation Element (PCE) [RFC4655][CRAB13] with instantiation capabilities, commonly called an active PCE, can be used to control an IP/MPLS network. Using PCEP [RFC5440], an external controller running a Path Computation Client could conceivably request the establishment, removal or modification of LSPs. PCEP notification messages could, furthermore, be used to push alarms in case of failures or perform other monitoring tasks. Such a solution, however, would be lacking support for topology discovery, which can be remedied by having the overlaying orchestrator listen to the IGP as the PCE does. ABNO, which includes a PCE as an internal component, falls in this category.

4.3.2.1 NorthStar Controller

NorthStar [NORTHS] is advertised as an SDN controller for IP/MPLS from Juniper and has been singled out in this section as one of the few IP-focused controllers currently available. It is based on an active Path Computation Element (PCE), speaking PCEP towards network devices and listening to the local IGP (BGP-LS, OSPF, ISIS) to learn the state of network resources, but exposing a REST-based north-bound API. The cen-

tralized controller performs online/offline LSP computations and can autonomously trigger re-optimizations, using PCEP to instruct head routers to signal an LSP tunnel via RSVP-TE.

The north-bound API exposed by NorthStar is a REST interface accessible via either HTTP or HTTPS, and uses the widely used JSON data format. According to the available documentation, NorthStar functionalities include the following (functionalities that are of no direct interest to the project were omitted for greater clarity):

- Dynamically learn the IP topology and existing LSPs
 - Expose the learned topology in terms of nodes and links, fulfilling [IP-TD-1]
- Provision new LSPs, fulfilling [IP-PROV-1].
 - Supports LSP diversity
 - Support standby LSPs
 - Supports scheduled provisioning

From the available documentation it is not known whether NorthStar supports the required monitoring functions, and this functionality may be implemented via a SNMP polling mechanism for the orchestrator..

4.3.3 ONOS-IP

As described earlier, ONOS is a full-fledged SDN controller, mainly built around OpenFlow switches but supporting other protocols. ONOS implements an IP application that can let an SDN-controlled network appear as a standard autonomous system speaking eBGP, learning networks as advertised by its peers and configuring internal forwarding paths accordingly. The current implementation of this application has a few limitations, including support for IPv4 only.

5 Multi-layer Network Models

The ACINO orchestrator will need to perform routing and resource allocation over both IP/MPLS (packet) and Optical (circuit) network topologies, with the latter acting as a substrate of and supporting the former. For this reason, it requires a multi-layer network topology model that is able to model the details of both these layers, as well as means to enforce its decisions.

Fervent activity has been ongoing in standardization bodies such as the IETF and ONF regarding new YANG-based topology models, many of which are suitable, at least in structure, for multi-layer environments.

From an analysis of the available (public) multi-layer models, two main alternatives emerge and are detailed in the rest of this section: the Core Information Model, supporting the Transport-API, from the ONF, and the topology family of models (generic, TE, L3, L2), to which the OpenConfig initiative is aligned, from the IETF. An analysis of the functionalities supported by these APIs is presented below.

5.1 ONF Core Information Model

The ONF Core Information Model (CIM) [ONFCIM], depicted in Fig. 8⁶, describes multi-layer topologies with an emphasis on lower-layer details and adaptations between layers. It is based on the concept of Forwarding Domains (FDs), which can be abstracted to higher layers through partial or opaque views, and model the connectivity inside that domain. At the lowest level, they represent the connectivity within Network Elements (NEs), for example the connectivity matrix of a ROADM, while at higher layers they may represent e.g. the connections between ROADMs in the optical layer, or the connectivity between IP routers.

At the borders of Forwarding Domains (many of which can be abstracted as a single network element even if unrelated) are Logical Termination Points (LTPs), which model the changes in framing, encoding, etc. between layers. Two or more LTPs are connected via links, potentially with additional structures to model multi-point cases. Several parameter packs have been defined for links.

A multi-layer service in this model is simply a service following the chain of adaptation functions up and down its trail. This makes modeling them natural, but also makes creating TE tunnels impractical. In addition, it requires detailed modeling of the boundaries of different layers, which is outside the scope of the ACINO project.

The characteristics of this network model are summarized in Table 3.

Table 3: Summary of the ONF Network Model.

Good	Bad
Precisely models vertical integration	Difficult to model tunnels and overlays – the actual topology (links) is only at the lowest layer
Multi-layer services come for free - they are mod-	Requires detailed technology-specific adaptations at

⁶ A YANG implementation of which is available in [LAMTE].

eled as trails going up and down the layer stack	each layer – too fine a level of detail for ACINO
Already supports some optical details	

