Virtualization of Disaggregated Optical Networks with Open Data Models in Support of Network Slicing

RAMON CASELLAS, [1,*] ALESSIO GIORGETTI, [2] ROBERTO MORRO, [3] RICARDO MARTINEZ, [1] RICARD VILALTA, [1] RAUL MUÑOZ, [1]

[1]*CTTC/CERCA, Av. Carl Friedrich Gauss n.7, 08860 Castelldefels, Barcelona, Spain.*
[2]*CNIT/SSSUP, Via Moruzzi n. 1, 56124 Pisa, Italy*
[3]*TIM, Via G. Reiss Romoli n. 274, 10148 Torino, Italy.*

**Corresponding author: ramon.casellas@cttc.es*

Received 06 July 2019; revised XX Month, XXXX; accepted XX Month XXXX; posted XX Month XXXX (Doc. ID XXXXX); published XX Month XXXX

Network slicing has been a major selling point of 5G networks, where a slice is roughly defined as a self-contained logical network on top of a shared infrastructure tailored for a specific service or vertical industry, composed of heterogeneous resources (computing, storage, bandwidths) and typically involving specific application and/or networking functions. The relationship between the network slicing and the underlying network virtualization is an open research topic, allowing multiple deployment models. In particular, a specific model of interest involves the virtualization of the optical infrastructure, where optical virtual networks are instantiated in support of network slicing, so the virtual network underlies and provides connectivity to component functions of a network slice.

In this paper, we propose and implement a network virtualization architecture for open optical (partially) disaggregated networks, based on the concept of device hypervisor relying on OpenConfig and OpenROADM data models, in support of 5G network slicing over interconnected NFVI-PoPs. The architecture is experimentally validated, showing the provisioning of ITU-T flexi-grid Network Media Channels across a virtualized network.

http://dx.doi.org/10.1364/JOCN.99.099999

1. INTRODUCTION

Network slicing has been a major selling point of 5G networks since initially proposed by the NGMN [1], to deal within the heterogeneous requirements of services offered to a multiplicity of vertical industries, in terms of e.g., latency, bandwidth, security, or resiliency.

Briefly, a network slice instance (NSI) is composed of a set of network functions and the resources enabling the deployment of these functions, forming a complete instantiated logical network to meet certain network characteristics for a specific service. Macroscopically, corresponds to a logical network of interconnected functions and resources, supported over a shared infrastructure, generalizing and unifying previous concepts such as virtual private networks (which become mechanisms by which slicing is deployed and enforced).

In this work, we focus on network slicing that is implemented in terms of the ETSI NFV framework, with the specific scenario that the underlying infrastructure spans multiple NFVI points of presence (NFVI-PoP) that are geographically distributed and interconnected by an optical transport network. Network Slices are mapped to ETSI NFV Network Services (NS, roughly defined as a set of interconnected VNFs) supporting a given application. The adoption of a reference architecture such as the ETSI NFV Management and Orchestration (MANO) framework can provide an efficient network service management and resource orchestration from an end-to-end perspective [2]. In particular, we assume that the optical transport network is partially disaggregated (disaggregating the optical transceiver from the network elements) and using open and standard data models for the considered devices. In this context, the transport network needs to become a resource under the umbrella orchestrating system.

The relationship between slicing and the allocation of resources from the underlying transport network is still a topic of active research, including the role of network virtualization. In this regard, it is expected that the *virtualization of the transport network* itself [3] can provide the desired isolation from the transport network perspective. *Network virtualization* is roughly defined as the partitioning and composition of the underlying physical optical infrastructure to create co-existing *Optical Virtual Networks (OVN),* and is a key enabler for network slicing, as it provides the necessary technologies to fulfill the transport requirements associated to the supported slice instance(s). An OVN underlies and provides connectivity to component functions of a network slice. Ultimately, the joint optimization of the Network Slice Instances(s) – including function placement -- and the underlying OVN provides the maximum flexibility for the constrained allocation of resources and efficient end-to-end network slice orchestration relies on multi-layer optimization (e.g. dynamic demands for the slices instances may have a clear impact on the optical transport). Fig. 1 shows the architectural elements in which Network Slices are deployed over one or more ETSI NFV network service(s) using a NFV Orchestrator (NFVO), which allocates Virtualized Network Functions (VNFs). The NFVO is able to request connectivity services from a SDN controller, by means of a dedicated functional component of the MANO named WAN Infrastructure Manager (WIM).

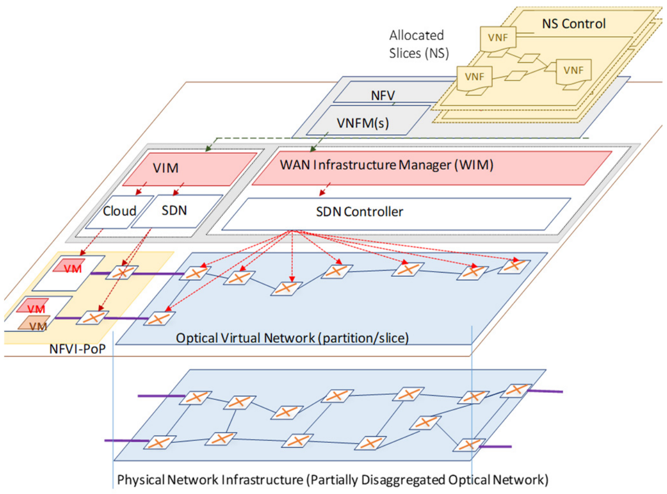

Allocated Slices (NS)
NS Control
VNF
VNF
VNF
NFV
VNFM(s)
VIM
WAN Infrastructure Manager (WIM)
Cloud
SDN
SDN Controller
VM
VM
VM
NFVI-PoP
Optical Virtual Network (partition/slice)
Physical Network Infrastructure (Partially Disaggregated Optical Network)

Fig. 1. Enabling Network slicing over virtualized disaggregated optical networks with ETSI NFV MANO.

Network virtualization has been thoroughly studied in the past (see [3, 4] and references within) both from an architectural and protocol point of view, including challenges and enabling technologies. Algorithmic aspects related to the so-called *network embedding* [5, 6, 7] address the computation of virtual networks on top of a common infrastructure. This work complements previous ones in the sense that we consider the virtualization of an open transport network, with the aim of supporting the aforementioned NFV-based slicing, and assuming a partial disaggregation model. Let us note that an in-depth discussion of the different options regarding disaggregation is provided in [8]. We limit our scope to partial transceiver and network element disaggregation as the main use cases in the scope of the Metro-Haul [9] and the ODTN [10] projects.

Our macroscopic contribution covers the design, implementation and experimental evaluation of a *network virtualization mechanism that relies on virtualized devices characterized by their data model*. We cover devices modeled as OpenConfig terminal devices/Optical Platforms [11] or OpenROADM devices [12] as detailed in Section 2. Macroscopically, our main contributions are:

1) We propose and implement a virtualization/slicing architecture based on the concepts of *OVN manager* and *per-device hypervisor entity.*
2) We extend the ONOS SDN controller with OpenConfig and OpenROADM device drivers and applications exporting a TAPI North Bound Interface (NBI) to control a disaggregated optical network based on such data models, enabling end users to automatically provision connectivity services that correspond to ITU-T network media channels (NMC) either directly over a physical infrastructure or a virtualized one.
3) We demonstrate the provisioning of such NMCs a particular OVN, with emulated optical nodes that are virtualized using containers.

