JOURNAL OF
Optical Communications
and Networking

Software defined networking agent demonstration to enable configuration and management of XGS-PON architectures

David de Pintos,[1,2] Noemí Merayo,[1,*] Carlos Sangrador,[1] Juan Carlos Aguado,[1] Ignacio de Miguel,[1] and Ramón J. Durán Barroso[1]

[1]Universidad de Valladolid, Valladolid, Spain
[2]david.pintos@uva.es
*noemer@tel.uva.es

Received 3 May 2023; revised 8 July 2023; accepted 13 July 2023; published 16 August 2023

This paper describes the design and implementation of an OpenFlow software defined network (SDN) agent that manages and configures 10-gigabit-capable symmetric passive optical network (XGS-PON) architectures. Acting as an OpenFlow switch, the SDN agent communicates with an SDN controller using OpenFlow, while holding direct communication with the optical line terminal (OLT) through the chipset manufacturer-specific application programming interface, eliminating the need for emulating SDN layers in hardware devices. The proposal was evaluated through experiments conducted on a White Box XGS-PON OLT using the Open Network Operating System. The results demonstrate that the proposal facilitates a real-time SDN configuration of various Internet services, successfully fulfilling different quality of service requirements. Due to its ease of deployment, low complexity, smooth learning curve, scalability, and flexibility in integrating services, the proposal has significant potential. As a result, it offers a rapid SDN solution for configuring and testing new functionalities with minimal programming changes required in specific layers of the developed SDN agent.

https://doi.org/10.1364/JOCN.494694

1. INTRODUCTION

As data traffic continues to grow exponentially, due to bandwidth-hungry applications, the rise of video streaming, or the rollout of 5G, the need for higher capacity, faster, and more reliable networks is becoming more pressing. In addition, the COVID-19 crisis has led to more data traffic and more bandwidth demand, which have accelerated the deployment of optical fiber connections in many countries. Indeed, forecasts foresee around 309 million fiber to the home/building (FTTH/B) connected households in 2027 in the EU region [1]. In fact, the global FTTH/B market is expected to reach USD 29.7 billion by 2026 [2]. For instance, China is expected to reach an estimated market size of USD 8 billion by 2026, with a steady annual growth rate of 15%. Japan and Canada, other leading countries, are expected to grow by 10.5% and 11.8%, respectively, by 2026 [2]. In this global scenario, passive optical networks (PONs) are the preferred technology for FTTH deployments, as they offer a flexible and scalable solution and provide gigabit speeds over long distances without the need for active components. In fact, the global GPON (PONs based on the gigabit standard) market is estimated to be USD 13.52 billion by 2027, growing at a compound annual growth rate (CAGR) of 12.1% [3].

A PON is a fiber-optic network that employs a multipoint topology and optical splitters to transmit data from a single point of transmission, the optical line terminal (OLT), to multiple user end points, optical network terminals/optical network units (ONTs/ONUs). The concept of passive refers to the absence of a power supply to the fiber and optical splitters. However, while GPON is the basis of current FTTH deployments, the declining costs of 10G PON equipment and the prospect of faster time-to-value for their networks is leading many service providers to consider deploying symmetrical 10G networks (ITU-T G.9807.1), also called 10-gigabit-capable symmetric PON (XGS-PON) [4]. Thus, XGS-PON is expected to reach 55% of the GPON market by 2026, as it already accounted for 15% in 2021. In fact, migration from GPON to XGS-PON is simple, brings great flexibility as both technologies can coexist, and can be done progressively as bandwidth demands increase, since part of the deployed network equipment and the optical distribution network itself are reused [5].

The simultaneous integration of multiple PON technologies as well as the increase of services with high quality of service (QoS) requirements make these networks increasingly complex to manage. This is coupled with the fact that PONs are often

managed by proprietary and inflexible network management systems. The development of software defined network (SDN) solutions for PON infrastructures effectively addresses these challenges, enabling more efficient operation and management of networks, thus creating more self-aware as well as autonomous and secure networks [6]. SDN solutions allow network operators to centrally control network management operations and to apply automation to complex and time-consuming tasks, avoiding manual interventions as much as possible. In addition, SDN technology can bring other benefits to PONs such as enabling fast and efficient configurations of services and residential networks as well as allowing the coexistence of PON devices from different manufacturers and PON technologies. SDN uses software-based controllers to direct traffic and communicate with the underlying hardware infrastructure, separating the control plane from the data plane to provide intelligent, centralized network management, so that operators can manage and monitor the network independently of the underlying technology [7]. Therefore, in an SDN architecture, the software application implementing the SDN controller takes control of the network, centralizing management and intelligence, and communicates with the hardware using different protocols such as OpenFlow (the most widely deployed SDN-based protocol) [8], NETCONF [9], or RESTCONF [10].