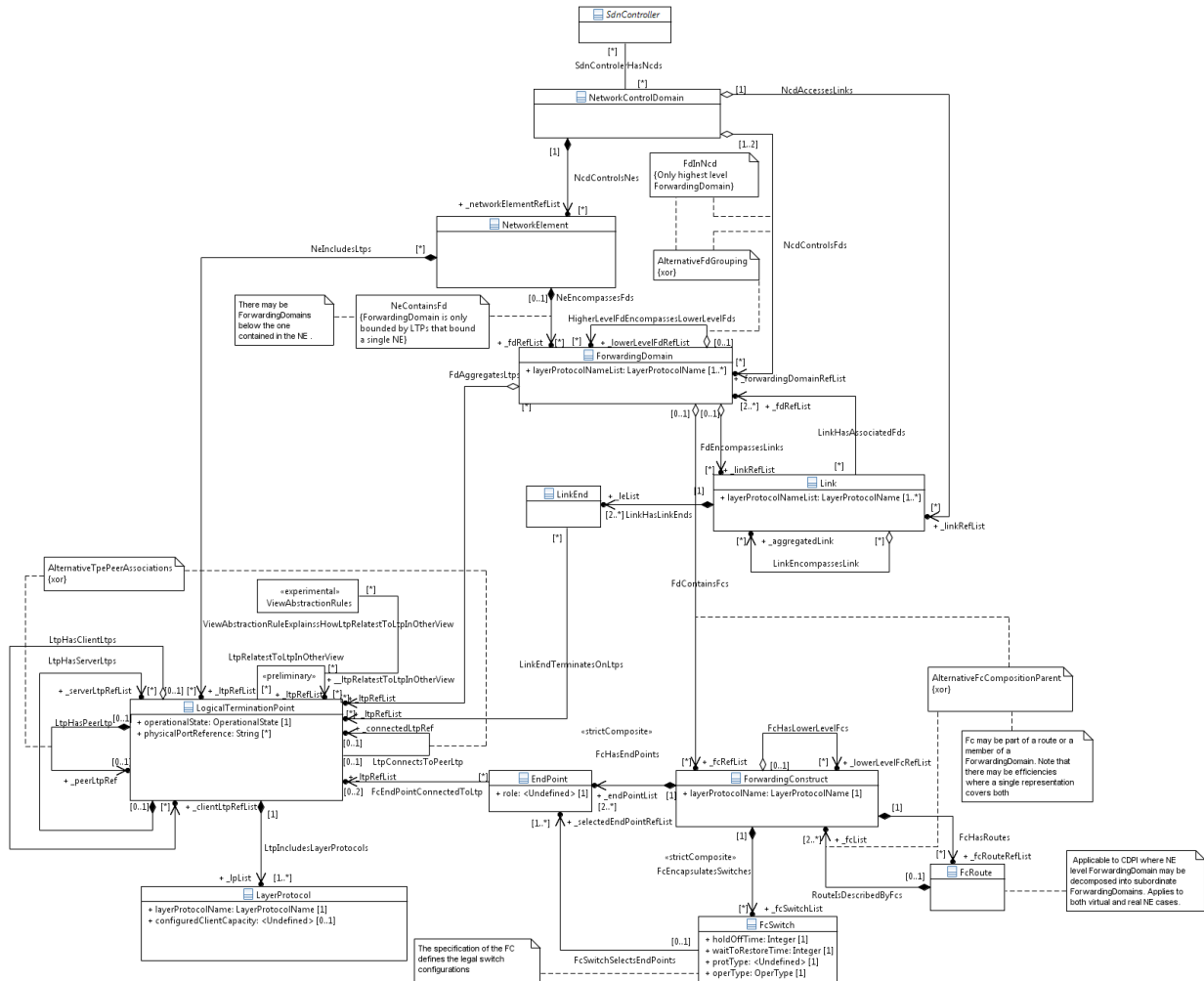


Fig. 8: ONF Core Information Model schema [ONFCIM].

5.1.1 Transport-API

The ONF Transport-API (T-API), of which the COP protocol is a subset, is a proposed interface to create services over then ONF CIM. The requirements of the T-API are described in [TAPIFR]⁷.

Notable features of T-API include:

- separation of connection requests from services implementing them (a key consideration for supporting intents);
- the ability to request connections using a multitude of constraints;

⁷ An initial YANG implementation is available in [LAMTE].

- explicit references to protocols/formats to be used;
- path computation/optimization requests;
- explicit support for virtualized networks.

A notable missing feature, currently marked as “to be decided” in the available documentation, is the notifications of errors or failures through this interface.

5.2 IETF Topology Models

The IETF developed several topology models, mostly augmenting one another to model extra capabilities or details pertinent to specific contexts, as shown in Fig. 9, which are in various stages of progress with respect to properly and accurately modeling the relevant context. The main Abstract Topology model [CLETOPO] is a generic hierarchy of largely independent topologies, each comprising its own nodes, links and termination points (ports), which reference which other equivalent objects sit above or below them in the hierarchy.

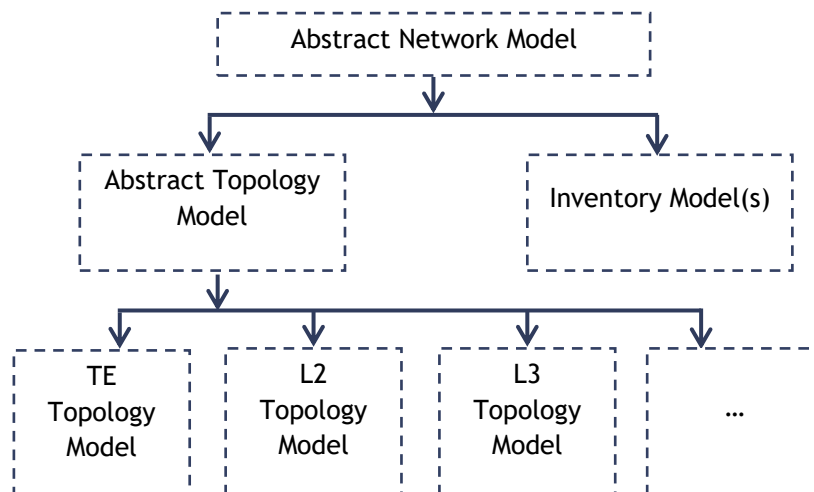


Fig. 9: IETF Model Hierarchy.