This paper extends the contribution presented in OFC2019 [13]. The main extensions in this paper are as follows. First, we elaborate on the concepts of network slicing and network virtualization, linking with actual initiatives such as the ACTN model, the ONF network virtualization and TAPI models in order to provide the technological background. Next, the used data models for the SDN controller North Bound Interface (NBI) as well as the underlying optical devices (in particular, OpenConfig and OpenROADMs) are detailed in Section 3. A new section has been introduced to provide as short summary of implementation aspects -- related to the integration of such devices, covering both the development of drivers within our selected SDN framework and the development of the SDN agents – reported in Section 4, along with pointers to Open Source software within the ONF ONOS Network Operating System framework. Additional examples on how a device is (statically) partitioned are provided. Finally, additional experimental data from the validation and proof-of-concept has been added to the section on performance evaluation.

2. OPEN DATA MODELS

The proposed network virtualization approach of disaggregated optical networks relies on (ideally open and standard) data models that are modular and heavily rely on *componentization*. Such data models abstract the underlying devices in terms of *components*, ports, links, circuit packs, degrees / directions, etc., easing the design of (quasi static) partitioning of such devices, assigning a subset of resources to a particular instance. Before introducing the virtualization and partitioning mechanism, this section reviews the main traits and characteristics of the involved data models, not only from a device perspective, but also from the perspective of the service models and northbound interfaces.

A. Transport API (TAPI) as North Bound Interface

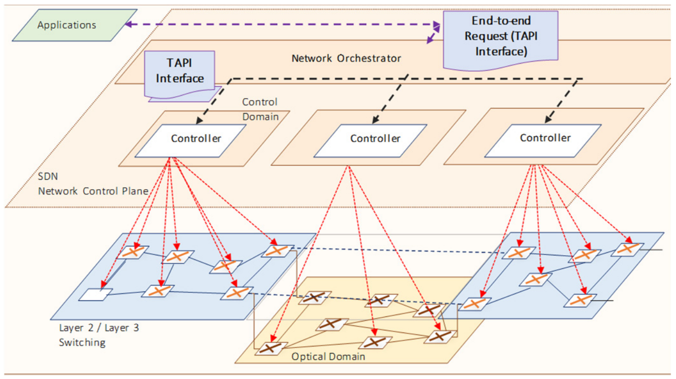


Fig. 2. TAPI models as Controller NBI (network orchestration)

With end-to-end optical networks being increasingly segmented in domains, (e.g. in deployment models commonly referred to as vendor domains or islands), SDN Controllers offer proprietary interfaces to applications (or Network Orchestrators). This heterogeneity of controllers interfaces forces the use of “dedicated plugins”, making it difficult and expensive to extend. There is a clear need for a standard interface, across multiple domains and vendors, with common models, to act as a controller NBI (see Fig. 2). The Transport API (TAPI) [14, 15] published by the ONF meets the main requirements to be a protocol and interface used between an orchestrator and multiple domain controllers. Of particular interest is the TAPI 2.1 release, with its increased support for the photonic media layer. In simple terms, a TAPI based interface between a client and a server offers multiple services.

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

Topology context and models. If a given TAPI server supports the topology model, it augments the TAPI shared context with a list of topologies. Each topology is composed of a list of nodes, which, in turn, have Node Edge Points (NEPs). Links connect two NEPs. The model is flexible enough to support recursive topologies and different levels of abstraction, with the level of detail exported being configurable by SDN controller policy. A client is thus able to obtain an (abstracted) view of the topology and map TAPI SIPs to external NEPs. Topologies can be recursive in which a given node at a given topology abstraction layer may abstract multiple nodes in a network subdomain (see Fig. 3).

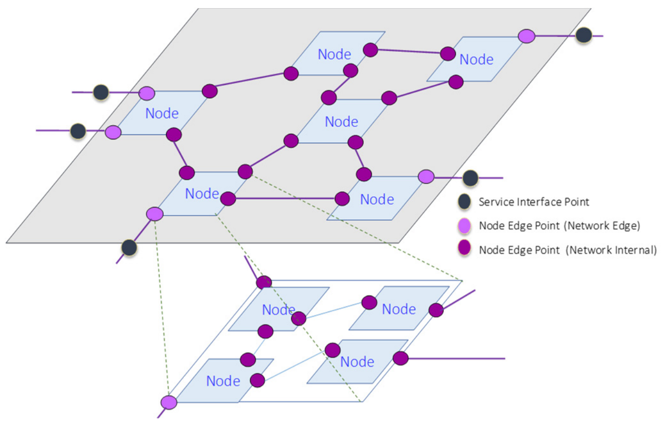

Node
Node
Node
Node
Node
Node
Service Interface Point
Node Edge Point (Network Edge)
Node Edge Point (Network Internal)
Node
Node
Node
Node

Fig. 3. Illustration of TAPI models: the notion of TAPI domain (recursive) topology.

Connectivity context and models. This model augments the shared context in order to support Connectivity Services. The instantiation of a connectivity service relies on the instantiation of several connections (e.g. one end-to-end and internal at each TAPI node, see Fig. 4). For this, Connection End Points (CEPs) are instantiated over NEPs and connections involve two or more CEPs. A common operation is thus to request a connectivity service between two SIPs (for a point-to-point connection) and the SDN controller shall perform the logic to perform per-device configuration based on the end-to-end service.

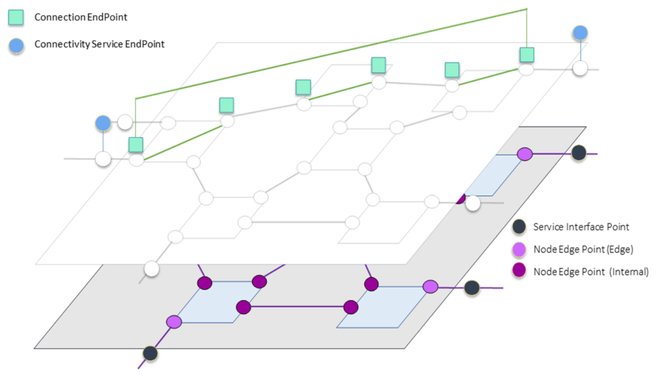


Fig. 4. Illustration of TAPI models: Connectivity Service / Connection concepts, between Connection End Points (CEPs).

B. OpenConfig Optical Platform and Terminal Devices

The OpenConfig project focuses on compiling a consistent set of vendor-neutral YANG data models. These data models cover a variety of network elements, from routers to optical switches. In our scope, optical transceivers (and similar devices such as mux-ponders) are modeled using the OpenConfig platform and the optical terminal-device models. The former defines a platform as having software and hardware components (such as line cards, ports, transceivers or optical channels). The latter allowing the activation of optical channels – in terms of frequency, power and operational mode – within line ports and the association (mapping) of client signals to optical channels in a quite flexible way. The latter covers the configuration of a terminal device and encompasses two main operations: first, configuring the optical channels that are present at the line ports, in terms of nominal central frequency, operational mode, and transmission power. Second, to associate client ports signals to optical channels in one or more multiplexing stages [11].

C. OpenROADM device model

The device model proposed by OpenROADM (in this work we cover device model v2.2) is sketched in Figure 5. From a device perspective, a ROADM is composed of a number M of directions or degrees (DEG) and a number N of Shared Risk Groups or SRGs (OpenROADM terminology for Add/Drop stages). A given degree has RX and TX amplifiers and a WSS to MUX/DEMUX the optical signals coming to/from external links, other degrees or add/drop stages. SRGs combine WSS, amplifiers and Combiners/Splitters. All the different elements are interconnected by physical and internal links.

The actual YANG device model is quite comprehensive. Macroscopically, it defines a first section related to the device information (common language node identifiers, vendor, model, serial number, geolocation coordinates, etc.) followed by a section that includes a list of circuit-packs, describing the physical architecture including their components ports and naming, as well as the correspondence in terms of actual racks and shelves.

