



Article

Emulating Software-Defined Disaggregated Optical Networks in a Containerized Framework

Francisco-Javier Moreno-Muro ^{1,*} , Miquel Garrich ¹, Ignacio Iglesias-Castreño ¹, Safaa Zahir ¹ and Pablo Pavón-Mariño ^{1,2} 

¹ Department of Information and Communication Technologies, GIRTEL Group, Universidad Politécnica de Cartagena, Plaza del Hospital S/N, 30202 Cartagena, Spain; miquel.garrich@gmail.com (M.G.); iglesias.castreno@ignacio.xyz (I.I.-C.); zahie.safaa@gmail.com (S.Z.); pablo.pavon@upct.es (P.P.-M.)

² E-Lighthouse Network Solutions, 30202 Cartagena, Spain

* Correspondence: javier.moreno@upct.es

Abstract: Telecom operators' infrastructure is undergoing high pressure to keep the pace with the traffic demand generated by the societal need of remote communications, bandwidth-hungry applications, and the fulfilment of 5G requirements. Software-defined networking (SDN) entered in scene decoupling the data-plane forwarding actions from the control-plane decisions, hence boosting network programmability and innovation. Optical networks are also capitalizing on SDN benefits jointly with a disaggregation trend that holds the promise of overcoming traditional vendor-locked island limitations. In this work, we present our framework for disaggregated optical networks that leverages on SDN and container-based management for a realistic emulation of deployment scenarios. Our proposal relies on Kubernetes for the containers' control and management, while employing the NETCONF protocol for the interaction with the light-weight software entities, i.e., *agents*, which govern the emulated optical devices. Remarkably, our agents' structure relies on components that offer high versatility for accommodating the wide variety of components and systems in the optical domain. We showcase our proposal with the emulation of an 18-node European topology employing Cassini-compliant optical models, i.e., a state-of-the-art optical transponder proposed in the Telecom Infrastructure Project. The combination of our versatile framework based on containerized entities, the automatic creation of agents and the optical-layer characteristics represents a novel approach suitable for operationally complex carrier-grade transport infrastructure with SDN-based disaggregated optical systems.

Keywords: disaggregated optical networks; software defined networking; optical network emulation



Citation: Moreno-Muro, F.-J.; Garrich, M.; Iglesias-Castreño, I.; Zahir, S.; Pavón-Mariño, P. Emulating Software-Defined Disaggregated Optical Networks in a Containerized Framework. *Appl. Sci.* **2021**, *11*, 2081. <https://doi.org/10.3390/app11052081>

Academic Editor: Fabio Cavaliere

Received: 1 February 2021

Accepted: 22 February 2021

Published: 26 February 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The rapid growth of Internet traffic that we are currently experiencing [1,2] is based on the expansion of cloud services and the huge amount of traffic supported by the content delivery networks (CDNs) [1]. This clearly increases congestion issues in communication networks, with particular emphasis in the core and backbone segments [3]. Furthermore, the global crisis of COVID-19 has revealed the need for remote communications in both the social and business spheres [2,4]. This crisis has become an incipient increase in video-call-related traffic through the use of applications such as Zoom [5], Microsoft Teams/Skype [6] or Cisco Webex [7]. Finally, these challenges are compounded by the need to meet the performance specifications for the fifth generation Internet (5G), where it is planned to obtain (a) a scalable management environment that enables the dynamic deployment of cloud-based applications, (b) a reduction of approximately 20% in operational costs and (c) total latency to the user of less than 1 ms [8].

The combination of all these challenges requires drastic changes in the way transport network resources are controlled and managed, moving away from traditional network management techniques. Under this umbrella, software defined networking (SDN) and

network function virtualization (NFV) emerge as key technologies to address the above challenges in an efficient way. In this work we will put the light on SDN.

SDN is becoming a consolidated technology for network management that encompasses a set of techniques aimed at the management of networks that focus mainly on one basic principle: decoupling the decisions made at the control plane with respect to the actions taken at the data plane [9]. This principle allows unprecedented degrees of flexibility in the system, because the network switches and routers become simple forwarding devices that send and receive traffic, while the logical control of the network can easily be centralized on an external controller [10]. Conversely to traditional vendor-locked communication networks, SDN technology brings to network operators, among other benefits, the capability to: respond to challenges (such as big traffic fluctuations due to major sport or public events), operate their network in a cost-effective manner, evolve the network infrastructure, foster innovative solutions [10].

Relevant for optical networks, SDN is a powerful enabler to control and manage the wide variety of network elements with particular photonic transmission and switching characteristics inherent of the optical domain [11]. In this context, software-defined optical networks (SDONs) aim to exploit the flexibility of SDN control to support network applications with an underlying optical network infrastructure [12]. Within SDON, software models dedicated to the characterisation of optical elements, devices and systems are of utmost significance to enable the creation of *software agents*. In this regard, an agent is assumed to be a light-weight software that translates the necessary commands to permit communication between the SDN controller and the optical device placed in the optical data plane. Consequently, agents play a relevant role in disaggregated optical networks.

Horizontal disaggregation in optical network is aimed at decomposing the optical data in its single components [13]. Disaggregated optical networks (DONs) leverage this idea to be presented as a new technology in optical networks and brings synergies with the SDN paradigm. Under this umbrella, several cooperative projects, such as Open Config [14], Open ROADM [15] or Telecom Infra Project [16], have been arising for the last years, with the target of providing unified models of optical devices. Thus, allowing the deployment of complex optical network architectures through the use of vendor-independent devices. This fact is remarkable because it avoids one of the most challenging traditional issues in optical network implementations, the vendor islands. The inherent independency of vendor equipment obtained in the aforementioned projects is mainly based on using a specific modelling language, namely, the Yet Another Next Generation (YANG) one [17]. YANG offers the possibility to consolidate in a common language and a catalogue of rules the definition of the main characteristics of device models that can be used in software agents. In SDN-based DONs the communication between the agents generated based on YANG models and the SDN controller (via South Bound Interface, SBI, in the canonical SDN architecture) usually relies on the NETCONF protocol [18]. The YANG/NETCONF symbiosis provides the ideal breeding ground not only for the development of optical devices that facilitate vendor interoperability but also for the development of software tools that emulate the real behaviour of such devices, which is basic in the early stages of the development of new devices and network planning. Further details on the YANG/NETCONF usage for optical networks and additional alternatives for SBIs can be found in [19].

1.1. State of The Art

The literature has a growing number of works focused on SDN DON and optical network emulation. However, studies that consider both topics from the application layer to the data plane are scarce. One interesting starting point to link SDN and DON is to put the light on the SDN controller. Some works like [13,20,21], embrace this approach, first by proposing a model-driven design of a SDN controller and, second, upgrading the functionalities of the SDN controller ONOS to support disaggregated optical networks. Such new capabilities include YANG models compatible with NETCONF-based SBIs that

communicate with the optical devices. In [22], this idea is expanded to not only be applied to the controller but also to a high level of abstraction that comprises the control plane in multi-domain disaggregated optical networks. Furthermore, a monitoring and data analytics architecture is proposed in [23] to monitor optical disaggregation.