More in detail, the abstract network model contains two main objects:

- A list of Networks, each comprising:
 - One or more types, representing its layer or technology
 - References to supporting networks (i.e., underlays)
 - A list of nodes in the network (each of which may reference a supporting node in underlying networks)
- A network state object
 - Mainly specifying whether the data source of the network is internal or external in this module

Such a model allows the creation of hierarchical topologies via the explicit references to underlying objects, as depicted in Fig. 10.

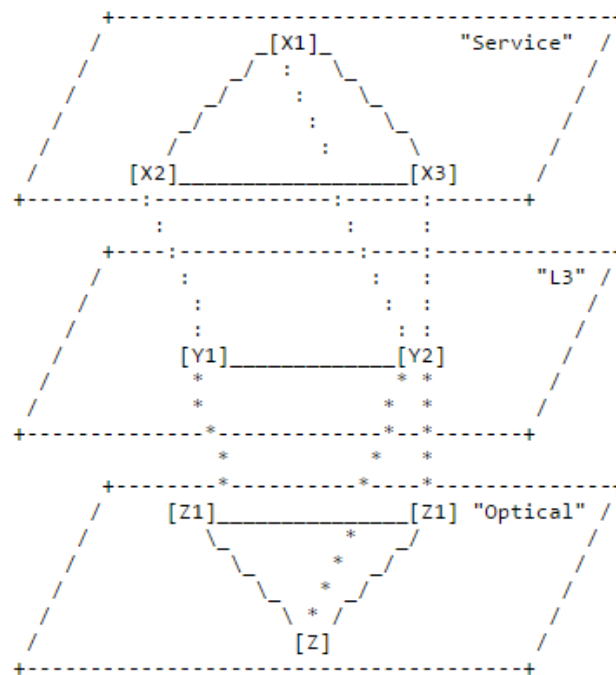


Fig. 10: IETF model topology hierarchy example [CLETOPO].

This model is augmented by the topology model [CLETOPO], which contains:

- An augmentation of the previous model node object to contain a list of Termination Points (effectively, ports), which can, as most objects in this model, reference a supporting equivalent in an underlay topology
- A list of links, which are point-to-point and unidirectional, with parameters including:
 - Source and Destination Nodes and Termination points (ports)
 - References to one or more supporting links in an underlay topology

Then, models [CLEL3] and [CONL2] cover the unicast L3 and L2 topologies, respectively, adding details regarding the IGP parameters and addresses, VLAN tags, etc.

Additionally, the Traffic Engineering (TE) model, described in [LIUTE], defines a technology-agnostic TE-topology type, which can be further augmented by technology-specific models (yet to be developed or at least not yet publicly available). It directly augments the abstract topology model with:

- TE-Topology ID, Provider ID, Client ID: values uniquely identifying a topology; client ID is “0” for native topologies.

- Link TE Attributes: include things such as Admin Status, Performance (TE) Metric, Admin Group, Max Bandwidth, Max Reserved Bandwidth, Unreserved bandwidth, Link protection type, Interface switching capability, SRLGs, and information sources.
- Node TE Attributes: includes an explicit connectivity matrix and information sources.

Additionally, Scheduling parameters for every object (nodes, links ports) are expected to be added in future revisions of the draft.

The characteristics of this network model are summarized in Table 4.

Table 4: Summary of the IETF Network Model.

Good	Bad
Easy to model overlays and tunnels – model is built for it	Topologies are only loosely coupled – may be difficult to manage independent topologies, and there is limited vertical integration
Good technology-specific augmentations for L3 (IP), TE (MPLS/SR), L2 (Ethernet)	No standard technology-specific augmentations for Optical

5.2.1 OpenConfig

OpenConfig [OPCFG] is an industry-led initiative that recently published a set of YANG models, intended to complement and eventually integrate with the available IETF models, representing a vendor-agnostic configuration of IP and MPLS network elements.

While not all models produced by the OpenConfig consortium have been released to the public domain, what is available [OPCFG] covers most aspects of the configuration of TE features in IP networks, but support for optical features is currently severely lacking.

OpenConfig supports the following features of interest to ACINO:

- Management of IP/MPLS interfaces, including administrative status and address;
- Management of MPLS LSP tunnels, including metric, total/reserved bandwidth and administrative status;
- Definition and management of Shared Risk Link Groups (SRLGs) on groups of links.

Alone, the available OpenConfig models only cover the configuration of mostly IP/MPLS network elements, but do not describe the topology in which they operate.