Due to these advantages, there are multiple proposals exploring the integration of SDN into PON architectures, as opposed to using traditional proprietary and inflexible management systems. In this regard, many proposals focus on implementing a software abstraction layer in PON devices (OLTs, ONUs) to make them SDN controllable and able to emulate the behavior of SDN switches [11–15]. Under this approach, it is necessary to program a virtual layer into an external device (one for each PON device), such as a Raspberry Pi or minicomputer, that communicates with all OLTs and ONUs, since they cannot be directly integrated into the devices themselves. This is not a native implementation, leading to increased complexity and cost. In contrast, other proposals such as Virtual OLT Hardware Abstraction (VOLTHA) [16], an open source project of the Central Office Re-architected as a Datacenter (CORD) initiative, reduce the PON to a programmable switch controlled by an SDN controller capable of translating SDN commands sent by the controller into the proprietary management system of the OLTs/ONUs without using external hardware devices or additional software layers while allowing for efficient network management. This open source project is integrated in the SDN Enabled Broadband Access (SEBA) architecture [17], an Open Networking Foundation (ONF) project that builds an open source solution for virtualized broadband access.

In previous research, we developed a solution based on this last approach, where an SDN agent based on OpenFlow was programmed and experimentally tested over a commercial GPON [18]. However, that SDN proposal interacted indirectly with the OLT, since the agent translated the OpenFlow messages from the platform to the vendor's proprietary management system control commands. On the contrary, in this paper, we present an SDN-OpenFlow agent for XGS-PON architectures that directly interacts with the OLT and ONU chipsets through the specific application programming interface (API) of the OLT chipset. This approach is highly scalable, as it is divided into simple and highly differentiated blocks, allowing new SDN functionalities to be easily added in one block without affecting the rest, unlike other approaches such as VOLTHA, where the implementation of new functionalities affects the modification of several layers and blocks. It is also very easy and fast to deploy, as the management system is fragmented into a small number of containers, with well differentiated tasks, through a simple dockerisation system. Furthermore, the developed agent is very flexible, because in addition to being managed through tools such as the Subscriber/Access Device Information Service (SADIS) or Open Network Operating System (ONOS), it can also be managed through an easy and intuitive menu-driven interface. Therefore, the learning curve for managing the proposed OpenFlow-based SDN agent is shallow and short, and configuring it and programming new functionalities are easier compared to other proposals. In addition, its installation is lightweight and simple, as it consists of very few Docker containers to deploy, as opposed to other approaches that require a larger number of containers (resulting in increased complexity), such as VOLTHA. The simplicity of the installation also translates into a reduction of the complexity of the configuration. By reducing the number of components, the interdependencies are simplified, which is one of the major drawbacks of systems with a large number of components. Furthermore, the SDN approach offers great potential as it has been developed for OLT PON medium access control (MAC) chipsets supporting multiple PON technologies (GPON, XGPON, XGS-PON, NGPON2, EPON, and 10G-EPON) through a common API.

In summary, this work presents a solution for the management of XGS-PON OLTs through SDN. The PON is controlled by means of OpenFlow, so that an SDN controller such as ONOS [19] can configure the traffic rules. This solution has been experimentally tested on an XGS-PON architecture, and the results are also presented in this paper. In addition, the open source code of this SDN solution is available on GitHub to be directly applicable for network designers.

This paper is structured as follows. Section 2 describes the state of the art of SDN in XGS-PONs. Section 3 presents the design of the OpenFlow-based SDN agent for XGS-PON infrastructures, and Section 4 describes the communication among the agent, SDN controller, and XGS-PON devices. Section 5 explains the menu-driven user interface and Section 6 the experimental scenario. Then, Section 7 shows the validation and results of the proposal over an experimental XGS-PON testbed. Finally, Section 8 summarizes the main conclusions.

2. STATE OF THE ART

This section describes various approaches for integrating SDN into PONs to achieve different objectives, including QoS management, user-side home network management, enforcement of protection policies, and control of PON devices.