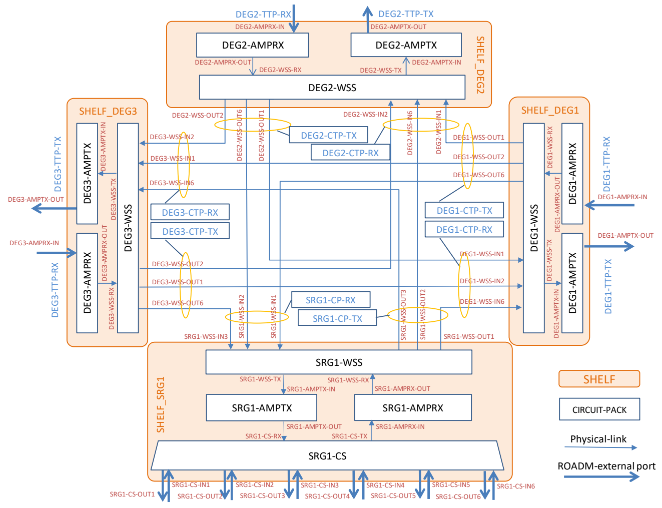
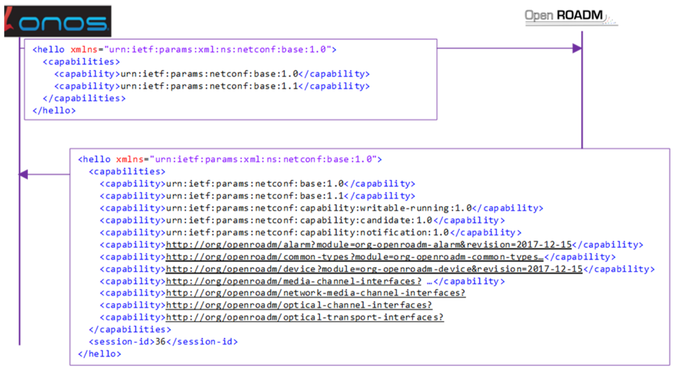


Fig. 5. Logical view of an OpenROADM v2.2 device, showing the main components such as 3 degrees/directions (with their respective TX/RX amplifiers and WSS) and a Shared Risk Group (SRG).

The next section details the set of ROADM interfaces (logical connection or trail termination points that can be supported over either underlying logical interfaces or directly physical ports), resulting in an interface hierarchy. Another section lists the internal links (links between ports of a given component or circuit pack), the physical links

(links between different components, such as a link between degrees or between degrees and SRGs, commonly named express, add or drop links) and external links (links that are instantiated to reflect connectivity between ROADMs). Next, the model also includes two lists for the main components: a list of numbered degrees, which defines the ROADM overall degree, and a list of SRGs including a list of add drop port pairs (client ports towards the terminal devices). Finally, the `roadm-connection` list includes the connections that are active (established) in the device, between two logical interfaces.



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

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

Fig. 6. Initial Capability discovery of the device, using the exchange of NETCONF hello messages.

Network and Topology Discovery: From the point of view of networks operation, the network operator configures the SDN controller with attributes of the devices that are to be controlled. This includes IP addresses and ports where the NETCONF server is running as well as other relevant data such a user credentials, etc. The SDN controller establishes a persistent NETCONF session with the device. After an initial capability exchange, in which the NETCONF client (ONOS) discovers what models and features are supported (Fig. 6) using a `HELLO` message. Next, the SDN controller issues `<get>` and `<get-config>` messages as needed to retrieve info about the devices. Typically, a `<get>` operation with a subtree filter of `<org-openroadm-device><info/>` allows retrieving basic data to add the device into the SDN controller device manager. Similar operations regarding the composition of the device circuit packs and ports, in order to retrieve internal connectivity and to discover port capabilities.

Service Creation: to request a connectivity service, the user or operator may retrieve the list of the TAPI context SIPs and request a connectivity service between a pair of them. The SDN controller maps those SIPs to node edge points, and performs a routing and spectrum assignment process that finds the k-shortest path between the devices and performs first fit spectrum allocation. Once completed, the controller configures flow rules each device: for the Terminal Device, a logical channel association is instantiated within the device between a client (transceiver) port and an optical channel component bound to the line port of the device. For each of the OpenROADM devices across the path, a ROADM internal connection is requested: i) OTS and OMS (optical transport and optical multiplex) interfaces are created within each degree (if not existing), ii) Supporting Media Channel and NMC interfaces are created, iii) Followed by the creation of a `roadm-connection` object.

3. DISAGGREGATED NETWORK VIRTUALIZATION

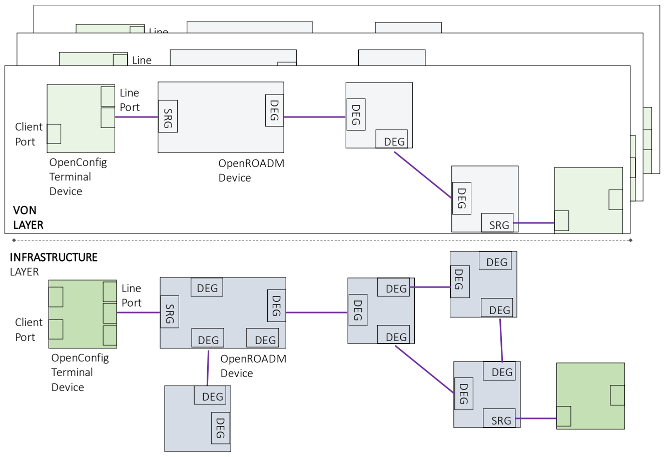


Fig. 7. Sample virtual optical networks on top of a shared common infrastructure for a given resource partition.

The need to provide network abstraction and virtualization has emerged as a key requirement for operators [16], as transport networks evolve. Tenants are given abstracted topology views of the physical underlying network and are allowed to utilize and independently control allocated virtual network resources as if were real (see Fig. 7). Multiple deployment models are possible, in which the granularity level of control given to tenants can vary, ranging from simple static allocation to the allocation of virtual mesh network topology with dynamic customer control.

Generally, virtualizing an optical network can be performed using a combination of a given device support (that is, hardware support) and/or the use of a virtualization layer (commonly referred to as hypervisor) that extends or emulates such support. This, in turn, can be supported at the device level, at the SDN controller level, or both.

A. SDN Controller based Network Virtualization

A common approach for network virtualization relies on implementing the abstraction, isolation and lifetime management of the Optical Virtual Networks at the SDN controller level. This is motivated, in part, by the centralized nature of SDN Controllers, in which partitions or views of the network topology can be exported to specific clients based on configuration and policies, and the controller implements the logic to map from OVN instances to the underlying physical infrastructure. In other words, the controller implements a *network hypervisor* offering isolated network views to customers or tenants.

For example, the IETF Abstraction and Control of Traffic-Engineering Networks (ACTN) architecture [17], defines a hierarchical SDN based solution spanning multiple SDN Controlled domains and remaining compatible in supporting legacy heterogeneous transport network control/management technologies (e.g., GMPLS/ASON, PCE, NMS/EMS) as shown in [16]. In this setting, the ACTN defines the requirements, use cases, and architectural elements, relying on the concepts of network and service abstraction, detaching the network and service control from the underlying data plane. The architecture encompasses Physical Network Controllers (PNCs), responsible for specific technology and administrative domains, orchestrated by Multi-Domain Service Coordinator (MDSC), which, in turn, enables underlay transport resources to be abstracted and virtual network instances to be allocated to customers, under the control of a Customer Network Controller (CNC) [19].

Another relevant example is found in the TAPI model family. In addition to common models related to topology, connection management and path computation, more advanced models allow the management of virtual networks within a specific TAPI client / server context [**¡Error! No se encuentra el origen de la referencia.**]. The abstraction offered by the TAPI models and the Common Information Model (CIM) elements enables to virtualize the network, to support Virtual Transport Network services offering dynamically controllable virtual resources e.g., connecting remote sites of large customers.