5.3 ONOS Model

The ONOS model is an abstract, protocol-agnostic, distributed network graph, containing the following elements:

- **Device:** represents a network element, e.g. a switch, router, ROADM, etc. Devices are vertices of the network graph representing the network infrastructure components. The description of the vertex includes a chassis ID to identify a physical device, and descriptors such as manufacture name, serial number, software version, etc. One notable missing feature of this model is that it does not allow the representation of resource/path constraints in terms of cross-connectivity between ports within the same device.
- **Port:** describes an interface/port on a Device. A port and a device form a so-called ConnectPoint, which represents an endpoint of a graph edge. It is characterized by an ID to uniquely identify a port within a device, a pointer to the parent device, the port status and a tag to categorize the domain of different network elements (e.g., optical, Layer 2, Layer 3...). The ONOS model defines also a domain-specific API to describe domain-specific ports:
 - Optical port: models OTN, WDM, and has limited support for Flexi-grid. Physical fiber ports are modeled as OMS (Optical Multiplex Section) ports, multiplexing multiple OCH (Optical Channel) ports modeling individual optical signals (lambdas),
 - IP interface: characterized by an IP addresses and a physical identifier.
- **Link:** represents the connectivity between devices and is modeled as an edge between two ConnectPoints belonging to two different Devices. Link descriptions contain the source and destination ConnectPoints and status. An IP-specific link is modeled with source and destination IP addresses and traffic engineering parameters like bandwidth, TE metrics, protection type, etc.
- **Host:** is a node not directly controlled by ONOS and is connected to a Port of a Device. It represents an external agent that drives the network behavior by issuing service requests (via “intents”), but is not part of the network. The host description includes a MAC address, VLAN ID and IP address. It may or may not be relevant for the ACINO model.
- **EdgeLink:** a specialized link connecting a host to a port of a device

Some of these entities rely on the existence of other entities; for example, Ports cannot exist without a parent Device, Links can only exist between existing Ports, etc.

Network topology entities are constructed and maintained by the appropriate ONOS **provider**, which translates the underlying network topology to a protocol-specific model. Providers are software elements that are part of the South-Bound Interface of ONOS, and are responsible for managing the network environment using various control and configuration protocols, insulating the ONOS network model from protocol-aware classification. Providers exploit device-specific communication models/protocol using Drivers. The

aim of this architecture is to provide a network model unbound from specific protocol libraries or implementations.

Furthermore, the current ONOS model defines specific tags to categorize the domain of different network elements. This tagging allows to freely abstract a multi-layer or recursive topology on top of the existing ONOS model.

As mentioned in section 4.1.3, ONOS provides a distributed mechanism to allow fault-tolerance, scalability and resiliency, by ensuring that the global and correct network model is always exposed by the distributed control plane. In particular, each ONOS instance in a multi-controller environment always exposes a view of the entire network even though it has a limited view and direct oversight of a small part of the network.

Lastly, the ONOS model is not concerned with just the underlying network topology, but also maintains the dependencies between high-level services requests and low level network operations. For example, a multi-layer service request involving an IP connection over an Optical lightpath is mapped into two high-level intents:

- A packet intent (e.g. flow-based) representing the connection in the IP layer.
- An optical intent describing the lightpath.

Additionally, each high-level Intent is linked with its own installable operations (i.e., the related configuration of the underlying network elements).

The main characteristics of ONOS model are summarized in Table 4.

Table 5: Summary of the ONOS Network Model.

Good	Bad
Able to model multi-layers topologies	Missing some entities to model both optical and IP features/devices
Easy to freely abstract a multi-layer or recursive topology on top of the network model	

5.4 Selected Model for the ACINO Orchestrator

With the aim of exploiting, preserving compatibility with and contributing back to the ONOS project, the ACINO consortium has decided to focus on extending the existing ONOS core model.

As described above, the model already offers some limited support for the abstraction of both optical and IP devices, however the scope and details of the available modeling are lacking with respect to the needs of the project. In order to comply with the requirements of ACINO, we would need to extend the model and the following entities:

- Optical Transceivers: currently not considered in the model;
- Optical Device: may need a detailed internal cross-connection matrix, if not taken care of by an underlying optical controller;
- MPLS Device/Ports: need to add support for label-switched paths;
- IP Ports (interfaces): need to add management parameters;
- SRLGs: need to add them (and possibly other risk groups).
- IGP Metric: need to know it for properly exploiting and controlling the IP layer.

A more detailed description of the low-level changes to be implemented within the ONOS model and other systems to support the ACINO approach will be reported in deliverables D4.1 (internally) and D4.3 (publicly).

6 ACINO Orchestrator Architecture

As outlined in Section 2, the ACINO orchestrator will be a SDN controller interfacing to underlying IP and Optical controllers (or some other form of control plane) via a South-Bound Interface, and exposing a set service primitives via an intent-based North-Bound Interface, which also gives access to online planning functionalities. These latter two functions are implemented by a multi-layer provisioning and an online planning tool internal modules, respectively.

The proposed architecture is largely bounded and guided by the structure of the chosen SDN framework, ONOS and by the requirements detailed in Section 3. The a high-level architecture is presented in this document with the intent to represent the main functional blocks needed to realize the ACINO concept, rather than a faithful representation of the final modules to be developed (which are the scope of WP4). Nonetheless, some initial ideas about the integration of the ACINO modules with the existing ONOS modules is provided where appropriate and known from the initial activities of T4.1.

A notable feature of the architecture is the online planning module, which could be implemented as one of an application running on top of the orchestrator, communicating through the NBI and exposing its own interface to client applications, an internal OSGI module embedded within the orchestrator, or even a dummy OSGI module implementing a side interface towards an external tool.