On the other hand, several studies focus on the data plane, precisely on modelling devices in the context of DON. For instance, YANG models and their implementation, via agents, of (SDN-enabled) sliceable bandwidth/bitrate variable transceivers (S-BVTs) have been presented for partially [24] and fully disaggregated optical networks [25]. One of the most important actors in optical networks is the reconfigurable optical add/drop multiplexer (ROADM) that implements the add and drop capabilities in optical nodes. They are addressed within the DON scope in [26,27]. Additionally, focusing on the data plane, [28] reports a gap analysis on physical-layer parameters of YANG models available in the OpenConfig, OpenROADM and OpenDevice projects for estimating the quality of transmission (QoT). In addition, this work is complemented by a proof of concept focusing on the reception of the relevant parameters from an OpenConfig muxponder in the application layer for planning purposes.

Relevant in the context of data-plane initiatives, it is worthwhile mentioning GNPY [29]: the open-source planning tool for the data plane in optical networks. GNPY is based on the Gaussian Noise (GN) model [30], and reported validation results in large-scale optical transmission testbeds [31]. In particular, [31] reports the GNPY validation in a mixed-fibre test-bed at Microsoft labs in which transponders of eight different manufacturers operated in the C-band reaching a propagation distance of 1945 km. Relevantly, GNPY has also been considered as an assistance tool for the computation of impairments in optical-layer paths on top of SDN controllers [32]. Nonetheless, these efforts are scarce in the literature because optical-layer awareness is commonly considered as stand-alone planning tools separated from operational aspects addressed by SDN controllers.

The next stage towards emulating a DON environment is device virtualisation that exploits the benefits provided by YANG models of devices, which is also explored in the community. A network virtualisation architecture for open (partially) disaggregated network uses the concept of device hypervisor in virtual networks by using OpenROADM data models [33]. Furthermore, another common approach for adding virtualization to DON-based devices is to implement their related agent in a virtual machine (VM)/container that can emulate a realistic behaviour of the optical device. This eases the instantiation and the management of the virtual entities with a computer or a server. Those VMs/containers must be configured to enable the SBI communication implementing the interaction with the SDN controller [13].

Finally, it is also worthwhile mentioning comparable initiatives to our proposal such as the software-defined packet-optical network emulator [34], which is based on an optical extension of the popular packet-oriented network emulator Mininet [35]. Relevantly, ref. [34] proposes a software suite capable of emulating optical-layer QoT performances such as OSNR, gOSNR and BER, and report its impact to the SDN controller. This emulation is based on augmenting packet in traditional Mininet instances with wavelength/channel information, which is used in modified instances of Open vSwitch. In fact, the entire Mininet-Optical emulation relies on a software structure that invokes the Open vSwitch instances as Python processes. Consequently, this approach diverges from realistic deployment/production scenarios in which optical systems are managed/controlled by its own light-weight software agent in dedicated physical or virtual resources.

1.2. Motivation and Previous Work

The state of the art in SDN DON and optical devices' emulation reveals that the community is pushing considerably in this direction. YANG-based models in accordance with the NETCONF protocol are efficient approaches towards minimizing or even removing the drawbacks of the so-called vendor islands. Thus, there are ongoing projects to provide unified models for optical devices. However, to the best of our knowledge, optical device

emulation is in its early stages. Only some of the analysed works assume a scenario where SDN DON emulation tackles from the highest application layer down to the data plane. Handing over control and management of emulation to the application layer will permit a more precise optical network design and an automatic deployment of software agents related from device catalogue at the request of the user. Moreover, as pointed out in the data-plane initiatives in the previous subsection, interactions between realistic agent-based SDN implementations and data-plane models are scarce. In fact, the challenge of these initiatives relies on the combination of physical-layer modelling commonly devoted to off-line planning tools with operation-based systems such as SDN controllers. The complementary nature of those systems makes the development of a DON emulator a difficult and challenging task. Such context is the starting point that motivates this paper with the aim at contributing to achieve a such challenging goal.

This work is a significant follow-up of an open line started in [36], in which we reported our preliminary progress towards an optical emulation framework built on automatic agent creation. In particular, ref. [36] focused on a specific use case of an 18-node European topology with nodes based on open packet transponder (Cassini [37]) in which we were able to modify parameters such as optical launch power, wavelength, and modulation format. Here, we extend that work and present a substantial evolution of our Containerized Framework for emulating SDN-DONs with special attention to the coexistence of the Kubernetes-based [38] master nodes server-client architecture and a new automatic agent creation framework using NETCONF in the SDN controller-optical agent interaction irrespective of the YANG model used. Additionally, to test the practical feasibility of our proposal, a proof-of-concept is also presented.

1.3. Structure of the Paper

The rest of the paper is structure as follows. In Section 2, all the software tools needed, and the methodology followed are exposed. Section 3 is aimed at showing the proof-of-concept proposed to evaluate the proposal. Section 4 reports the results obtained with our framework accompanied with their discussion. Concluding remarks and future directions of our work are highlighted in Section 5.

2. Proposed Architecture for SDN-DON Emulation

In this section, we present all the details required to define and configure our proposal for the SDN DON emulated framework. First, we expose a summary of the previous work that feeds this proposal. Then, we present the general architecture that permits the automatic deployment of emulated SDN DON, the core of this work. This section concludes with configuration considerations that are required to set up the framework.

2.1. Legacy Work

In the previous work presented in [36], we reported our preliminary advancements towards an optical emulation framework for automatic creation of agents in SDONs.

As seen in Figure 1a, this architecture comprises three main elements: the ONOS SDN controller [39]; a pool of software agents that emulate the interaction with optical devices, in this case, Cassini transponders; and the API gateway.

ONOS is chosen to play the role of the SDN controller, and is configured with the required modules and YANG drivers to interact with the optical devices or agents in the data plane via NETCONF protocol. A pool of agents emulates a realistic environment of a DON data plane; those agents are implemented in docker containers and they are created automatically by the API gateway REST-based interface according to the topology specification given by ONOS.

