A User-friendly Interface for Virtualization and Management of Optical Access Networks

Carine Mineto, Luis Riveros, Julia Maciel, Tiago Sutili, and Rafael C. Figueiredo
CPQD
Campinas, Brazil
{cmineto, lriveros, jsousa, tsutili, rafaelcf}@cpqd.com.br

Fernando Farias
RNP
Rio de Janeiro, Brazil
fernando.farias@rnp.br

***Abstract*—This paper presents an orchestration interface that integrates Virtualization and Control agents to abstract and virtualize open line terminals (OLTs) while managing the allocation of users and services in the open network units (ONUs) within passive optical networks (PONs). The proposed interface is compared with the command-line operations provided by the developer community, highlighting improvements in network operation and management through a more intuitive and user-friendly approach. The results demonstrate a reduction in execution time and enhanced reliability in applied configurations.**

***Keywords*—Software-defined networking, disaggregation, network orchestration.**

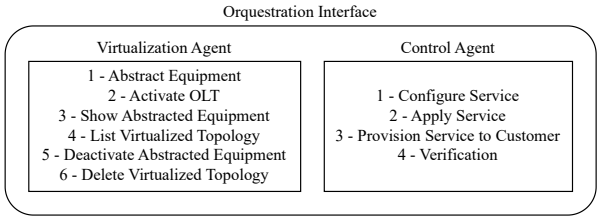

Fig. 1. High-level interface framework developed to create and control a virtualized optical access network.

I. Introduction

SOFTWARE-DEFINED networks (SDN) have replaced physical hardware with software capabilities, increasing flexibility and innovation in network infrastructure [1]. This paradigm shift facilitates the development of streamlined network models, significantly reducing both capital (CAPEX) and operational (OPEX) expenditures, while simultaneously intensifying vendor competition [2]. Notably, the emergence of Software-Defined Passive Optical Networks (SD-PON) within access network architectures exemplifies this trend, embracing an open model that promotes interoperability, cost-effectiveness, and rapid innovation, ultimately benefiting both operators and end-users [3–5]. However, the multiplicity of components in access networks introduces challenges to the complete disaggregation of these networks. Moreover, many manufacturers still do not adopt open interfaces and standards in the architecture in their equipment architectures, hindering interoperability and causing communication failures, degraded service quality, and additional costs for users. In this sense, standardizing open interfaces and protocols enables interoperability and ease of equipment integration from different manufacturers, leading to optimized performance in disaggregated networks.

In this context, an ecosystem of open-source project development is formed by groups of foundations, companies, network operators, and technology centers. The Open Networking Foundation (ONF) [6] is the leading organization developing advancements for access networks, being responsible for the Virtual OLT Hardware Abstraction (VOLTHA) project [7]. This project consists of a solution based on the abstraction of the decentralized physical layer of the PON network, with an architecture in microservices orchestrated in Kubernetes and an ONOS Controller that presents southbound interfaces for communication with equipment, and northbound interfaces for communication with the controller and the management system. In this scenario, this paper introduces a high-level interface that simplifies interaction with the northbound interface. Unlike the current ONF-provided solution, which relies on command-line operations, the proposed system enhances usability. The proposal is to allow the user to operate the abstraction layer, interacting dynamically with an operator-friendly interface, without demanding specific knowledge of the VOLTHA project. Moreover, the interfaces presented here can be easily installed and operated externally to the equipment that will be virtualized, and can be implemented in future SD-PON applications.

This work was partially supported by the Brazilian Ministry of Science and Technology (MCTI), FUNTTEL/Finep (grant number 0943/23), and CNPq (grants number 305104/2021-7 and 305777/2022-0). This work was developed within the OpenRAN@Brasil Program, with execution and infrastructure resources provided by RNP and CPQD.