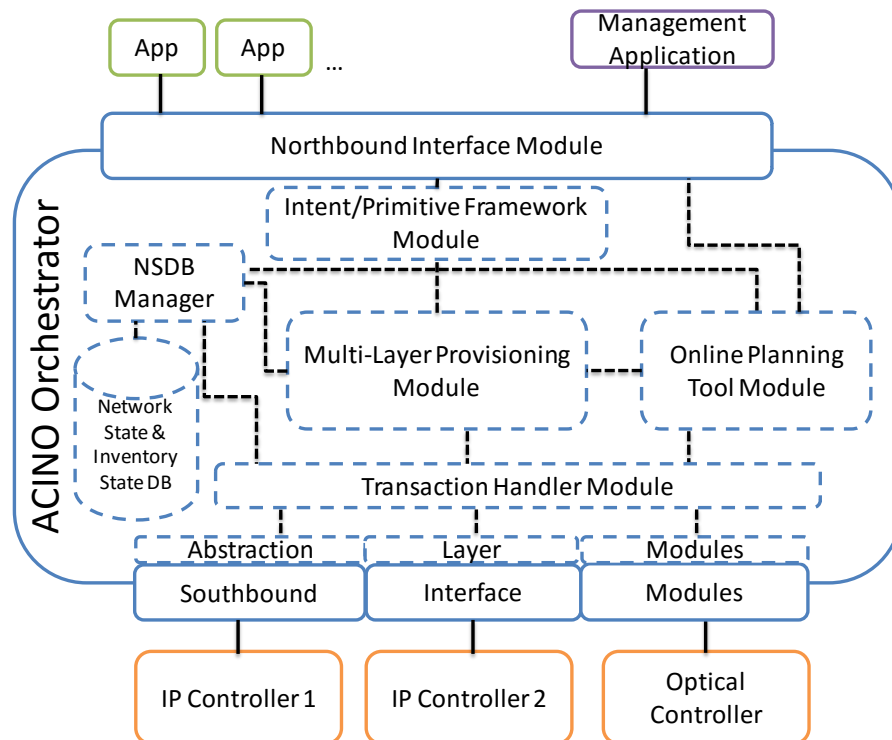


Fig. 11: Proposed architecture for the ACINO orchestrator.

6.1 Detailed Internal Architecture

Given these premises, the architecture proposed for the ACINO orchestrator is depicted in Fig. 11.

In the proposed architecture and future implementation, it is assumed that the ACINO Orchestrator is the only entity taking decisions about network configuration. Such an assumption, while somewhat limiting from a commercial perspective, is deemed acceptable in the context of a research project, as it is mostly bound to the implementation rather than the concept. In fact, the ACINO approach could be applied nearly unaltered even if changes to the network configuration can be injected at the lower layers, as long as:

- there are mechanisms that ensure the prompt update of the network view of the orchestrator when something changes underneath and
- the internal mechanisms of the orchestrator implementation are preemptable, i.e., the internal computations can be forced to restart upon an external change in the underlying topology.

6.1.1 North-Bound Interface Module

The intent-based north-bound interface of the ACINO orchestrator is the primary interface with client applications, including generic network-aware client applications requesting services with specific characteristics and NMS-class applications using the multi-layer allocation and planning functionalities implemented by the Orchestrator to indirectly manage the underlying network.

It exposes two main set of functions:

- 1) a set of intent-based operations to create, modify and cancel network services, which can be characterized using high-level primitives understandable by applications, such as bandwidth, maximum latency, maximum cost, security (encryption), maximum downtime (reliability), and scheduling parameters.
- 2) A set of operations to request online planning computations to test possible new configurations, and push (implement) a new overall configuration to the underlying network.

The first set of functions interfaces with the internal **Intent/Primitive Framework**, while the planning operations interface with the internal **Online planning Module**.

A detailed description of the North-Bound API, possible interface implementations, and the workings of the Intent/Primitive Framework will be reported in D3.2.

Regarding the implementation of this module, it could integrate and expand the REST module already available within ONOS.

6.1.2 Intent/Primitive Framework Module

This module interfaces with the NBI and the multi-layer provisioning module, and is responsible for translating application-level requirements into service requirements, i.e. infrastructure-aware representations used as input constraints by the multi-layer provisioning algorithm.

A detailed description of the functioning of the Intent/Primitive Framework and the NBI operations it supports will be reported in D3.2.

Regarding the implementation of this module (and the actual NBI), it could integrate and greatly extend the Intent Framework already available within ONOS, mapping application intents into ONOS high-level intents and service requirements into ONOS installable intents.

6.1.3 Network State and Inventory DB and Manager Module (NSDB)

This database and manager stores a local representation of the underlying IP/MPLS and optical layers, including state, configuration and (potentially) inventory information. The network state stored in this module is consumed by the Multi-Layer Provisioning and Online Planning Modules.

Network state is initially populated by the Abstraction Layer module, using available discovery process to determine the underlying network configuration. The DB is kept up to date by both internal requests and external triggers. For example, when a new configuration is pushed down to the networks by the Multi-Layer Provisioning or Online Planning modules, it is also stored in the DB (possibly marked as unconfirmed). Once this configuration is provisioned, confirmations of the implementation of new configuration and asynchronous events (including alarms or changes in the network triggered via other channels) coming from the Transaction Handler or Abstraction Layer modules are also stored in the DB (possibly matching to existing unconfirmed configurations).