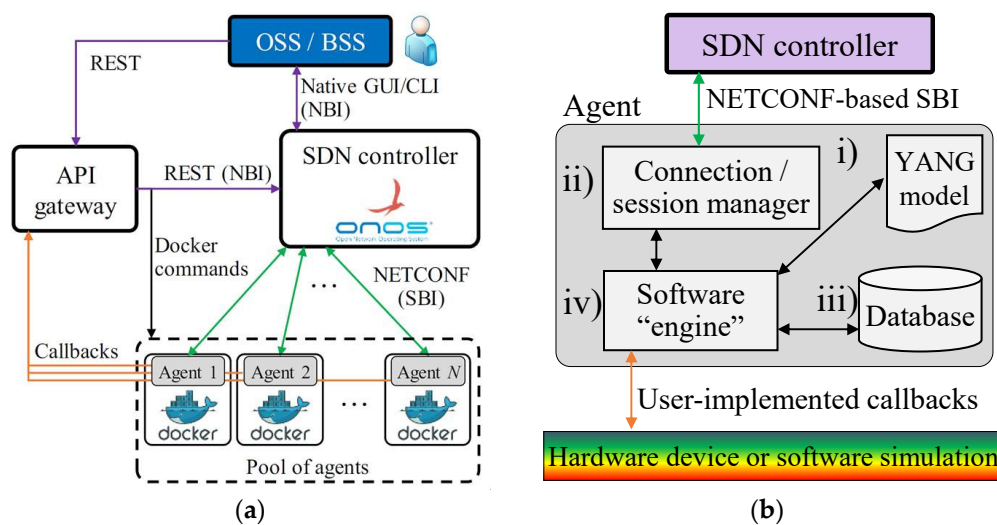


Figure 1. Architectural details of the previous work [36] (a) General overview of the architecture for automatic creation of agents in a software-defined optical networks (SDNs) environment; (b) Detailed view of the components in an optical network agent.

The other key point of this work is the structure of the optical network agent, depicted in Figure 1b. The software agents in the pool are virtualised entities within their respective Docker container. In addition, each agent includes four components: (i) the OpenConfig YANG model of Cassini transponders, (ii) the connection manager (i.e., the Netopeer2 NETCONF server) that enables connectivity with the controller, (iii) a database to store the configuration files and (iv) a software engine to detect and interact with changes in the configuration files. It is relevant to mention that the last three elements belong to the same subsystem, a Sysrepo framework aimed at automatic creation of YANG-based agent configuration. The Netopeer2 server is built in the Sysrepo framework and it provides the adequate NETCONF-based server for interacting with the SDN client. The Netopeer2-Sysrepo symbiosis provides benefits in the automatic creation of YANG agents but they have a closed and short catalogue of YANG-based optical device models that limits the scalability of a SDN DON emulation framework.

2.2. Advanced Automatic Deployment of Emulated SDN DON

The main target of this work is to enhance the dynamicity and the scalability of the automatic deployment of an emulated SDN DON environment. To achieve this challenging objective, we leverage the previous work focusing on two major and significant changes: (i) automatic container management by Kubernetes and (ii) a novel scalable agent definition.

2.2.1. General Architecture

Figure 2a showcases the general architecture of our proposal for an advanced automatic deployment of an emulated SDN DON framework. As it can be seen in the figure, the main difference with the legacy work is to leverage Kubernetes functionalities to manage and automatize the creation of agents, thus, minimizing the dependency of the API gateway. The other key contribution is the novel structure of the optical agent, shown in Figure 2b, facilitating the scalability and the emulation of a wide catalogue of YANG-based optical models. Further details of the structure of the agent are elaborated in next subsections.

This architecture is targeted to maximize the automation and the scalability of the emulation framework trying to exploit the benefits of a dynamic management and orchestration of the containerized optical agents. Moreover, two architectural decisions in our implementation deserve further description. First, it is worthwhile mentioning that we choose to employ docker containers because of they are more lightweight in terms of resources than VMs. We leverage on the fact that containers share the operating systems

whereas VMs aim at emulating virtual hardware. Given that docker containers share the operating systems, docker applications consume a portion of the resources compared to a VM. Second, our architecture considers one docker container per agent for a realistic emulation of a carrier-grade deployed scenario. In particular, telecom operators commonly deploy control architectures that handle distinct geo-located optical nodes with dedicated computing resources for service deployment (e.g., in the edge computing paradigm) while also devoting part of those resources for the control of the network infrastructure. In this context, optical resources are managed and operated with a per-node computing element that receives and sends commands from/to the SDN controller while interfacing the hardware at the data plane. Hence, our proposal aims at emulating such conditions with an independent and isolated entity, i.e., docker container, per software agent as in the case of distinct geo-located optical nodes.

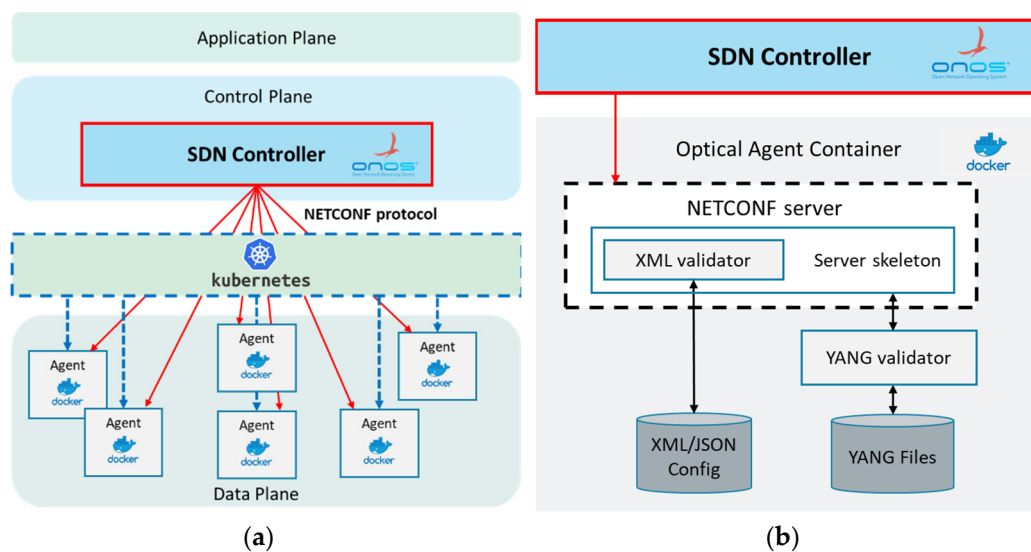


Figure 2. Architectural details of the proposal: (a) General overview of the architecture for the Advanced Automatic Deployment of an emulated software-defined networking (SDN) disaggregated optical network (DON) framework; (b) Schema of the elements in a generic optical network agent.

2.2.2. Kubernetes-Based Agent Management

In this work we propose to manage the agents' docker containers through Kubernetes. The choice of Kubernetes is based on the necessity of providing scalability and intelligent computational management in order to support large scale SDN DON emulated scenarios. Kubernetes has a wide range of benefits for virtualised systems that could help to accomplish the expected targets, e.g., service discovery and load balancing, storage orchestration, automated rollouts and rollbacks or self-healing just to mention a few. Moreover, Kubernetes is currently one of the most popular DevOps tools for container management, hence with a large support in the community [38]. Additionally, Kubernetes and Docker are complementary technologies, Kubernetes orchestrates one or more hosts that run containers, and Docker is the technology that starts, stops, and also manages those containers. In our model, Docker is a low-level technology orchestrated and managed by Kubernetes.