II. Proposed High-level Interface

We present a simplified orchestration interface to facilitate use by integrating control and orchestration, which are usually managed separately by the control agent and virtualization agent. In the proposed approach, these agents are combined into a single interface to enhance the operator's usability of the tool. Specifically, the developed interface allows to: expand and adjust the resources necessary for SDN orchestration, making management more adaptable to operators' evolving needs; disaggregate the architecture (*i.e.*, this interface can be implemented in environments external to the traditional SDN infrastructure of the passive optical network), providing greater versatility in network management; and simplify the creation of applications and services by adopting a high-level programming language (Python) that can send REST and

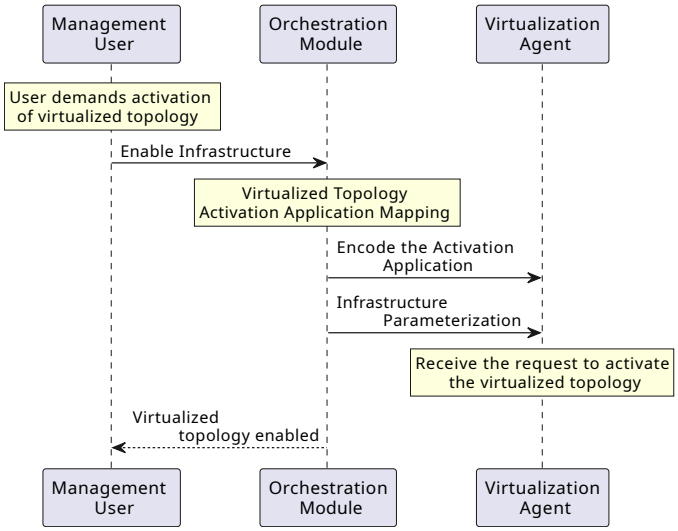

Fig. 2. Flow to be followed by the high-level interface here developed to activate the virtualized abstracted equipment.

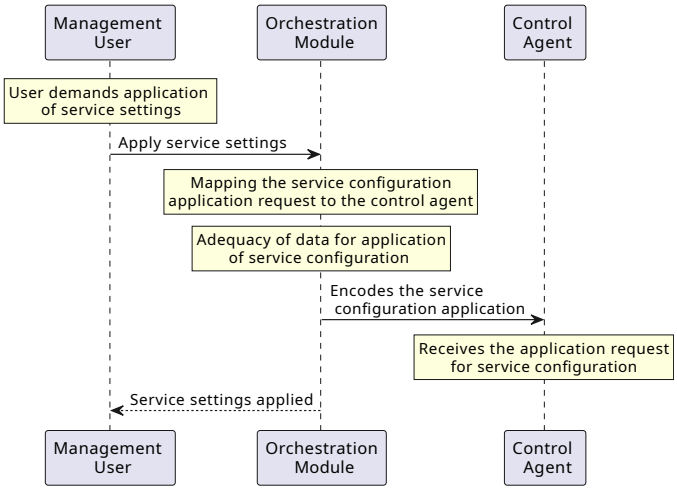

Fig. 3. Flow to be followed by the high-level interface developed to apply user services.

GRPC requests to the SDN infrastructure, depending on the acting agent.

A schematic of this implemented orchestration interface is presented in Fig. 1, showing: the Virtualization Agent, which contains the functions of abstracting the equipment to be virtualized, activating it, viewing the abstracted equipment, listing the virtualized topology, deactivating optical elements and also deleting the created topology; and the Control Agent where the functions related to assigning a network service to the client are implemented, which can configure, apply, provision and verify. Thus, the process of virtualizing an optical line terminal (OLT) of an PON transmission system to provision a client to a service involves four distinct entities, named: 1 - Management User, which in the description was denoted as User; 2 - Orchestration Module, which was referenced as Modulator; 3 - Virtualization Agent; and 4 - Control Agent. Therefore, the process begins when the User requests a virtualized topology and communicates with the Modulator. The system then identifies the required functions, variables, and protocols to create the topology. This data is sent to the Virtualization Agent, which generates the virtual optical element. Finally, the Modulator informs the User that the process is complete.

Moreover, it is then possible to virtually activate the optical element, as presented in the schematic in Fig. 2. Therefore, the User's request is required, allowing the Modulator to start the communication and gather the necessary information. Next, the Modulator transmits an activation request encoded with the collected data to the Virtualization Agent, which activates the virtual optical element. Finally, the Modulator confirms the successful activation to the User. After topology creation, the User could also request to view it. In this case, the system maps the necessary information and sends a request to the Virtualization Agent, which receives and adapts the returned data, presenting it to the User.