A significant number of studies propose that bandwidth management and service configuration policies should be performed by an SDN controller external to the network, so that PONs do not use proprietary and inflexible management systems. In this way, Khalili *et al.* [20] propose an SDN architecture (SIEPON) in which they move non-time critical tasks, namely, the registration of ONTs and some dynamic bandwidth allocation (DBA) policies, to an external SDN controller. Also, the SDN architecture proposed in [21] integrates a reprogrammable DBA module inside the SDN controller that manages several DBA algorithms and activates them according to the traffic demands of the ONTs. In [22], an SDN architecture for the remote management of real-time services according to QoS in a PON is presented. Liang *et al.* propose in [23] a resource allocation algorithm using a long short-term memory (LSTM) neural network to predict traffic in an SDN-based PON. Regarding time wavelength division multiplexing (TWDM)-PON, in [24], an SDN architecture is proposed where an SDN controller dynamically allocates bandwidth and wavelengths to users (ONUs) to meet QoS requirements. The SDN architecture proposed in [25] similarly allows an SDN controller to manage bandwidth allocation policies to guarantee QoS requirements in TWDM-PONs. Along this line, the authors of [26] proposed to integrate an SDN solution in a hybrid xhaul-TWDM-PON architecture for interactive video applications. Simulation results show that the proposal improves average packet delay, packet loss, and bandwidth efficiency. Contrary to most research on bandwidth and service configuration strategies using simulation models, the authors of [15] propose the implementation of an SDN management layer on a GPON testbed that applies service configuration strategies related to bandwidth levels, according to the real-time QoS requirements of network subscribers. Centofanti *et al.* [27] have experimentally tested different SDN strategies to support low latency services, using a NETCONF-based SDN controller that performs slice management over a commercial XGS-PON infrastructure.

Other SDN proposals focus on managing residential networks in PON architectures, due to the growing number of connected devices in users' homes and applications with high QoS requirements [14,28]. Regarding protection and energy saving in PONs, an SDN-based architecture was implemented in [29] to allow fast feeder fiber protection over PONs. Other solutions [30] propose to save energy by moving the power control of OLTs and ONTs to an external SDN controller. In addition, SDN is integrated into protection tasks in long-reach PONs (100 km), as the long range and high division ratio mean that any network outage can interrupt the services of thousands of users [31]. Moreover, Wang *et al.* [32] propose and experimentally tested an SDN-based PON radio access network (RAN) protection scheme. The proposal significantly improves on other existing protection mechanisms and can be used as a protection mechanism for non-time critical services. On the other hand, SDN solutions are also applied in virtual PONs (VPONs). Quian *et al.* propose in [30] a hybrid software defined PON architecture where multiple VPONs are managed by a DBA algorithm. Also, the flex PON architecture [33] implements SDN to flexibly schedule services across multiple VPONs.

Other proposals are based on mapping of OpenFlow messages into PON commands, thus proposing extensions to the current OpenFlow protocols. Parol and Pawlowski propose in [34] to integrate GPON-related functions to develop an extension to the OpenFlow protocol (called OpenFlowPLUS, since the current OpenFlow specification does not natively support GPON. Amokrane *et al.* propose in [35] extensions to OpenFlow that consist of mapping flows (defined in OpenFlow) to GPON Encapsulation Method (GEM) ports (defined in PON), as well as inserting and extracting virtual local area network (VLAN) tags from traffic.

However, a great deal of research work is focused on making PON devices, OLTs, and ONUs, controllable via SDN technology [11–15]. In fact, many proposals integrate an abstraction layer to make PON devices controllable by SDN, as there are currently no SDN-based OLTs or ONTs on the market. In [11], the authors have programmed an SDN agent that on one hand interacts with the OLT through PON commands and on the other hand with an SDN controller through OpenFlow. In [12], the authors implemented a GPON-based SDN-enabled virtual switch so the GPON is transformed into a single OpenFlow switch, hiding the GPON operational details from users. In the same way, the authors of [13] developed an OpenFlow-based architecture for gigabit Ethernet passive optical networks (GEPONs). Merayo *et al.* [14,15] have designed and experimentally tested an SDN solution on GPONs, integrating a software layer on top of OLTs and ONTs (programmed on external devices such as Raspberry/Banana Pi or minicomputers) to emulate the behavior of OpenFlow virtual switches (OVSs) [36]. These approaches imply having a virtual layer programmed in PON devices, which increases complexity in the network and moves away from being an integrated, native solution. Consequently, other approaches choose to avoid this complexity by developing SDN agents capable of transforming SDN commands into the native configuration of PONs. Along this line is the VOLTHA project [16], which abstracts the PON to a programmable switch managed by an SDN controller and interacts with PON devices using proprietary and manufacturer-specific languages. It does this by making use of the OpenOLT agent and adapters implemented in OLTs and ONUs. However, the agent makes use of Broadcom's Broadband Adaptation Layer (BAL) software to interface with OLT chipsets, and some of this software is not open source. In fact, VOLTHA is integrated in the SEBA project [17], an open source development and integration project promoted by the ONF. It builds on and integrates a great number of other ONF projects, such as VOLTHA, ONOS, Trellis, SD-BNG, XOS, P4, OpenOLT, BBSim, Stratum, and Tassen. SEBA brings the advantages of virtualization and cloudification to PONs since it uses a common API to abstract various OLT and ONUs and to control them using an SDN controller. In this sense, Suzuki *et al.* proposed in [37] that SEBA can support IEEE PON, as the current abstraction unit in SEBA supports only ITU-T PON. Indeed, the same authors also proposed to implement edge computing on a 10G EPON based on SEBA [38]. These experimental architectures are very promising in PONs, as the rapid deployment and coexistence of different PON technologies (GPON, EPON, 10G PON)

have led to the emergence of many PON device manufacturers, leading to some incompatibility issues when managing them simultaneously.