To complement the justification about why Kubernetes is key in our proposal, we present a representative use case. Consider a need to emulate a full SDN-based DON scenario assuming realistic functionalities, with multiple optical nodes and at each node multiple optical devices such as, transponders, wavelength selective switches, EDFA amplifiers or optical channel monitor. Just considering a 25-node optical network and for these four types of devices, $25 \times 4 = 100$ agents are needed, in the most favourable case assuming only 1 device per type and node (in reality this number is much higher). For

this case alone, managing 100 agents instantiated in virtual machines or virtual containers without intelligent management, the scenario is completely unaffordable for one single personal computer, computationally speaking. This is where Kubernetes comes in; this tool not only provides intelligent management of containers (agents), but its automatic load balancing and the minimization of the usage of computational resources, both features related to scalability, among others, are basic to allow the emulation of realistic and large-scale SDN DON scenarios. Thus, one of the major benefits that Kubernetes brings to our work is that it allows the emulation of complex SDN DON scenarios running on a single laptop, this way exploiting the potential usability and exploitation of our framework (e.g., teaching, academia, SDN DON pre-deployment phases, etc.).

From an implementation perspective, we follow a traditional master nodes approach. A Kubernetes cluster is made up of one master and one or more nodes, as can be seen in Figure 3. Both master and nodes are Linux hosts that run on anything from VMs, bare metal servers, to private and public cloud instances.

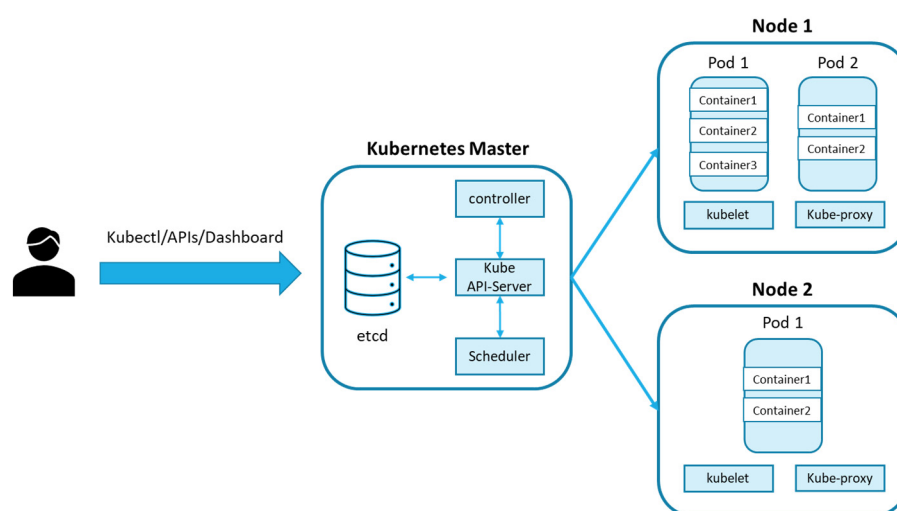


Figure 3. The Kubernetes master nodes-based architecture.

The Kubernetes master is composed of a set of small services that conform the control plane of the cluster and it can be externally accessed by the *kubectl* service, a RESTful API or a graphical dashboard, all managed by the *kube* API-server. The configuration and status of the cluster gets persistently stored in the *etcd*, while the controller implements the functionalities for controlling the Kubernetes cluster, such as, the node controller or the endpoints controller. Finally, the Kubernetes master offers the possibility to monitor new workloads and assign them to the nodes.

Still regarding Figure 3, the nodes can be seen as clients in a server–client architecture, with the master as server. The nodes are where the containers are hosted in correspondence to the agents in our framework. Within the nodes, there are independent spaces to run the containers, called pods. One node can have several pods and in each pod several containers. This approach is totally in line with network slicing, essential for 5G-aware deployments [40], being this fact another solid reason to choose Kubernetes as agent manager in a SDN DON emulation framework. Note that the server–client architecture, in correspondence with the Kubernetes master nodes (also illustrated with blue dashed arrows in Figure 2a), coexists with the client–server structure used in NETCONF protocol in correspondence with SDN controller–optical agent container (illustrated in solid red arrows in Figure 2).

Concerning Kubernetes integration as part of the Advanced Automatic Deployment of Emulated SDN DON oversees agents’ management. First, the configuration for creating containers (e.g., agents) can be performed by three different ways: via script by using the *kubectl* service, manually from dashboard or remotely through the Kubernetes API.

Once the containers are up and running in one or multiple pods, the Kubernetes Master is in charge of managing the computational resources according to the workload and the specifications stated in the configuration phase.

2.2.3. Versatile Agent Definition

The second major contribution of this work is a novel definition of versatile optical agents that can be automatically instantiated. We present a generic definition of agents suitable not only for specific YANG models belonging to specific projects (e.g., Telecom Infra Project [37]) but also for all optical models that can be defined using the YANG language. This novel structure is decisive in the sense that a user can evaluate scenarios with a potentially infinite number of optical devices under the same emulation framework irrespective of their vendor, technology, or type. Therefore, this new functionality is critical not only to emulate DON scenarios with realistic current data models, but also to be compatible with all future data models using the YANG language.

Figure 2b depicts the structure proposed for a generic agent that emulates the role of a software entity in charge of governing optical devices. It is composed of four major components, a NETCONF server, a YANG validator, an XML (eXtensible Markup Language) /JSON (JavaScript Object Notation) config database and the files of the YANG model necessary to emulate an optical device, all coming from open-source initiatives and they are implemented in one single Docker container. Further details follow:

- **NETCONF server:** it provides NETCONF connectivity with the SDN controller with the aim at avoiding dependency of existing agent automation framework, similarly to the Sysrepo in the legacy work. In this work, the NETCONF skeleton is given by the Choppsv1 library [41] that supports both the creation of NETCONF clients and servers, however it does not provide the full implementation of the server. The *pyangbind* module complements the server, because it permits the generation of automatic code from YANG files, the validation of XML calls, essential in NETCONF-based communications and XML/JSON conversion amplifying the potential for connectivity [42].
- **YANG validator:** *pyang* is Python-based library playing the role of a validator, transformer, and generator of YANG code. It can be used for validating YANG modules, YANG modules transformation into different formats, and code generation from modules. This module is key to provide potential scalability and generalization [43].
- **XML/JSON config database:** it stores the configuration files related to the configuration of the agent for the different functional states, such as boot or running.
- **YANG model files:** finally, a set of YANG files that define the behaviour required to emulate an optical device. They are validated by the *pyang* module.