With respect to the implementation, this module is envisioned to be an extension of the ONOS core model, with the additional of several details stored in either extended or new drivers (e.g. for optical).

6.1.4 Multi-Layer Provisioning Module

This module implements the dynamic multi-layer provisioning algorithms to be developed within the ACINO project. It interfaces with the Intent/Primitive Framework to receive service requests, and maps them to the available network resources, as stored in the Network State and Inventory DB module. It also interfaces the Transaction Handler module for pushing down new configuration to the underlying networks and pulling their initial/updated configuration.

It produces configuration from service requests, taking into account the current state of both optical and IP/MPLS resources, and deciding e.g. whether the incoming request can be satisfied by adjusting the current IP/MPLS topology or new optical connections are needed.

Additional details regarding the multi-layer provisioning algorithms will be found in deliverable D3.4.

One possibility is to implement this module within the ONOS Intent Framework, as a new installer object implementing new, multi-layer optimization algorithms.

6.1.5 Online planning Tool Module

This module implements the online planning functions of the ACINO Orchestrator. More specifically, it allows an external application (e.g. a NMS) to simulate the effect of failures, selective addition of resources

or re-optimization of the network configuration. Such application may then push the resulting configuration down to the underlying networks.

The module interfaces with the NBI module to expose access to its functions, to the Networks State and inventory DB Manager to retrieve an up-to-date representation of the controlled networks' state, and to the Transaction Handler module to inject the computed new configuration into the network elements.

This module is decoupled from the Multi-Layer Provisioning Module, despite dealing in similar multi-layer problems, due to their different scopes. The Multi-Layer Provisioning Module handles direct requests for new services, which must be served in a small timeframe, while this module handles much slower overall re-optimization and simulation problems.

Additional details regarding the multi-layer optimization algorithms will be found in deliverable D3.3.

6.1.6 Transaction Handler Module

This module interfaces with both the Multi-Layer Provisioning and Online Planning modules, as well as with the Abstraction Layer Module responsible for translating network-specific models into the internal representation used by the ACINO Orchestrator.

The purpose of this module is to ensure the transactional nature of configuration changes involving multiple operations in the underlying networks is preserved. This involves keeping state of ongoing configuration operations, ensuring that all atomic configuration sub-actions are completed successfully before marking their parent action as completed, and ensuring correct rollback in case one or more atomic sub-actions fail to complete.

6.1.7 Abstraction Layer Modules

These modules interface with the Transaction handler module to receive incoming configuration requests, to the Network State DB manager module to store both the initial configuration and any unsolicited update coming from the underlying networks, and to the modules implementing specific south-bound protocols that make up the SBI proper.

The function of these modules is to translate between the internal representation used by the ACINO Orchestrator and those (possibly standardized) used by specific underlying networks control planes.

With respect to the implementation, these modules match well with the concept of “providers” within the ONOS framework; several must be extended or created from scratch to model our IP/MPLS and Optical layers.

6.1.8 Southbound Interface Modules

This collection of modules interfaces with specific Abstraction Layer modules of the ACINO Orchestrator, and implements the various protocols and specific commands used to interact with the underlying layers.

The southbound interface is responsible for:

- pulling up topology, capabilities and notifications/alarms (i.e., both state and initial/current configuration) from the underlying controllers/control planes of the IP and Optical layers;
- Pushing down configuration to implement the selected multi-layer strategies, i.e., provision, modify and delete IP/MPLS tunnels and optical circuits.

Various possible protocols and models to implement this interface for IP/MPLS and Optical controllers were discussed in Sections 4.2 and 4.3; several of the standard multi-layer models discussed in Section 5 could also be used. Most controllers expose some form of REST north-bound interface, although these are typically not standardized. One possible emerging standard protocol for this interface is NETCONF, using YANG models from the COP protocol [COP], developed as part of the STRAUSS European project [STRAEU].

Regarding their implementation, these modules would match well with the “driver” objects recently introduced within the ONOS framework; at least a few will need to be developed to interface with the technologies that will be used in the project testbed and emulation environments.

6.2 Examples of typical Orchestrator Operations

This section describes some typical operations of the proposed ACINO Orchestrator, specifically: setting up a dynamic connection from an intent, computing a what-if scenario, performing global re-optimization, and handling a restoration scenario.

6.2.1 Intent-based Dynamic Connection Provisioning

This scenario represents a service request from an application:

1. The applications sends an Intent to the NBI, specifying how to recognize its traffic and the characteristics of the service it desires;
2. The NBI forwards this request to the Intent/Primitive Framework, which produces a set of possible service descriptions (e.g. IP tunnel(s), a new optical lightpath, etc.) implementing the requested service, and forwards it to the Multi-Layer Provisioning module;
3. The Multi-Layer Provisioning module pulls an updated representation of the network from the NSDB Manager and internally checks, among the various options provided by the Intent/Primitive Framework, identifies options that can be physically be realized and selects the option with the best overall “cost”.
4. If no realization of the service is found, the Multi-Layer Provisioning module sends a negative reply to the Intent/Primitive Framework, which in turn forwards it to the NBI, which informs the application. The level of detail of this reply (e.g., indicating which constraint(s) cause the rejection or possible feasible alternatives) will be defined in deliverables D3.2 and D3.3.
5. If a feasible realization is found, it is forwarded to the Transaction Handler module, which stores it as “temporary configuration” in the NSDB and individually contacts the various Abstraction layer

and SBI modules responsible for the control of the affected devices to implement the desired changes.