B. Device based Network Virtualization and Device Hypervisor

A complementary approach relies on supporting virtualization directly at the device level. A key advantage is that with full virtualization support, the SDN controller and other functional elements may operate without change, unaware of whether they operate on virtualized of physical devices. However, it is worth noting that the use of a device-based virtualization mechanism does not exclude the use of a controller-based approach: we define the **OVN manager** as a functional component that instantiates and controls the lifetime of the OVNs. As such, it can be operated independently and deployed as part of the network operator operation support system (OSS), or as part of an SDN controller in an integrated solution. The second key component in the proposed architecture is the **OpenROADM hypervisor**. Conceptually, it behaves like a hypervisor in the computing domain: a hypervisor is a functional element (e.g. computer software, firmware or hardware) that instantiates and enables the controlled execution of virtual machines – referred to as guests – running over a common physical server - referred to as host – and presents the guest operating systems with a virtual operating platform. Multiple instances may thus share the virtualized hardware resources.

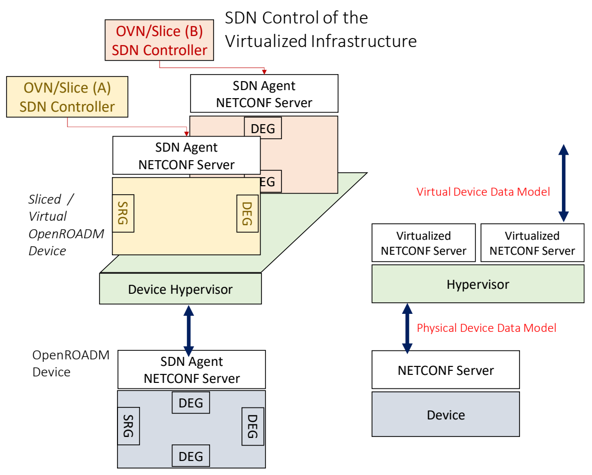


Fig. 8. Basic architecture of a device hypervisor

An OpenROADM hypervisor allows multiple virtual ROADM devices to behave independently and to be controlled by a dedicated SDN Controller using the same open interfaces and protocols as the physical device (see Fig. 8). A hypervisor partitions a device according to its standardized device data model, into multiple virtualized devices (e.g., virtualized ROADMs). This hypervisor ensures isolation and acts as a (restricted) NETCONF/YANG proxy to the physical device, so a per-OVN dedicated controller can control the OVN independently.

The role of the ROADM hypervisor is to coordinate access to the underling physical device agent, so each virtual device only sees and operates on a partial (restricted) view of the data model configuration and operational datastore.

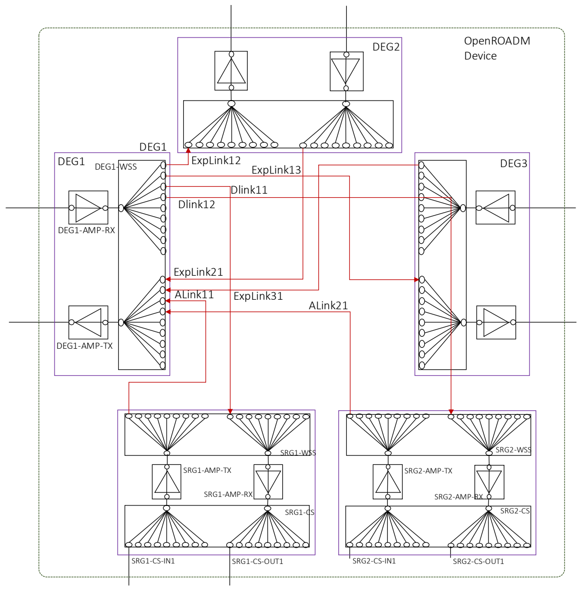


Fig. 9. Example of a 3-DEG, 2-SRG ROADM device corresponding to the physical ROADM device.

There are multiple degrees of freedom in defining an OVN/partitioning the devices: assigning a device's degrees (DEG) and SRGs, assigning the internal and external ports as well as internal and physical links and partitioning the usable spectrum. For example, Fig. 9 shows an example illustrative device and Fig. 10 a possible partition of such OpenROADM device into 2 logical virtualized ROADMs.

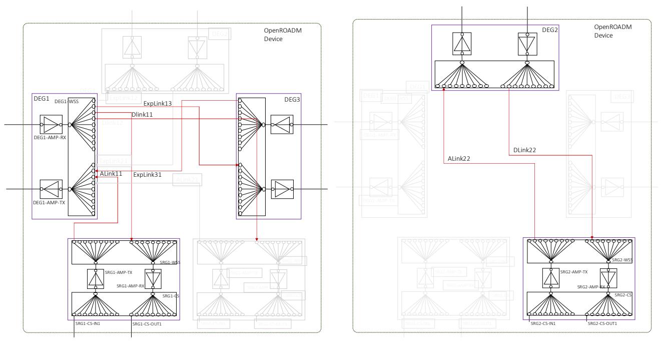


Fig. 10. Two virtualized ROADMs corresponding to a given partition of the underlying device. The left one is a 2-DEG, 1-SRG and the right one is a 1-DEG, 1-SRG one.

C. Optical Virtual Network Lifetime management

The OVN manager processes a request for an OVN instance and partitions the devices accordingly; configuring the hypervisors and, in our specific demonstration allocating a new SDN controller instance, configured to connect to the virtual devices. During the lifetime of the OVN, such SDN instance can provide end-to-end connectivity upon request between (virtualized) transceivers.

For each active partition, the OVN manager instantiates a virtualized NETCONF server and creates, dynamically, a *running* datastore encompassing only the elements included in the partition. Such XML encoded datastore is provisioned in the virtualized server. To illustrate the concept, consider Fig. 11: the tree on the left represents the configuration and operational data of a physical device. The one on the right for a virtualized ROADM only encompassing a subset of circuit packs (e.g. CP2), ports (port2, port3), degrees (DEG2, DEG4) and SRG1.

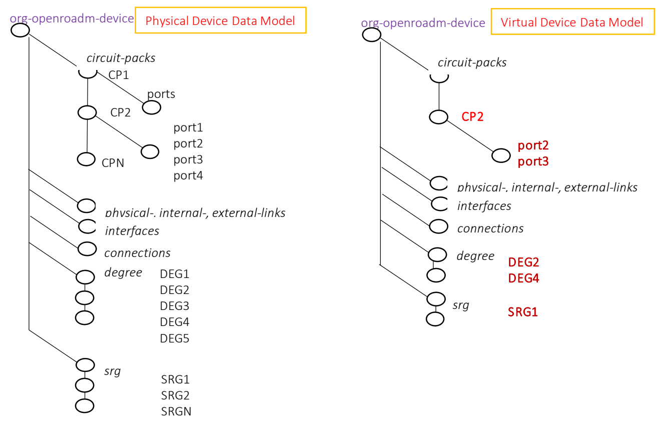


Fig. 11. Data nodes reflecting the physical device composition (left) and the virtualized one (right) as per a given partition.

From a workflow perspective, the process is as follows: each physical device is equipped with a NETCONF server/agent and a hypervisor. The hypervisor initially retrieves the operational and confirmation data of the device running datastore. Upon request, the OVN manager creates an OVN, translating the network request into a set of partitions of the individual devices. For each device partition, the manager allocates, via the hypoevisor) a dedicated virtualized NETCONF agent with a dynamically created datastore that reflects the partition and partial view corresponding to the virtualized device.

SDN Controllers interact with agents running in dedicated containers. NETCONF operations are processed by the NETCONF frontend (over the partial view of the device), and forwarded to the hypervisor.