From a global point of view, it is worthwhile to indicate the interaction among the components described above. First, YANG models are provided, e.g., such as Cassini transponder models. Then, agents leverage on the YANG validator based on *pyang* for the creation of the NETCONF skeleton within the NETCONF server. Notably, it is worth recalling that SDN controllers commonly include a series of drivers that implement NETCONF client functionalities for interacting with the agents via NETCONF protocol. This interaction, on the agent side, will leverage on the *pyangbind* module for XML/JSON validation. Then, also in the NETCONF server side, custom instructions are implemented for interacting with the data plane. Finally, we report the interaction between control and data signals flow among the components in the next subsection. For instance, Figure 8 will showcase the interaction between agents (data plane) when connectivity is requested from the SDN controller (control plane).

3. Proof-of-Concept Configuration

In order to test and evaluate the emulation framework of this work, in this section we propose a proof-of-concept that integrates the key elements of the entire architecture. Here we detail the configuration and settings for performing such proof-of-concept.

The proof-of-concept follows the design exposed in Figure 2a, with three main actors: (i) ONOS playing the role of SDN controller that interacts and configure the emulated optical data plane (set of agents in this case), (ii) Kubernetes, in charge of the management of the Docker containers where the optical agents are implemented, and finally (iii) the data plane, represented by the set of agents emulating the software that governs the optical devices. For the sake of simplicity, we only use an agent of Cassini transponder according to the OpenConfig project. All the evaluation system has been deployed and executed in a high-performance laptop (Intel 8th generation i7 with 8 logical cores, 32 GB of 2400 MHz DDR4 RAM and 1 TB of SSD). Further details follow:

- Agent definition and integration: a specific docker container is built to integrate the architecture proposed and described in Section 2.2.3 for an automatic agent deployment. The set of YANG files that replicate the behaviour of a Cassini transponder are extracted from the GitHub repository of the OpenConfig project [44]. Precisely, the files *openconfig-terminal-device.yang*, *openconfig-if-ethernet.yang* and *openconfig-types.yang* are stored in the YANG file catalogue of the agent container.
- Kubernetes configuration: following the structure presented in Figure 3 Kubernetes Cluster is installed in Linux, an Ubuntu distribution, and it is composed by one master and one worker node and one pod in this worker node. Agents' creation and configuration are driven by the *deploy.yaml* file, see Figure 4, where it commands the deployment of the Docker containers. The service *kubectl* is the one to execute the deployment of the agents where results of such deployment are presented in the next section.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cassini-deployment
  labels:
    app: software-agent-cassini
spec:
  replicas: 18
  selector:
    matchLabels:
      app: software-agent-cassini
  template:
    metadata:
      labels:
        app: software-agent-cassini
    spec:
      containers:
        - name: software-agent-cassini
          image: software-agent-cassini
```


Figure 4. Containers deployment configuration script for Kubernetes.

- ONOS: we chose a virtualized distribution of ONOS (version 2.3.0) in a Docker container. The installation using Docker also permits to open the necessary ports for this proof-of-concept in one single command as presented below. Port 8181 is used for external communication with ONOS via GUI, port 5005 is a Java debugger port, port 8101 permits command line interface (CLI) communication via CLI and the port 830 permits NETCONF-based connectivity. Finally, once the container is up and running, pertinent “apps” must be activated to enable the expected functionalities within the SDN controller framework: “odtn-service”, “roadm” and “optical-rest”.

```
docker run -t -d -p 8181:8181 -p 8101:8101 -p 5005:5005 -p 830:830 --name onos onosproject/onos:2.3.0
```

- Topology definition: the topology is defined in a JSON file so that it can be interpreted by ONOS. Seven metrics can be set “Devices”, “links”, “hosts”, “Apps”, “ports”, “regions” and “layouts”. In this work we only use the objects “device” and “link”. Each device object is linked to each container previously deployed by indicating some specific fields, such as IP of the container and port to enable NETCONF connectivity, ONOS driver or map location, as can be seen in the exemplary definition

if Figure 5a. Moreover, the links define the interconnection between two agents by setting their IP and ports, type of connection or bidirectionality, among others as depicted in Figure 5b.



```

"netconf:192.168.31.239:8025": {
  "basic": {
    "name": "Rome",
    "driver": "cassini-openconfig",
    "locType": "geo",
    "latitude": "41.9",
    "longitude": "12.5"
  },
  "netconf": {
    "ip": "192.168.31.239",
    "port": "8025",
    "username": "root",
    "password": "root",
    "idle-timeout": 0
  }
},