6. If one or more modules fail to implement the changes in a timely fashion, the Transaction Handler module reverts the configuration it sent to the other modules, erases the new service, and signals back all the way to the NBI that the new intent has failed. Attempts to re-install the service may be made at intermediate modules before finally informing the requesting application (not detailed here, will be discussed in WP4).
7. If all changes are implemented correctly and in a timely fashion, the transaction handler marks the relevant resources as occupied in the NSDB and informs the upper modules, all the way to the application, that the requested service is now up and running.

6.2.2 What-If Scenario

This scenario represents a what-if computation asked by a NMS to the Online Planning tool of the ACINO Orchestrator:

1. The NMS pulls an updated multi-layer topology and active service representation from the NBI (which requests it to the NSDB).
2. The NMS sends a request with a modified topology (e.g., the addition of a fiber link, or the setup of a new optical connection) to the NBI, which forwards it to the Online Planning tool.
3. The Online planning tool pulls the current network state from the NSDB and computes the requested scenario (in this case evaluating possible new paths for existing optical connections or the placement of IGP-controlled traffic, respectively), reporting it back to the requester.
4. In the case of a software configuration change (such as establishing a new lightpath), the requester, after evaluating the effects of the change, can request their actual implementation via the NBI. In this case the request is forwarded all the way to the Transaction Handler which handles this in the same fashion as a request generated by a simple intent.

6.2.3 Global Re-Optimization

This scenario represents a global-re-optimization triggered via a NMS:

1. The NMS requests the re-optimization of the network to the NBI, which forwards it to the Online Planning tool.
2. The Online planning tool pulls the current network state from the NSDB, signals that it is going to re-optimize the network (so that incoming requests are delayed, or that it is informed of any change and restarts the computation if a new intent comes while it is busy) according to some metrics (to be defined in D3.3 and D3.4) and begins the computation.
3. Once the computation is concluded, if a new optimal configuration (and a migration strategy) is found, it is sent to the requester (via the NBI) for validation.

4. If the requester accepts (via the NBI) the new configuration, it is sent to the Transaction Handler for implementation (as for standard intents).

6.2.4 Restoration Scenario

This scenario represents a failure event in which a fiber is cut and the Orchestrator must take decision regarding the fate of current traffic:

1. A physical link in the optical layer fails, and the optical control plane exports this event to the Orchestrator via the SBI.
2. While the optical control plane attempts to restore all lightpaths affected by the link failure (either on its own or directed by the orchestrator), ONOS recompiles the affected intents and reroutes traffic on different IP links.
3. Once the optical restoration process is complete and the path or characteristics of the restored lightpath are known, ONOS once again recompiles the affected intents, potentially using the restored link for some but not all traffic it previously carried.

7 Conclusions

This deliverable describes the requirements and proposed architecture of the ACINO orchestrator. After an overview of the main elements of the system, it gives detailed requirements for each of the components. The requirements related to the primitives and the intents have been linked with the Case Studies to which they refer, as defined in deliverable D2.1. Potentially, these requirements could be tweaked in a future stage, e.g. due to the inclusion of additional use-cases or new features from the underlying controllers.

The document also provides an overview of existing SDN frameworks and discusses potential options for frameworks that could be used to control the IP/MPLS network, the optical network, or to realize the ACINO orchestrator. The review finds that the ONOS framework is the most suitable among those available to realize the ACINO orchestrator, due to its high visibility and a number of features that may be employed as a basis for the development of the ACINO concept.

With respect to the network topology model, several multi-layer models being proposed in multiple standardization bodies are analyzed with their pros and cons. In particular, with the aim of exploiting, preserving compatibility with and contributing back to the ONOS project, the ACINO consortium has decided to focus on extending the existing ONOS core model. Consequently, an initial list of the additions that will need to be integrated into the starting framework model is presented.

Finally, a detailed description of the proposed architecture and internal modules, how they would map into existing ONOS objects and how they would interact to perform the functions of the Orchestrator is given.

Overall, the information included into this deliverable is expected to feed the design (T3.2) activities of the North-Bound Intent-based Interface, by defining its requirements and the modules it must interface with. It also drives the implementation activities of WP4, as it defines the starting framework, the modules to be realized and describes how they should interact to achieve an application-centric control of network resources.