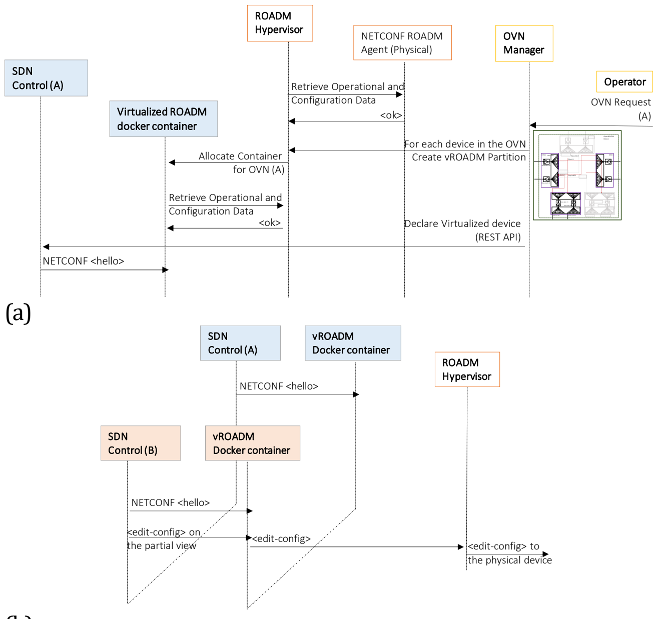

ROADM Hypervisor
NETCONF ROADM Agent (Physical)
OVN Manager
SDN Control (A)
Operator
Virtualized ROADM docker container
Retrieve Operational and Configuration Data
<ok>
OVN Request (A)
For each device in the OVN Create vROADM Partition
Allocate Container for OVN (A)
Retrieve Operational and Configuration Data
<ok>
Declare Virtualized device (REST API)
NETCONF <hello>
(a)
SDN Control (A)
vROADM Docker container
ROADM Hypervisor
NETCONF <hello>
SDN Control (B)
vROADM Docker container
NETCONF <hello>
<edit-config> on the partial view
<edit-config>
<edit-config> to the physical device
(b)

Fig. 12. Workflow to instantiate a OVN upon request from the Operator (a) and subsequent NETCONF operations from the SDN Controller (b)

D. Isolated (in-slice) service provisioning

As stated, during the lifetime of the OVN, the manager provides the OVN SDN controller with the credentials of the (virtual) devices, so NETCONF sessions are established and the devices' capabilities discovered (tunability and switching capabilities). NMC are set up upon request between two terminal devices, with terminal devices client ports are mapped to Service Interface Points (SIPs).

As stated in Sect 2.C , this triggers a routing and spectrum assignment process that finds the k-shortest path between the devices and performs first fit spectrum allocation. For the Terminal Device, a logical channel association is instantiated within the device between a client (transceiver) port and an optical channel component bound to the line port of the device. For each of the OpenROADM devices across the path, a ROADM internal connection is requested. Section 6 details the procedures for an express connection as defined by the OpenROADM MSA. Fig. 12 macroscopically illustrates both the OVN instantiation process, as well as the in-slice service provisioning process.

4. IMPLEMENTATION ASPECTS

A. SDN Controller TAPI North Bound Interface

The implementation of the TAPI 2.1 NBI interface allowing the query of the topology and the provisioning of connectivity services (i.e., the TAPI topology and connectivity models) relies on the existing ONOS Dynamic Configuration Subsystem (DCS). In short, the DCS allows YANG service models to be compiled and registered in the ONOS SDN controller automatically, in such a way that the underlying data nodes and Remote Procedure Calls (RPCs) can be accessed by TAPI clients (e.g. network orchestrators or end users) using a compliant RESTCONF implementation. SDN applications can register for events in the Dynamic Configuration subsystems such as, for example, the `create-connectivity-service` RPC that encompasses the endpoints of a point-to-point network media channel request [21]. Such endpoints refer to SIPs that are exported to TAPI clients in the TAPI context, which also includes a detailed topological view of the (virtualized, disaggregated) optical network, since Topological elements are added to the DCS dynamically upon discovery. Internally, the TAPI Connectivity Service Request is mapped to an Optical Connectivity intent, using ONOS intent framework. Optical Connectivity Intents reflect desired connectivity between two transceivers client ports. A path meeting the optical constraints is computed and the subsequent resource allocation is translated into per-device configuration instructions, using a generalized model of flow, as detailed next.

B. SDN Controller OpenROADM and OpenConfig drivers

The integration of OpenConfig terminal devices as well as OpenROADM devices into the ONOS SDN Controller involved the development of specific drivers in order to implement the abstracted *Behaviours* within ONOS optical network model. A *Behaviour* models a particular capability of abstracted interface of a device, which is used by the core ONOS components from an abstracted perspective.

In our particular case, both device drivers implement a set of key behaviours within the ONOS Optical Model [22], providing the required functionality using the NETCONF transport protocol.

The `DeviceDescriptionDiscovery` behavior is implemented in order to retrieve data about the device (identifiers, number of ports, port attributes, etc.) and allow the dynamic discovery of nodes, ports and links (for the specific case of OpenROADM external links). ONOS core systems are able to register the device into the DeviceManager and into the underlying Optical layer topology.

The `LambdaQueryBehaviour` is implemented in order to allow other subsystems to query which nominal central frequencies are available at a given device port, thus affecting end-to-end routing and Spectrum Assignment.

Finally, the implementation of the `FlowRuleProgrammable` behavior maps high level operations defined in terms of Flow Rules (getFlowEntries, applyFlowRules). Flow Rules define a set of criteria in order to match traffic and a set of actions to forward that traffic. They are involved in the management of device cross-connections and are translated into NETCONF edit-config operations to create the data node entries in the OpenROADM device.

The implementation of the driver can be obtained in [23].

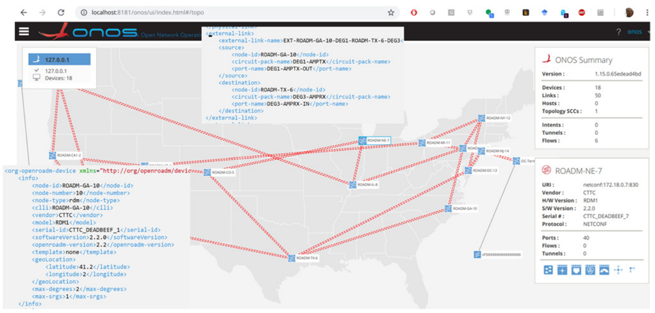
C. OpenROADM hypervisor

The OpenROADM hypervisor software is implemented in C++ using the ConfD framework [24], a management agent software for network elements. It provides a compliant NETCONF/ÝANG server implementation. When the server is configured, OpenROADM v2.2 YANG models are compiled into ConfD and the software is responsible for common aspects such as NETCONF support, data model validation, or integrity checks. A "*host*" instance of ConfD runs allowing direct access to the physical device, mapping NETCONF operations to hardware configuration operations, and uses a NETCONF datastore that corresponds to the physical components of the underlying device.

Next, a dedicated instance of ConfD is executed on a per virtual device basis (i.e., "*guest*" instances), with a NETCONF datastore constructed by the OVN manager as a partial view of the device, as detailed in Section 3.C. The pre-loaded configuration and operational data nodes correspond to the intended partition. ConfD behaves as a NETCONF agent/server frontend. Configuration operations are reflected in the virtualized ROADM datastore and then processed as events by the hypervisor, which forwards the operation to the physical device NETCONF server. The MAAPI and CDB application programming interfaces are used in this regard: a CDB socket is used to register subscriptions for data nodes of interest, and the library offers an API to read and write operational and configuration data. In particular, the agent processes asynchronous notifications corresponding to the creation, update or removal of data nodes corresponding to OpenROADM interfaces and roadm-connections., and NETCONF messages are sent to the host instance.

Consequently, a NETCONF client that operates on a virtualized ROADM can only see the partial view as defined by the datastore managed by its dedicated instance, ensuring isolation between different instances.