"netconf:192.168.31.239:8025/220-netconf:192.168.31.239:8018/21": {
  "basic": {
    "type": "OPTICAL",
    "metric": "1",
    "durable": true,
    "bidirectional": true
  }
},

```

(a)
(b)

Figure 5. Topology details in the JSON configuration file: (a) Device definition; (b) Link definition.

Once the file is ready, it is sent via ONOS NBI API to inform the details of the topology. Topology definition and JSON file creation are performed manually in this proof-of-concept emulating the 18-nodes European Optical Network. Future works will target the automatization of this procedure.

4. Results

This section collects and analyses the results obtained in the deployment of the proof-of-concept focusing on the main points of our proposal for an Advanced Automatic Deployment of Emulated SDN DON. The execution of this experiment is performed following the statements proposed in Section 2 and configured according to Section 3. Given the lack of complex SDN DON emulators in the literature (see Section 1.1) that can not only emulate such scenarios but also interact with it in a realistic way, the objective of this proof of concept is not aimed at evaluating the performance of our framework, but to highlight the feasibility and functionality of our proposal. All the results exposed in this section are obtained after the configuration stage is properly completed.

The first stage of the running workflow in this proof-of-concept is to show the performance of Kubernetes in the agent/container management. Figure 6 illustrates the 18 containers of the Cassini agents properly running according to the configuration file (*deploy.yaml*) defined under the Kubernetes umbrella.

One of the main benefits of managing Docker containers with Kubernetes is that the computational resources of the host are efficiently used to balance the workload of the containers according to their computational needs and performance. Once all the containers are up and running, the next stage is to bind the containers to the proposed optical topology by sending the JSON configuration file via ONOS API. At this point, ONOS recognizes the query and the topology is tied to the containers by assigning the existing agents to the nodes defined in the topology file and links are created to interconnect the nodes. As can be seen in Figure 7, ONOS shows the emulated 18-node European optical network topology in which each node corresponds to a Cassini agent.

software-agent-cassini-2mqn6	1/1	Running	0	4m25s
software-agent-cassini-54wlg	1/1	Running	0	4m25s
software-agent-cassini-6krmp	1/1	Running	0	4m25s
software-agent-cassini-bd2cn	1/1	Running	0	4m25s
software-agent-cassini-bgt52	1/1	Running	0	4m25s
software-agent-cassini-bkx76	1/1	Running	0	4m25s
software-agent-cassini-ffvw6	1/1	Running	0	4m25s
software-agent-cassini-fzxf7	1/1	Running	0	4m25s
software-agent-cassini-hb8qw	1/1	Running	0	4m25s
software-agent-cassini-kzf8c	1/1	Running	0	4m25s
software-agent-cassini-lbx7c	1/1	Running	0	4m25s
software-agent-cassini-lpzwb	1/1	Running	0	4m25s
software-agent-cassini-mlc85	1/1	Running	0	4m25s
software-agent-cassini-n9rt4	1/1	Running	0	4m25s
software-agent-cassini-psfkk	1/1	Running	0	4m25s
software-agent-cassini-q5w5m	1/1	Running	0	4m25s
software-agent-cassini-ts76	1/1	Running	0	4m25s
software-agent-cassini-xtwq9	1/1	Running	0	4m25s

Figure 6. Screenshot of the CLI of Kubernetes showing 18 running agent containers.

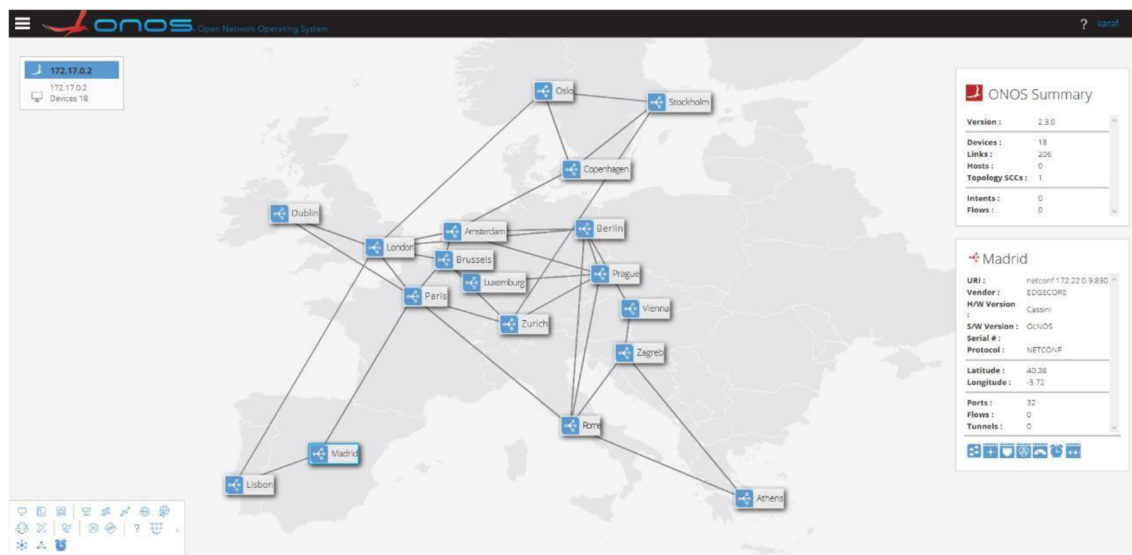


Figure 7. Emulated optical topology deployed on ONOS.

The functionality of the agents deployed in this emulated framework is not only related to the optical topology discovery from the SDN controller side, but also they can perform other common network tasks. Figure 8 illustrates how two containers establish real TCP-based package traffic flows leveraging the deployment of the Emulated SDN DON framework.

64 bytes from 192.168.31.18: seq=9 ttl=64 time=0.666 ms	64 bytes from 192.168.31.125: seq=2 ttl=64 time=0.801 ms
64 bytes from 192.168.31.18: seq=10 ttl=64 time=0.987 ms	64 bytes from 192.168.31.125: seq=3 ttl=64 time=0.570 ms
64 bytes from 192.168.31.18: seq=11 ttl=64 time=0.631 ms	64 bytes from 192.168.31.125: seq=4 ttl=64 time=0.307 ms
64 bytes from 192.168.31.18: seq=12 ttl=64 time=0.573 ms	64 bytes from 192.168.31.125: seq=5 ttl=64 time=0.476 ms
64 bytes from 192.168.31.18: seq=13 ttl=64 time=0.503 ms	64 bytes from 192.168.31.125: seq=6 ttl=64 time=0.431 ms
64 bytes from 192.168.31.18: seq=14 ttl=64 time=0.443 ms	64 bytes from 192.168.31.125: seq=7 ttl=64 time=0.508 ms
64 bytes from 192.168.31.18: seq=15 ttl=64 time=0.500 ms	64 bytes from 192.168.31.125: seq=8 ttl=64 time=0.395 ms
64 bytes from 192.168.31.18: seq=16 ttl=64 time=0.446 ms	64 bytes from 192.168.31.125: seq=9 ttl=64 time=0.622 ms
64 bytes from 192.168.31.18: seq=17 ttl=64 time=0.389 ms	64 bytes from 192.168.31.125: seq=10 ttl=64 time=0.585 ms
64 bytes from 192.168.31.18: seq=18 ttl=64 time=0.266 ms	64 bytes from 192.168.31.125: seq=11 ttl=64 time=0.362 ms
64 bytes from 192.168.31.18: seq=19 ttl=64 time=0.531 ms	64 bytes from 192.168.31.125: seq=12 ttl=64 time=0.533 ms
64 bytes from 192.168.31.18: seq=20 ttl=64 time=0.544 ms	64 bytes from 192.168.31.125: seq=13 ttl=64 time=0.492 ms
64 bytes from 192.168.31.18: seq=21 ttl=64 time=0.398 ms	64 bytes from 192.168.31.125: seq=14 ttl=64 time=0.448 ms
64 bytes from 192.168.31.18: seq=22 ttl=64 time=0.459 ms	64 bytes from 192.168.31.125: seq=15 ttl=64 time=0.644 ms
64 bytes from 192.168.31.18: seq=23 ttl=64 time=0.399 ms	64 bytes from 192.168.31.125: seq=16 ttl=64 time=0.246 ms
64 bytes from 192.168.31.18: seq=24 ttl=64 time=0.269 ms	64 bytes from 192.168.31.125: seq=17 ttl=64 time=0.545 ms
64 bytes from 192.168.31.18: seq=25 ttl=64 time=0.727 ms	64 bytes from 192.168.31.125: seq=18 ttl=64 time=0.445 ms
64 bytes from 192.168.31.18: seq=26 ttl=64 time=0.470 ms	64 bytes from 192.168.31.125: seq=19 ttl=64 time=0.360 ms
64 bytes from 192.168.31.18: seq=27 ttl=64 time=0.553 ms	64 bytes from 192.168.31.125: seq=20 ttl=64 time=0.674 ms
64 bytes from 192.168.31.18: seq=28 ttl=64 time=0.472 ms	64 bytes from 192.168.31.125: seq=21 ttl=64 time=0.443 ms
64 bytes from 192.168.31.18: seq=29 ttl=64 time=0.336 ms	64 bytes from 192.168.31.125: seq=22 ttl=64 time=0.601 ms
64 bytes from 192.168.31.18: seq=30 ttl=64 time=0.501 ms	64 bytes from 192.168.31.125: seq=23 ttl=64 time=0.854 ms
64 bytes from 192.168.31.18: seq=31 ttl=64 time=0.772 ms	64 bytes from 192.168.31.125: seq=24 ttl=64 time=0.257 ms
64 bytes from 192.168.31.18: seq=32 ttl=64 time=0.698 ms	64 bytes from 192.168.31.125: seq=25 ttl=64 time=1.956 ms

Figure 8. Real traffic flows between two ports of two different nodes.

In a similar way than a real optical device, our emulation framework permits to retrieve the status of the emulated devices in real time regarding their current configuration or other functional metrics. Figure 9 showcases an exemplary query requested by the SDN controller via NETCONF protocol about the status of the configuration of an optical agent (namely, emulated Cassini transponder) in a specific instant of time. In particular, the agent is queried about its current target-output, current-output, and current-input optical powers.