The Management User, in addition to requesting actions from the Virtualization Agent, which is responsible for managing a virtualized data network, can also, through the Orchestration Module, request control of equipment. This request will be executed by a Control Agent, whose purpose is to manage services and clients. As can be seen in Fig. 1, these operations can include the configuration, application, and provision of services to a user, and there is also the possibility of viewing the services of each client. For comparison purposes, the application and provisioning actions are explained in detail below. For those actions, after the service configurations have been created, the User can request that they be applied, following the flow in Fig. 3. To achieve this, the Modulator organizes the configuration information so that it can be understood and processed by the Control Agent. The Modulator then sends a request to the Control Agent, asking it to apply the configurations. Next, the Control Agent receives the request and executes the target equipment or system configurations. Finally, the Modulator informs the User that the configurations have been successfully applied. Similarly, provisioning a client involves collecting information, adapting it into the appropriate format, creating

```
voltha@master-node:~$ voltctl device create -t openolt -H bbsim0.voltha.svc:50060
214de907-b725-4ef2-a76e-93f61bb511e6
voltha@master-node:~$
```

(a) Screenshot of creating an OLT using the command line as made available by the development community.

```
        Orchestration Method
Choose which Agent will be called:
1 - Virtualization              2 - Control: 1

Actions will be performed related to the Virtualization Agent.

Which action will be performed by the Virtualization Agent?
1 - Abstract Equipment;                 2 - Activate OLT;
3 - Show Abstracted Equipment;          4 - List Virtualized Topology;
5 - Deactivate Abstracted Equipment;    6 - Delete Virtualized Topology
7 - Check Ports of Abstracted Equipment: 2

Enabling Equipment

Enabling OLT...

OLT enabled
```

(b) Screenshot of activating an OLT using the interface here proposed.

Fig. 4. Comparison of the operation of a virtualized network when performing the actions of (a) creating an OLT via CLI and (b) activating it via a high-level interface.

```
        Orchestration Method
Choose which Agent will be called:
1 - Virtualization              2 - Control: 2

Actions will be performed related to the  Control Agent.

Which action will be performed by the Control Agent?
1 - Configure Service;                  2 - Apply Service
3 - Provision Service to Customer;      4 - Verification: 2

Applying the configured services to the ONUs

Applying the configurations to the ONUs...

Configurations applied!
```

(a) Screenshot of a service application to the ONU using the high-level interface here proposed.

```
voltha@master-node:~$ ssh -p 8101 karaf@localhost
Password authentication
Password:
Welcome to Open Network Operating System (ONOS)!
    ____  _  ______  ____
   / __ \/ |/ / __ \/ __/
  / /_/ /    / /_/ /\ \
  \____/_/|_/\____/___/

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

karaf@root > aaa-users
of:00000a0a0a0a0a0a/256: AUTHORIZED_STATE, last-changed=3h1m ago,
mac=2E:0A:00:01:00:00, subid=BBSM000a0001-1, username=user
karaf@root > volt-add-subscriber-access of:00000a0a0a0a0a0a 256
karaf@root > volt-programmed-subscribers
location=of:00000a0a0a0a0a0a/16777216 tagInformation=UniTagInformation{
uniTagMatch=Any, ponCTag=None, ponSTag=null, usPonCTagPriority=-1,
usPonSTagPriority=-1, dsPonCTagPriority=-1, dsPonSTagPriority=-1,
technologyProfileId=-1, enableMacLearning=false, upstreamBandwidthProfile='null',
downstreamBandwidthProfile='null', upstreamOltBandwidthProfile='null',
downstreamOltBandwidthProfile='null',serviceName='nni', configuredMacAddress='null',
isDhcpRequired=false, isIgmpRequired=false, isPppoeRequired=false}
location=of:00000a0a0a0a0a0a/256 tagInformation=UniTagInformation{uniTagMatch=0,
ponCTag=1, ponSTag=10, usPonCTagPriority=-1, usPonSTagPriority=-1, dsPonCTagPriority=-1,
dsPonSTagPriority=-1, technologyProfileId=64, enableMacLearning=false,
upstreamBandwidthProfile='User_Bandwidth1', downstreamBandwidthProfile='User_Bandwidth1',
upstreamOltBandwidthProfile='User_Bandwidth1', downstreamOltBandwidthProfile='User_Bandwidth1',
serviceName='hsi', configuredMacAddress='', isDhcpRequired=true, isIgmpRequired=false,
isPppoeRequired=false}
```

(b) Screenshot of provisioning a service for the ONU using the CLI, as provided by the development community.