6. EXPERIMENTAL DEMONSTRATION



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

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

ONOS Summary

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

ROADM-NE-7

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

Fig. 13. ONOS GUI showing the NSFNet OpenConfig/OpenROADM topology, with 14 nodes and 42 unidirectional links.

For the experimental evaluation, we consider an emulated NSFNet topology that corresponds to the physical infrastructure. The considered topology encompasses 14 OpenROADM devices with 42 unidirectional links (corresponding to external-links in the OpenROADM device model), with each node having 1 SRG. Fig. 13 shows the topology from the ONOS GUI perspective. To illustrate potential partitions that can be created, Table 1 shows the number of main elements form the OpenROADM device model for the NSFNet as well as for simple 3 and 5 node partitions, directly obtained from the NETCONF running datastores.

OpenROADM topology	NSFNET	3node	5node
Number of Nodes	14	3	5
Number of Ports	1880	330	552
Internal Links	1624	288	480
Physical Links	312	42	74
External Links	42	6	10
Circuit Packs	270	42	72
SRGs	14	3	5
DEGs	42	6	10

Tab. 1. Number of OpenROADM elements in NSFNet and basic partitions.

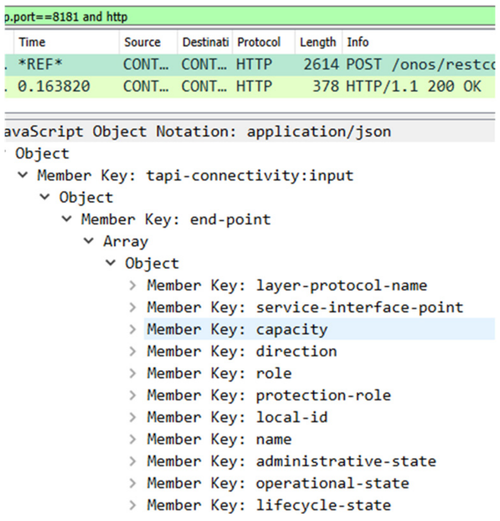
The experimental validation involves the creation of a subnet (OVN instance) in which a partition is defined selecting 10 nodes and a subset of the OpenROADM degrees, resulting on a sub-graph of the original topology. When the OVN is instantiated, virtualized agents running NETCONF servers with a partial view of the device are instantiated as Docker containers. Using the MAC VLAN Docker networking drivers, virtualized agents belonging to the same OVN instance are given IP addresses from the same (private) subnet, which are reachable from the per OVN dedicated ONOS SDN Controller instance. JSON encoded files are generated by the OVN manager (see Fig. 14) and uploaded to the SDN Controller using REST interfaces, so NETCONF sessions can be established by the controller.

```
"netconf:11.1.1.128:830": {
        "basic": {
                "driver": "openroadm",
                "name" : "ROADM",
                "locType" : "geo",
                      "latitude" : "45.8",
                      "longitude" : "9.57"
         },

         "netconf": {
                 "ip": "11.1.1.128",
                 "port": "830",
                 "username": "username",
                 "password": "password",
                 "connect-timeout": 20,
                 "reply-timeout": 25,
                 "idle-timeout": 30000
         }
}
```

Fig. 14. Sample JSON encoding used to register a virtualized logical device to the SDN controller via a dedicated REST interface.

Once the topology of the OVN has been discovered by the OVN SDN controller, the user (tenant) of the OVN can request the provisioning of connectivity services. For this, a TAPI request is sent involving two endpoints (SIPs) of the OVN. A RESTConf Remote Procedure Call (RPC) for the TAPI connectivity service encompasses the required information objects (Relevant photonic media layer, connectivity service endpoints, involved SIPs, requested capacity in GHz, etc.). Fig. 15 shows the Wireshark capture of the POST operation for the RPC.



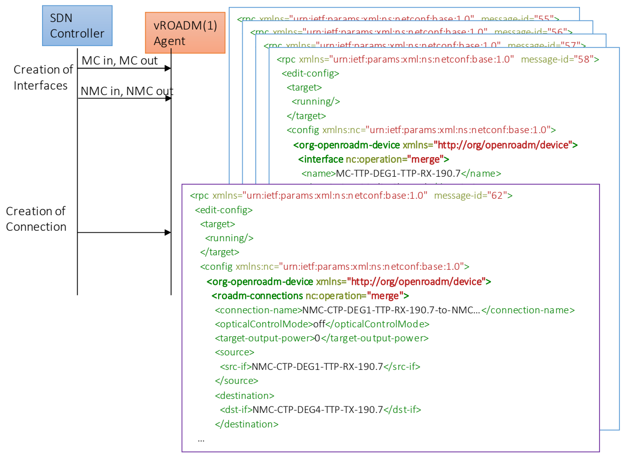
```
p.port==8181 and http

  Time       Source  Destinati Protocol  Length Info
. *REF*      CONT…   CONT…     HTTP      2614 POST /onos/restcc
. 0.163820   CONT…   CONT…     HTTP       378 HTTP/1.1 200 OK

avaScript Object Notation: application/json
 Object
  ⌄ Member Key: tapi-connectivity:input
    ⌄ Object
      ⌄ Member Key: end-point
        ⌄ Array
          ⌄ Object
            > Member Key: layer-protocol-name
            > Member Key: service-interface-point
            > Member Key: capacity
            > Member Key: direction
            > Member Key: role
            > Member Key: protection-role
            > Member Key: local-id
            > Member Key: name
            > Member Key: administrative-state
            > Member Key: operational-state
            > Member Key: lifecycle-state
```

Fig. 15. Wireshark capture of TAPI NBI Connectivity Request with the Connectivity Service RPC create-connectivity-service and endpoints.

As stated earlier, the SDN Controller ends up constructing flow rules that trigger the creation of ROADM connections on the virtualized devices. To provide an example, let us consider the provisioning of a unidirectional express connection supporting a 50 GHz signal centered at 190.7 GHz from degree 1 to degree 4. Given the fact that Optical Transmission Section (OTS) and Optical Multiplex Section (OMS) interfaces do not change dynamically, we can assume that such interfaces are pre-configured at the ROADM degrees. The actual provisioning thus involves the following steps: *i*) the creation of Media Channel (MC) interfaces supported over the OMS interfaces (by convention named e.g. MC-TTP-DEG1-TTP-RX-190.7) with a min-freq: 190.675 and max-freq: 190.725. *ii*) The creation of Network Media Channel (NMC) interfaces over the previously created MC interface, specifying a center frequency and slot width, which induces Connection Termination Points (CTPs). *iii*) The creation of the ROADM internal unidirectional connection, established between these two logical CTP interfaces i.e., from the interface `NMC-CTP-DEG1-TTP-RX-190.7` to `NMC-CTP-DEG4-TTP-TX-190.7`. The whole process is illustrated in Fig. 16; it is worth noting that the logical operation of constructing a cross-connection involves O(5) NETCONF `edit-config` messages for an express connection.

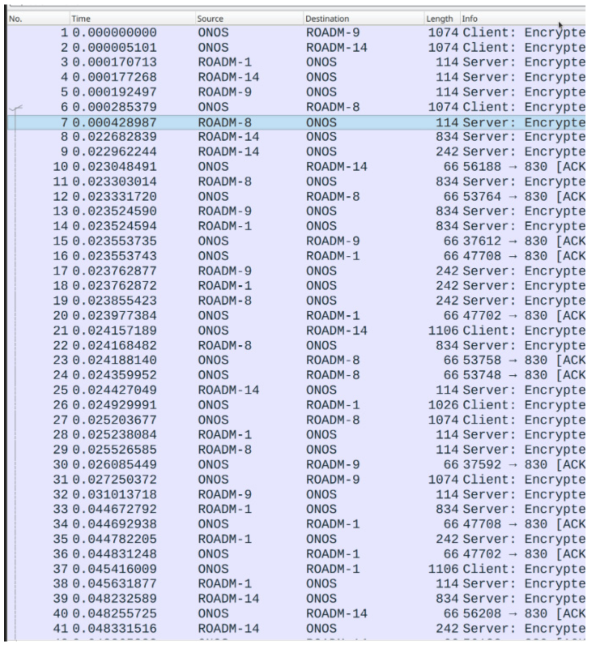


```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"  message-id="55">
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"  message-id="56">
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"  message-id="57">
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"  message-id="58">
 <edit-config>
  <target>
   <running/>
  </target>
  <config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
   <org-openroadm-device xmlns="http://org/openroadm/device">
    <interface nc:operation="merge">
     <name>MC-TTP-DEG1-TTP-RX-190.7</name>
```

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

Fig. 16. Sequence of NETCONF `edit-config` operations to create a connection, including the creation of supporting MC and NMC interfaces.