```
onos@root > power-config get netconf:192.168.31.239:8022/22

The target-output-power value in port 202 on device netconf:192.168.31.239:8022
is -5.100000.

The current-output-power value in port 202 on device netconf:192.168.31.239:8022
is 0.120000.

The current-input-power value in port 202 on device netconf:192.168.31.239:8022
is 0.120000.
```

Figure 9. Snapshot of NETCONF-based queries about the status of the configuration in an optical agent commanded from ONOS.

The functionality of the deployed agents is not only limited to configuration and status queries, the SDN controller can also change physical parameters of emulated devices in the same way as for a real optical device. Figure 10 is a proof of the performance of this procedure by using the CLI of ONOS to change the output power in an optical port for a specific Cassini transponder.

```
onos@root > power-config edit-config

netconf:192.168.31.239:8022/22 2 Set 2.00000 power on port
```

Figure 10. Snapshot of NETCONF-based queries to change the output power of the laser in an emulated optical agent commanded from ONOS.

5. Conclusions

In this work, we presented an emulation framework that leverages on a combination of SDN and container management with Kubernetes for the emulation of both control- and data-planes of disaggregated optical networks. Our legacy proposal was reviewed including the architectural details and the classical optical network agent structure that leveraged on the Netopeer2-Sysrepo framework. Then, we presented the Containerized Framework for emulating SDN-DONs paying special attention to the coexistence of the server–client architecture based on Kubernetes master nodes and the client–server structure that used NETCONF in the SDN controller–optical agent interaction. We further detailed the software agent structure which includes NETCONF server, YANG validator, XML/JSON config database and YANG models, whose combination offers versatility for accommodating the wide variety of optical components and systems. The implementation of the agents in this emulating system was deployed in Docker containers allowing advanced and efficient management by Kubernetes. We described all the configuration and settings for performing the proof-of-concept that consisted in an 18-node European topology with Cassini transponders. We reported the successful status request and configuration settings of optical-layer parameters such as transmitted and received optical power. Given the successful interaction between the control and data plane positions, our proposal demonstrates an adequate approach for managing an operationally complex carrier-grade transport infrastructure with SDN-based disaggregated optical systems.

Author Contributions: Conceptualization, F.-J.M.-M., M.G. and I.I.-C.; methodology, F.-J.M.-M. and M.G.; software, I.I.-C. and S.Z.; validation, F.-J.M.-M.; formal analysis, F.-J.M.-M. and P.P.-M.; investigation, F.-J.M.-M. and S.Z.; resources, P.P.-M.; writing—original draft preparation, F.-J.M.-M.; writing—review and editing, M.G., I.I.-C., S.Z., and P.P.-M.; supervision, P.P.-M.; funding acquisition, P.P.-M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded Spanish Government: ONOFRE-2 project under Grant TEC2017-84423-C3-2-P (MINECO/AEI/FEDER, UE) and the Go2Edge project under Grant RED2018-102585-T; and by the European Commission: METRO-HAUL project (G.A. 761727).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cisco. Cisco Annual Internet Report (2018–2023) White Paper. 2019. Available online: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf> (accessed on 30 December 2020).
2. Sandvine. The Global Internet Phenomena Report COVID-19 Spotlight. 2020. Available online: <https://www.sandvine.com/phenomena> (accessed on 27 January 2021).
3. Napoli, A.; Bohn, M.; Rafique, D.; Stavdas, A.; Sambo, N.; Poti, L.; Noelle, M.; Fischer, J.K.; Riccardi, E.; Pagano, A.; et al. Next generation elastic optical networks: The vision of the European research project IDEALIST. *IEEE Comm. Mag.* **2015**, *53*, 152–162. [CrossRef]
4. Feldmann, A.; Gasser, O.; Lichtblau, F.; Pujol, E.; Poesse, I.; Dietzel, C.; Wagner, D.; Wichtlhuber, M.; Tapiador, J.; Vallina-Rodriguez, N. The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic. In Proceedings of the ACM Internet Measurement Conference, New York, NY, USA, 27–29 October 2020.
5. Zoom. Available online: <https://zoom.us> (accessed on 27 January 2021).
6. Microsoft Teams. Available online: <https://www.microsoft.com/en-us/microsoft-365/microsoft-teams> (accessed on 27 January 2021).
7. Cisco Webex. Available online: <https://www.webex.com> (accessed on 27 January 2021).
8. 5G Infrastructure Public Private Partnership (5G PPP). Key Performance Indicators (KPIs). Available online: <https://5g-ppp.eu/kpis> (accessed on 27 January 2021).
9. Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. White Paper. 2012. Available online: <https://opennetworking.org/category/sdn-resources/whitepapers> (accessed on 27 January 2021).
10. Kreutz, D.; Ramos, F.M.V.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
11. Channegowda, M.; Nejabati, R.; Simeonidou, D. Software-defined optical networks technology and infrastructure: Enabling software-defined optical network operations. *IEEE/OSA J. Opt. Commun. Netw.* **2013**, *5*, A274–A282. [CrossRef]
12. Thyagaturu, A.S.; Mercian, A.; McGarry, M.P.; Reisslein, M.; Kellerer, W. Software Defined Optical Networks (SDONs): A Comprehensive Survey. *IEEE Commun. Surv. and Tut.* **2016**, *18*, 2738–2786. [CrossRef]
13. Giorgetti, A.; Casellas, R.; Morro, R.; Campanella, A.; Castoldi, P. ONOS-controlled Disaggregated Optical Networks. In Proceedings of the Optical Fiber Communication Conference (OFC), San Diego, CA, USA, 3–7 March 2019.
14. OpenConfig. Available online: <http://www.openconfig.net/> (accessed on 27 January 2021).