Fig. 5. Comparison of the operation of a virtualized network when performing the actions of (a) applying a service to the ONU via the high-level interface and (b) provisioning it via the CLI.

a provisioning application, and executing that application by the Control Agent, all based on the User's demand.

III. High-level Interface Operation

To demonstrate the interface's ease of use, we compare it to a command-line interface (CLI) by creating a virtualized PON network. Thus, in Fig. 4 it is possible to see the difference in operation, whereas in Fig. 4(a) it is possible to see the method of creating an OLT through the CLI, requiring several parameters to be defined by the user, such as the type of equipment being created, the host in which it will be allocated, to make communications through this IP and port. Alternatively, in Fig. 4(b) it is possible to verify the process of activating this OLT through the interface here proposed, where the only action required from the operator is to select a number that represents the action that will be executed, according to the steps presented in Section II. It is important to highlight that these parameters are also necessary for the abstraction of the equipment, but they are collected and applied automatically within the developed code, eliminating the need for the user to enter them manually.

```
       Orchestration Method
Choose which Agent will be called:
1 - Virtualization              2 - Control: 2

Actions will be performed related to the Control Agent.

Which action will be performed by the Control Agent?
1 - Configure Service;                  2 - Apply Service
3 - Provision Service to the Customer;  4 - Verification: 1

Service Settings for ONUs
Service Settings for OLT BBSIM_OLT_10

Active ONUs available for configuration: [ONU_1 ONU_2 ONU_3 ONU_4]

How many ONUs do you want to configure? 1
Choose one among the active ONUs: ONU_1
The requested ONU is serial BBSM000a0002-1

How many services do you want to configure for this ONU? 1
Enter the service name: HSIA
Enter the Client Tag: 1
Enter the Service Tag: 1

Choose between bandwidth profiles: Default

Client Upstream Bandwidth Profile: Default
Client Downstream Bandwidth Profile: Default

DHCP Required?
Choose between options: true; false
true

Service configuration data created
```

(a) Screenshot of a service configuration to the ONU using the high-level interface here proposed.

```
{
"bandwidthprofile": {
        "entries": [
                {
                        "air": 100000,
                        "cbs": 20000,
                        "cir": 0,
                        "ebs": 30000,
                        "eir": 200000,
                        "id": "Default"
                }
        ],
        "integration": {
                "cache": {
                        "enabled": true,
                        "maxsize": 40,
                        "ttl": "PT1m"
                }
        }
},
"sadis": {
        "entries": [
                {
                "hardwareIdentifier": "0a:0a:0a:0a:0a:0a\"",
                "id": "BBSIM_OLT_10",
                "ipAddress": "0.0.0.0",
                "nasId": "BBSIM_OLT_10",
                "uplinkPort": 0
                },
                {
                "circuitId": "BBSM000a0002-1",
                "id": "BBSM000a0002-1",
                "nasPortId": "BBSM000a0002-1",
                "remoteId": "BBSM000a0002-1",
                "uniTagList": [
                        {
                        "downstreamBandwidthProfile": "Default",
                        "isDhcpRequired": true,
                        "ponCTag": 1,
                        "ponSTag": 1,
                        "serviceName": "HSIA",
                        "technologyProfileId": 64,
                        "upstreamBandwidthProfile": "Default"
                        }
                ]
                }
        ],
        "integration": {
                "cache": {
                        "enabled": false,
                        "maxsize": 50,
                        "ttl": "PT0m"
                }
        }
}
}
```

(b) Screenshot of json file created to configure a service for the ONU and apply it in the CLI, with the minimum parameters to be accepted by the controller.

Fig. 6. Comparison of the configuration of a service to the ONU via the high-level interface (a) and (b) setting up from a json file to apply it via the CLI.