In our specific scenario, the NMC is set up across 4 virtualized ROADMs, and from the point of view of the controller, the process is completed in O(100) ms, as seen in the Wireshark capture (Fig. 17). Let us point out that the underlying physical devices are emulated and the service provisioning delay does not take into account the hardware configuration latency. This latency can be of the order of seconds/minutes, and largely dominates the end-to-end metric. From this perspective, the latency introduced by the control plane is almost negligible.



```
No.  Time         Source     Destination  Length Info
  1 0.000000000   ONOS       ROADM-9      1074 Client: Encrypte
  2 0.000005101   ONOS       ROADM-14     1074 Client: Encrypte
  3 0.000170713   ROADM-1    ONOS          114 Server: Encrypte
  4 0.000177268   ROADM-14   ONOS          114 Server: Encrypte
  5 0.000192497   ROADM-9    ONOS          114 Server: Encrypte
  6 0.000285379   ONOS       ROADM-8      1074 Client: Encrypte
  7 0.000428987   ROADM-8    ONOS          114 Server: Encrypte
  8 0.022682839   ROADM-14   ONOS          834 Server: Encrypte
  9 0.022962244   ROADM-14   ONOS          242 Server: Encrypte
 10 0.023048491   ONOS       ROADM-14       66 56188 → 830 [ACK
 11 0.023303014   ROADM-8    ONOS          834 Server: Encrypte
 12 0.023331720   ONOS       ROADM-8        66 53764 → 830 [ACK
 13 0.023524590   ROADM-9    ONOS          834 Server: Encrypte
 14 0.023524594   ROADM-1    ONOS          834 Server: Encrypte
 15 0.023553735   ONOS       ROADM-9        66 37612 → 830 [ACK
 16 0.023553743   ONOS       ROADM-1        66 47708 → 830 [ACK
 17 0.023762877   ROADM-9    ONOS          242 Server: Encrypte
 18 0.023762872   ROADM-1    ONOS          242 Server: Encrypte
 19 0.023855423   ROADM-8    ONOS          242 Server: Encrypte
 20 0.023977384   ONOS       ROADM-1        66 47702 → 830 [ACK
 21 0.024157189   ONOS       ROADM-14     1106 Client: Encrypte
 22 0.024168482   ROADM-8    ONOS          834 Server: Encrypte
 23 0.024188140   ONOS       ROADM-8        66 53758 → 830 [ACK
 24 0.024359952   ONOS       ROADM-8        66 53748 → 830 [ACK
 25 0.024427049   ROADM-14   ONOS          114 Server: Encrypte
 26 0.024929991   ONOS       ROADM-1      1026 Client: Encrypte
 27 0.025203677   ONOS       ROADM-8      1074 Client: Encrypte
 28 0.025238084   ROADM-1    ONOS          114 Server: Encrypte
 29 0.025526585   ROADM-8    ONOS          114 Server: Encrypte
 30 0.026085449   ONOS       ROADM-9        66 37592 → 830 [ACK
 31 0.027250372   ONOS       ROADM-9      1074 Client: Encrypte
 32 0.031013718   ROADM-9    ONOS          114 Server: Encrypte
 33 0.044672792   ROADM-1    ONOS          834 Server: Encrypte
 34 0.044692938   ONOS       ROADM-1        66 47708 → 830 [ACK
 35 0.044782205   ROADM-1    ONOS          242 Server: Encrypte
 36 0.044831248   ONOS       ROADM-1        66 47702 → 830 [ACK
 37 0.045416009   ONOS       ROADM-1      1106 Client: Encrypte
 38 0.045631877   ROADM-1    ONOS          114 Server: Encrypte
 39 0.048232589   ROADM-14   ONOS          834 Server: Encrypte
 40 0.048255725   ONOS       ROADM-14       66 56208 → 830 [ACK
 41 0.048331516   ROADM-14   ONOS          242 Server: Encrypte
```

Fig. 17. (Partial) Wireshark capture of NETCONF/SSH exchanges between controller and agents, for the provision of a Network Media Channels over an OVN involving 4 out of the 10 node- NSFNet topology.

Finally, a second metric of interest is the control plane throughput. Fig. 18 shows the I/O plot of traffic in packets per second, including not only the NETCONF messages but also the SSH/TCP overhead. In any case, the bandwidth capacity required to support NETCONF configuration operations is relatively low.

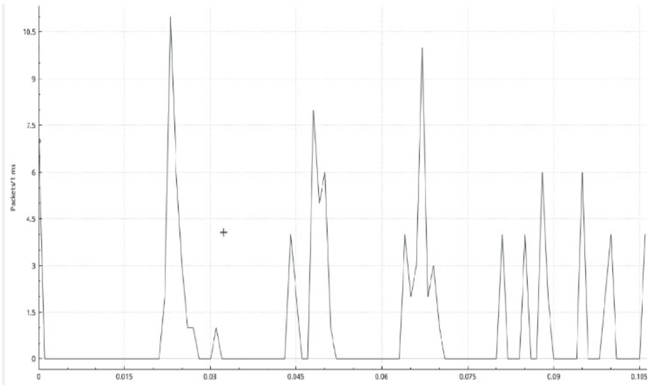


Fig. 18. Wireshark I/O plot in pps from the SDN Controller for the OVN, showing the traffic variations during the O(100) ms of the provisioning window,

7. CONCLUSIONS

The adoption of model driven developments, the availability of open data models for an increasing range of optical devices and the overall major softwarization trends in the scope of SDN control of optical networks is enabling an unprecedented degree of flexibility and efficient resource usage. With a unified modeling approach (e.g. relying on common data model languages, transport protocols such as NETCONF/RESTCONF) coupled to a steady increase in devices programmability, SDN control planes can evolve to match the requirements related to network sharing, virtualization and slicing, including ultimately empowering end users to control their allocated slices. This is particularly important in the scope of disaggregated optical networks, which are a main use case for open data models. In particular, we have briefly summarized the use of TAPI interfaces and OpenROADM device models.

This work has addressed and demonstrated the virtualization of an open and disaggregated optical network, along with the provisioning of network media channels within a given OVN instance, relying on device hypervisors and standard data models. The implemented extensions to ONOS SDN controller for OpenConfig or OpenROADM can be used regardless of the infrastructure is real or virtual. The validated proof-of-concept demonstrates the setup of a connection as an optical connectivity intent, largely satisfying the target values and demonstrating the feasibility of a model driven SDN control for a virtualized network.

One of the limitations of the approach is that the actual underlying partition (down to Degrees, SRGS, circuit-packs, ports and optical spectrum) is quasi-static, bound to the lifetime of the OVN with a given resource is exclusively assigned to a single virtual instance. The extension of the approach for a more flexible resource sharing is left for further study.

Funding Information. Work funded by the EC H2020 project METRO-HAUL (761727) and the Spanish AURORAS (RTI2018-099178-BI00) project.