15. OpenROADM. Available online: <http://www.openroadm.org/> (accessed on 27 January 2021).
16. The Telecom Infra Project. Available online: <https://telecominfraproject.com/> (accessed on 27 January 2021).
17. Bjorklund, M. YANG—A Data Modeling Language for the Network Configuration Protocol (NETCONF); IETF RFC 6020; ISSN: 2070-1721; IETF: Fremont, CA, USA, 2010.
18. Enns, R.; Bjorklund, M.; Schoenwaelder, J.; Bierman, A. Network Configuration Protocol (NETCONF); IETF RFC 6241; ISSN: 2070-1721; IETF: Fremont, CA, USA, 2011.
19. Garrich, M.; Bravalheri, A. Overview of South-Bound Interfaces for Software-Defined Optical Networks. In Proceedings of the International Conference on Transparent Optical Networks (ICTON), Bucharest, Romania, 1–5 July 2018.
20. Casellas, R.; Martinez, R.; Vilalta, R.; Munoz, R. Metro-Haul: SDN Control and Orchestration of Disaggregated Optical Networks with Model-Driven Development. In Proceedings of the Optical Fiber Communication Conference (OFC), San Diego, CA, USA, 1–5 March 2018.
21. Giorgetti, A.; Sgambelluri, A.; Casellas, R.; Morro, R.; Campanella, A.; Castoldi, P. Control of open and disaggregated transport networks using the Open Network Operating System (ONOS). *IEEE J. Lightw. Techn.* **2020**, *12*, A171–A181. [CrossRef]
22. Casellas, R.; Martinez, R.; Vilalta, R.; Munoz, R. Abstraction and control of multi-domain disaggregated OpenROADM optical networks. In Proceedings of the European Conference on Optical Communication (ECOC), Dublin, Ireland, 22–26 September 2019.

23. Gifre, L.; Izquierdo-Zaragoza, J.; Ruiz, M.; Velasco, L. Autonomic Disaggregated Multilayer Networking. *IEEE/OSA J. Opt. Commun. Netw.* **2018**, *10*, 482–492. [[CrossRef](#)]
24. Nadal, L.; Casellas, R.; Fabrega, J.M.; Munoz, R.; Moreolo, M.S.; Rodriguez, L.; Vilalta, R.; Vilchez, F.J.; Martinez, R. Multi-vendor sliceable transceivers in partial disaggregated metro networks. In Proceedings of the European Conference on Optical Communication (ECOC), Dublin, Ireland, 22–26 September 2019.
25. Nadal, L.; Fabrega, J.M.; Moreolo, M.S.; Casellas, R.; Munoz, R.; Rodriguez, L.; Vilalta, R.; Vilchez, F.J.; Martinez, R. SDN-Enabled Sliceable Transceivers in Disaggregated Optical Networks. *IEEE J. Lightw. Technol.* **2019**, *37*, 6054–6062. [[CrossRef](#)]
26. Kundrat, J.; Havlivs, O.; Jedlinsky, J.; Vojtvech, J. Opening up ROADMs: Let Us Build a Disaggregated Open Optical Line System. *IEEE/OSA J. Opt. Commun. Netw.* **2019**, *37*, 4041–4051. [[CrossRef](#)]
27. Xie, C.; Wang, L.; Dou, L.; Xia, M.; Chen, S.; Zhang, H.; Sun, Z.; Cheng, J. Open and disaggregated optical transport networks for data center interconnects. *IEEE/OSA J. Opt. Commun. Netw.* **2020**, *12*, C12–C22. [[CrossRef](#)]
28. Garrich, M.; San-Nicolas-Martinez, C.; Moreno-Muro, F.; Lopez-de-Lerma, A.M.; de Dios, O.G.; Lopez, V.; Giorgetti, A.; Sgambelluri, A.; Tancevski, L.; Verchere, D.; et al. Gap Analysis on Open Models for Partially-Disaggregated SDN Optical Transport Environments. In Proceedings of the Optical Fiber Communication Conference (OFC), San Diego, CA, USA, 3–7 March 2019.
29. GNPY. Available online: <https://gnpy.readthedocs.io/en/master/> (accessed on 10 February 2021).
30. Poggiolini, P. The GN model of non-linear propagation in uncompensated coherent optical systems. *IEEE/OSA J. Lightwave Technol.* **2012**, *30*, 3857–3879. [[CrossRef](#)]
31. Filer, M.; Cantono, M.; Ferrari, A.; Grammel, G.; Galimberti, G.; Curri, V. Multi-Vendor Experimental Validation of an Open Source QoT Estimator for Optical Networks. *IEEE/OSA J. Lightwave Technol.* **2018**, *36*, 3073–3082. [[CrossRef](#)]
32. Kundrat, J.; Campanella, A.; le Rouzic, E.; Ferrari, A.; Havlivs, O.; Havzlinzky, M.; Grammel, G.; Galimberti, G.; Curri, V. Physical-Layer Awareness: GNPY and ONOS for End-to-End Circuits in Disaggregated Networks. In Proceedings of the Optical Fiber Communication Conference (OFC), San Diego, CA, USA, 8–12 March 2020.
33. Casellas, R.; Giorgetti, A.; Morro, R.; Martinez, R.; Vilalta, R.; Munoz, R. Virtualization of disaggregated optical networks with open data models in support of network slicing. *IEEE/OSA J. Opt. Commun. Netw.* **2020**, *12*, A144–A154. [[CrossRef](#)]
34. Lantz, B.; Diaz-Montiel, A.A.; Yu, J.; Rios, C.; Ruffini, M.; Kilper, D. Demonstration of Software-Defined Packet-Optical Network Emulation with Mininet-Optical and ONOS. In Proceedings of the Optical Fiber Communications Conference (OFC), San Diego, CA, USA, 8–12 March 2020.
35. Mininet. Available online: <http://mininet.org> (accessed on 27 January 2021).
36. Iglesias-Castreno, I.; Alabarce, M.G.; Hernandez-Bastida, M.; Marino, P.P. Towards an Open-Source Framework for Jointly Emulating Control and Data Planes of Disaggregated Optical Networks. In Proceedings of the International Conference on Transparent Optical Networks (ICTON), Bari, Italy, 19–23 July 2020.
37. Cassini by TIP. Available online: https://cdn.brandfolder.io/D8DI15S7/as/q3wkdg-476u4o-8wg0g7/Cassini_at_a_Glance_-_Telecom_Infra_Project.pdf (accessed on 10 February 2021).
38. Kubernetes. Available online: <https://kubernetes.io> (accessed on 27 January 2021).
39. ONOS—Open Network Operating System. Available online: <https://opennetworking.org/onos/> (accessed on 10 February 2021).
40. Foukas, X.; Patounas, G.; Elmokashfi, A.; Marina, M.K. Network Slicing in 5G: Survey and Challenges. *IEEE Comm. Mag.* **2017**, *55*, 94–100. [[CrossRef](#)]
41. Chopps v1 Netcon Client/Server Library. Available online: <https://github.com/choppsv1/netconf> (accessed on 27 January 2021).
42. PyangBind. Available online: <https://github.com/robshakir/pyangbind> (accessed on 27 January 2021).
43. Pyang—A YANG Validator. Available online: <https://github.com/mbj4668/pyang> (accessed on 27 January 2021).
44. OpenConfig Project Repository. Available online: <http://openconfig.net/yang/terminal-device> (accessed on 27 January 2021).