In a similar comparison, Fig. 5(a) presents two actions performed with the developed Control Agent, applying the configured service to the ONU through the developed interface, in which the User selects the option to apply the service and, intrinsically, the developed code follows the steps presented in Fig. 3 for its implementation. On the other hand, Fig. 5(b) shows the number of steps to be taken to provision this applied service through the CLI method: initially, checking if the ONU is authenticated; verifying which port was allocated to it; provisioning the service; and, finally, checking if the provisioning worked. Alternatively, if the developed interface were used, the User would simply select a number from the available menu, and the necessary actions would be performed, with these previous steps running in the background using API/REST commands. These comparisons demonstrate the ease of use allowed by this interface: by intrinsically configuring multiple parameters, it reduces the operator interaction with several parameters of the SDN framework. In contrast, using the community-provided conventional structure could lead to errors or operational difficulties due to its reliance on prior knowledge of parameters not commonly used by network operators.

This comparison is evinced in the codes shown in Fig. 6, in which the service is configured via the proposed interface (Fig. 6a) and via the .json file (Fig. 6b). It is worth noting that the proposed interface output will generate the same .json file using the provided data in a user-friendly manner. However, if it is not used, the user needs to be familiar with the entire structure and formatting of the required json file. In a simpler manner, in the proposed interface, the operator only needs to input how many ONUs he wants to configure, knowing as shown that there are 4 available; and for the selected ONU, of which the corresponding serial number is already shown, how many services he will add, inserting parameters regarding the client and service tags, the bandwidth and whether DHCP is necessary. Comparatively, in the json file, the user needs to know the json architecture, starting the file with opening curly braces, and then starting with the bandwidth profile data, parameters that configure the service T-CONT, the profile entries, then the Subscriber Access Device Information Service (SADIS) data, with several parameters such as hardware identifier, IP address, circuit ID, remote ID, among others. In other words, to directly use the community architecture, it is necessary to have prior knowledge of the network architecture, as well as equipment data, which would be suppressed when using the proposed interface, which already collects most of such data in the background, improving its ease of use and operator friendliness.

IV. Conclusions and Future Works

To overcome the challenges associated with the complexity and rigidity of passive optical networks management, a simplified and disaggregated orchestration interface was introduced, aiming to make the orchestration of an open-source SDN infrastructure more intuitive and adaptable to the ever-changing needs of network operators. The results demonstrate that our proposed Python interface offers a significantly more intuitive way to configure PON elements, an improvement over directly using the CLI of ONOS and VOLTHA's Kubernetes solution. In addition to its application in optical access networks as here presented, the proposed interface can be adapted to: mobile networks and 5G networks, where flexibility in resource management is crucial; transport and distribution networks, optimizing large-scale content delivery; and cloud computing environments and data centers, where efficient management of network resources is essential. Furthermore, the interface is still under development, and, as its application is explored in different scenarios, it is possible to make adjustments so that it meets the most varied architectures of passive optical networks.

References

[1] D. Kreutz *et al.*, "Software-defined networking: A comprehensive survey - proceedings of the ieee," *Proceedings of the IEEE*, vol. 103, pp. 14 – 76, jun 2015. [Online]. Available: http://arxiv.org/abs/1406.0440

[2] A. Hakiri *et al.*, "Leveraging SDN for the 5G Networks: Trends, Prospects, and Challenges," *Software Defined Mobile Networks (SDMN) Beyond LTE Network Architecture*, pp. 61–80, 2015.

[3] L. Martins *et al.*, "Teste para validação do Virtual OLT Hardware Abstraction (VOLTHA) para gerência de redes SD-PON," *WPEIF - XIV Workshop de Pesquisa Experimental da Internet do Futuro*, pp. 1–6, 2023.

[4] F. Slyne *et al.*, "Demonstration of Cooperative Transport Interface using open-source 5G OpenRAN and virtualised PON network," in *2024 Optical Fiber Communications Conference and Exhibition (OFC)*, 2024, pp. 1–3.

[5] Fierce Network, "Creating New Telco Opportunities in the Cloud with Open RAN." [Online]. Available: https://www.fierce-network.com/resource/creating-new-telco-opportunities-cloud-open-ran

[6] Open Networking Foundation, "ONOS - Open Network Operating System." [Online]. Available: https://opennetworking.org/onos/

[7] ——, "SEBA/VOLTHA." [Online]. Available: https://opennetworking.org/voltha/