Acknowledgment. We thank the members of the ONF ODTN and Metro-Haul projects for their insightful comments and suggestions concerning the OpenROADM v2.2 ONOS driver implementation as well as the design of the ConfD-based agent(s).

References

1. Next Generation Mobile Networks (NGMN) Alliance White Paper on 5G **https://www.ngmn.org/5g-white-paper/5g-white-paper.html**
2. Li, X., Casellas, R., Landi, G. et al, "5G-Crosshaul Network Slicing: Enabling Multi-Tenancy in Mobile Transport Networks", IEEE Communications Magazine, special issue on Network Slicing in 5G systems, August 2017, Vol. 55, No. 8, August 2017.
3. R. Vilalta, A. Mayoral López-de-Lerma, R. Muñoz, R. Martínez, R. Casellas, "Optical Networks Virtualization and Slicing in the 5G era", in *Optical Fiber Communications Conference* (OFC2018), 11-15 March 2018, San Diego, CA, USA
4. R. Nejabati, E. Escalona, S. Peng and D. Simeonidou, "Optical network virtualization," 15th International Conference on Optical Network Design and Modeling - ONDM 2011, Bologna, 2011, pp. 1-5.Optical Network Virtualization
5. Y. Li, X. Chen, N. Hua, and X. Zheng, "A Novel Virtual Optical Network Embedding Strategy for Optical Network Virtualization," in Advanced Photonics for Communications, OSA Technical Digest (online) (Optical Society of America, 2014), paper NT1C.3.
6. L. Gong and Z. Zhu, "Virtual Optical Network Embedding (VONE) Over Elastic Optical Networks," J. Lightwave Technol. 32, 450-460 (2014)
7. N. Shahriar, S. Taeb, S. Chowdhury, M. Tornatore, R. Boutaba, J. Mitra, M. Mahdi, "Achieving a Fully-Flexible Virtual Network Embedding in Elastic Optical Networks", in *INFOCOM Conference* 10.1109/ INFOCOM.2019.8737601.
8. E. Riccardi, P. Gunning, O. González de Dios, et al., "An Operator view on the Introduction of White Boxes into Optical Networks", J. Lightwave Technol., 36, (15), pp. 3062-3072 (2018).
9. METRO-HAUL: High bandwidth, 5G Application-aware optical network, with edge storage, compUte and low Latency, **http://metro-haul.eu**
10. The Open Disaggregated Transport Network project, ONF, https://www.opennetworking.org/odtn/
11. OpenConfig project and data models http://openconfig.net and https://github.com/openconfig/public/tree/master/release/models
12. The Open ROADM Multi-Source Agreement (MSA) http://www.openroadm.org
13. R. Casellas, A. Giorgetti, R. Morro, R. Martínez, R. Vilalta, R. Muñoz, "Enabling Network Slicing Across a Disaggregated Optical Transport Network ", in *Optical Fiber Communications Conference* (OFC2019), 3-7 March 2019, San Diego, CA (USA). 2019.
14. Open Networking Foundation (ONF), Transport API (TAPI) overview https://www.opennetworking.org/wp-content/uploads/2017/08/TAPI-2.0-Updates-Overview.pdf
15. Open Networking Foundation (ONF), Transport API project https://wiki.opennetworking.org/display/OTCC/TAPI
16. R. Casellas, R. Martinez, R. Vilalta, R. Munoz, "Control, management, and orchestration of optical networks: Evolution, trends, and challenges," J. of Lightwave Technol., vol. 36, (7), pp. 1390–1402 (2018)
17. D. Ceccarelli, Y., Lee, editors, "Framework for Abstraction and Control of TE Networks (ACTN)", IETF RFC8453, august 2018.
18. R. Casellas, R. Vilalta, R. Martínez, R. Muñoz, "Experimental Validation of the ACTN architecture for flexi-grid optical networks using Active Stateful Hierarchical PCEs", in Proc. 19 International Conference on Transparent Optical Networks (ICTON2017), 2-6 July, 2017, Girona, Spain, July 2017.
19. Y. Lee, D. Dhody, editors, "A YANG Data Model for VN Operation", draft-ietf-teas-actn-vn-YANG-05, work in progress (2019)
20. ONF TAPI Virtual Network YANG model, https://github.com/OpenNetworkingFoundation/TAPI/blob/develop/YANG/tapi-virtual-network%402019-03-31.YANG
21. ODTN Service Application ONOS Release 2.2 https://github.com/opennetworkinglab/onos/tree/master/apps/odtn
22. ONOS optical Information Model (Wiki) https://wiki.onosproject.org/display/ONOS/Optical+Information+Model#OpticalInformationModel-DisaggregatedROADMmodel
23. ODTN OpenROADM v2.2 NetConf drivers. https://gerrit.onosproject.org/#/c/22106/
24. ConfD, management agent software framework for network elements https://www.tail-f.com/confd-basic Accessed: 2019-4-12.

Ramon Casellas (SM'12) graduated in 1999 from UPC, Barcelona and from the ENST, Paris, with an Erasmus student exchange program grant. He completed a PhD degree in 2002 and worked at the ENST as an Associate Professor. In March 2006, he joined the CTTC Optical Networking Dept., where he is currently holding a Senior Researcher position. His research interest areas include Traffic Engineering; Control and Management of Optical Transport Networks (GMPLS/PCE, SDN and NFV). He has been involved in several international, national and industrial R&D projects and has co-authored 5 book chapters, over 200 journal and conference papers and 5 IETF RFCs. He is an ONF contributor and member of the ONF ODTN project.

Alessio Giorgetti received the Ph.D. degree from Scuola Superiore Sant'Anna (SSSA), Pisa, Italy, in 2006. In 2007, he has been visiting scholar at Centre for Advanced Photonics and Electronics, University of Cambridge, UK. Currently, he is an Assistant Professor at SSSA. His research interests include: optical network architecture and control plane, industrial network design, software defined networking. He is author of more than 100 publications including international journals, conference proceedings, and patents.

Roberto Morro received a Dr. Ing. degree in electronic engineering from University of Genoa (Italy) in 1988. After a six year experience with Marconi as a testing engineer, he joined TIM (at that time CSELT) in 1995 where he is currently working in the technology evolution and innovation unit. He has been actively working in several European funded projects and in the related technology transfer inside the TIM group. His research interests include network control, SDN, NFV, focusing especially on multi-layer (IP over optics), multi-domain and traffic engineering aspects.

Ricardo Martínez (SM'14) received Ph.D. degree in 2007 in telecommunications engineering, from the Universitat Politècnica de Catalunya, Barcelona, Spain. He has been actively involved in several public-funded (national and European Union) research and development projects as well as industrial technology transfer projects. Since 2013, he is Senior Researcher of the Communication Networks Division at CTTC, Castelldefels, Spain. His research interests include service and resource (cloud and network) orchestration, control and network management architectures for packet and optical transport networks in multiple network segments.

Ricard Vilalta (SM'17) has a telecommunications engineering degree (2007) and Ph.D. degree (2013), at UPC, Spain. He is senior researcher at CTTC, in the Communication Networks Division. His research is focused on SDN/NFV, Network Virtualization and Network Orchestration. He has been involved in international, EU, national and industrial research projects, and published more than 200 journals, conference papers and invited talks. He is also involved in standardization activities in ONF, IETF and ETSI.

Raül Muñoz (SM'12) graduated in telecommunications engineering in 2001 and received a Ph.D. degree in telecommunications in 2005, both from UPC-BarcelonaTech, Spain. He is Head of the Optical Networks and Communications Networks Division Manager at CTTC (Barcelona, Spain). He has coordinated the H2020-MSCA-ITN ONFIRE project and the EU-Japan FP7-ICT STRAUSS project. He has published over 250 journal and international conference papers. His research interests include control and service management architectures (SDN, NFV, network slicing) for disaggregated optical/packet transport networks with edge/cloud computing.