List of abbreviations and acronyms

Abbreviation	Meaning
ABNO	Application-Based Network Operations
API	Application Program Interface
AS	Autonomous System
BGP	Border Gateway Protocol
CLI	Command Line Interface
COP	Control Orchestration Protocol
CS	Case Study
DCN	Data Communication Network
E2E	End-to-End
eBGP	External Border Gateway Protocol
ERO	Explicit Route Object
FCAPS	Fault, Configuration, Accounting, Performance and Security
GMPLS-ENNI	Generalized Multi-Protocol Label Switching - External Network to Network Interface
JSON	JavaScript Object Notation
HTTP(S)	Hypertext Transfer Protocol (Secure)
ID	Identifier
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IP	Internet protocol
IS-IS	Intermediate System to Intermediate System
L2	Layer 2
L3	Layer 3
L3VPN	Layer 3 Virtual Private Network
LSP	Label Switched Path
LTP	Logical Termination Point
MAC	Media Access Control

MPLS	Multi-Protocol Label Switching
MTOSI	Multi-Technology Operations System Interface
NBI	North-Bound Interface
NETCONF	Network Configuration Protocol
NFV	Network Function Virtualization
NMS	Network Management System
NSDB	Network State and Inventory DB and Manager Module
OAM	Operation Administration Maintenance
OCH	Optical Channel
ODL	OpenDaylight
OF	OpenFlow
OMS	Optical Multiplex Section
ONF	Open Networking Foundation
ONH	Optical Network Hypervisor
ONOS	Open Network Operating System
OSGi	Open Service Gateway Initiative
OSS	Operations Support System
OTN	Optical Transport Network
OSPF	Open Shortest path First
PCC	Path Computation Client
PCE	Path Computation Element
PCEP	Path Computation Element Protocol
PM	Provisioning Manager
REST	Representational State Transfer
RSVP-TE	Resource Reservation Protocol – Traffic Engineering
ROADM	Reconfigurable Optical Add Drop Multiplexer
SAL	Service Abstraction Layer
SBI	South-Bound Interface
SDN	Software Defined Networking

SNMP	Simple Network Management Protocol
SRLG	Shared Risk Link Group
SRNG	Shared Risk Node Group
T-API	Transport-API
TE	Traffic Engineering
TM	Topology Module
UNI	User Network Interface
VLAN	Virtual Local Area Network
VNTM	Virtual Network Topology Manager
WDM	Wavelength Division Multiplexing
XRO	eXclude Route Object

References

- [ACID21] ACINO Deliverable D2.1, “Initial report on network architecture, use cases and application requirements”, 2016
- [KRE14] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey”, CoRR, 2014, arXiv:1406.0440
- [ODL] OpenDayLight, <http://www.opendaylight.org/>
- [ONOS] ONOS, <http://onosproject.org/>
- [ODENOS] ODENOS, <https://github.com/o3project/odenos>
- [ABNO] ABNO, <https://github.com/telefonicaid/netphony-abno/wiki>
- [OF] ONF, OpenFlow, <https://www.opennetworking.org/sdn-resources/openflow>
- [SAUOF] Saurav Das, “Extensions to the OpenFlow Protocol in support of Circuit Switching”, http://archive.openflow.org/wk/images/8/81/OpenFlow_Circuit_Switch_Specification_v0.3.pdf
- [COP] Control Orchestration Protocol, <https://github.com/ict-strauss/COP>
- [STRAEU] STRAUSS European project, <http://www.ict-strauss.eu/>
- [RFC7491] D. King, A. Farrel, “A PCE-Based Architecture for Application-Based Network Operations”, RFC 7491
- [FLOVIS] FlowVisor, <https://openflow.stanford.edu/display/DOCS/Flowvisor>
- [RFC4655] A. Farrel, JP. Vasseur and J. Ash, “A Path Computation Element (PCE)-Based Architecture”, RFC 4655
- [CRAB13] E. Crabbe, I. Minei, J. Medved, R. Varga, “PCEP Extensions for Stateful PCE”, draft-ietf-pce-stateful-pce-13
- [RFC5440] JP. Vasseur, JL. Le Roux, “Path Computation Element (PCE) Communication Protocol (PCEP)”, RFC 5440
- [LIUTE] X. Liu, I. Bryskin, V. P. Beeram, T. Saad, H. Shah, O. G. D. Dios, “YANG Data Model for TE Topologies”, draft-ietf-teas-yang-te-topo-02
- [LIUAB] X. Liu, I. Bryskin, A. Clemm, V. P. Beeram, “A YANG Data Model for Abstract Network Topologies”, draft-liu-netmod-yang-abstract-topo-00
- [CHETE] X. Chen, Z. Li, X. Zeng, “Yang Model for MPLS Traffic Engineering(TE)”, draft-chen-mpls-te-yang-cfg-00
- [GANTE] R. Gandhi, T. Saad, R. Sawaya, “YANG Data Model Tree Structures for MPLS Traffic Engineering Tunnels and Links”, draft-gandhi-mpls-te-yang-model-00

- [NORTHS] Juniper NorthStar, <http://www.juniper.net/us/en/products-services/sdn/northstar-network-controller/>
- [ONFCIM] ONF Core Info Model, onf2015.074_Core_Info_Model.05
- [LAMTE] K. lam et al., “Usage of IM for network topology to support TE Topology YANG Module Development”, draft-lam-teas-usage-info-model-net-topology-02
- [TAPIFR] ONF “Functional Requirements for Transport API”, TAPI_FRS.11
- [CLETOPO] A. Clemm et al., “A Data Model for Network Topologies”, draft-ietf-i2rs-yang-network-topology-02
- [CLEL3] A. Clemm et al., “A YANG Data Model for Layer 3 Topologies”, draft-ietf-i2rs-yang-l3-topology-01
- [CONL2] J. Dong and X. Wei, “A YANG Data Model for Layer-2 Network Topologies”, draft-ietf-i2rs-yang-l2-network-topology
- [OPCFG] OpenConfig public Github model repository, <https://github.com/openconfig/public>