Although all PON devices comply with the same standards, there is no such thing as a fully universal PON device. This is because each vendor uses different proprietary software to access the OLT, leading to significant differences in how access methods are implemented in the internal chipsets (OLTs, ONUs). To address this issue, SDN protocols such as OpenFlow and NetConfig can be used to manage PON equipment from different vendors, technologies (such as GPON and 10GPON), and standards (IEEE PON, ITU-T PON). Consequently, our proposal is close to these approaches, as we have developed an OpenFlow-based SDN agent that interacts directly with the APIs of the chipsets of OLTs and ONUs. The proposal provides significant potential due to its simplicity in deployment, scalability, and adaptability in integrating new services and functionalities. This implies that the learning curve for managing our OpenFlow-based SDN agent is much shorter and consists of fewer containers (less complexity) compared to other existing approaches (such as VOLTHA). Consequently, it provides a fast SDN solution for configuring and testing new functionalities with minimal programming changes required in specific SDN agent layers. Furthermore, it has undergone experimental testing to demonstrate real-time deployment of different types of Internet services with diverse QoS requirements, thus proving its efficacy to service providers and network operators in XGS-PON architectures. Finally, it is important to note that it could be adapted to other PON technologies.

3. GLOBAL DESIGN OF AN OPENFLOW-BASED SDN AGENT FOR XGS-PON INFRASTRUCTURES

The proposed architecture permits operators and Internet service providers (ISPs) to configure 10G PONs (XGS-PONs), such as ONUs registration/deletion, configuration/deletion of services (Internet, video, voice), network parameters, using an SDN controller (ONOS [19]), and a novel OpenFlow-based SDN agent that interacts with these networks. The approach consists of an OpenFlow-based SDN agent with two differentiated interfaces, as can be observed in Fig. 1. On one hand, the ONOS adapter interface connects with the ONOS controller and captures messages coming from the ONOS controller, giving them a response as if the OpenFlow-SDN agent were an OVS. On the other hand, the OLT adapter interface connects to XGS-PON OLTs to manage and configure them in their language. Finally, there is an intermediate translation layer that connects both interfaces and is responsible for translating ONOS instructions into XSGPON configurations. In addition, a menu-driven user interface, which allows instances to the ONOS API to be created through a simple interface to configure PON parameters, has been implemented, although our proposal also allows instances to be created in ONOS directly. The code and the instructions to install and deploy the SDN agent (using Docker containers) are available on GitHub for researchers to replicate the SDN agent and its deployment in XGS-PON architectures [39]. In fact, the entire management

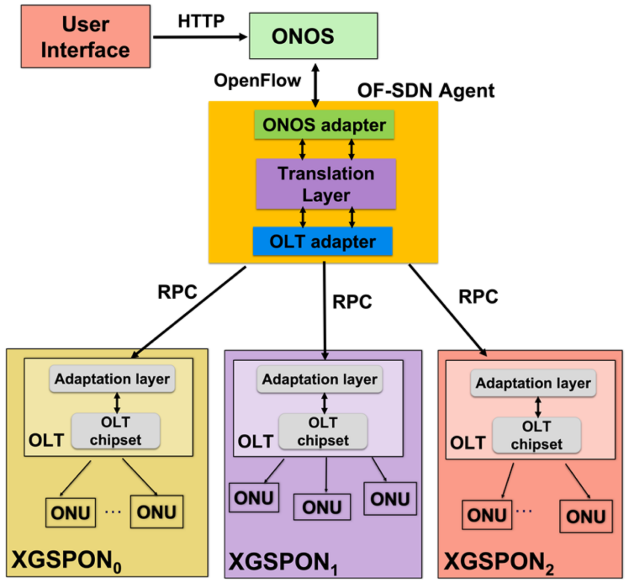


Fig. 1. OpenFlow agent to manage XGS-PONs from different vendors.

system consists of three containers, one for ONOS, one for the menu interface, and one containing the SDN agent.

Specifically, the OpenFlow-based SDN agent captures messages sent by ONOS, containing OpenFlow commands, to configure the OLT and ONUs connected to the XGS-PON as well as the end-user services/profiles (each one individually related to an ONU) through the OLT, since the OLT is the management element of an access network. In this way, the OpenFlow agent extracts the configuration parameters sent by ONOS in OpenFlow messages and with these parameters interacts directly with the specific API of the OLT chipset vendor. To do this, our proposal uses remote procedure call (RPC) connections [40] so that the OpenFlow agent interacts with the API of the OLT chipset. Then the OLT configures the devices and services (e.g., Internet, voice, video) contracted by users. Therefore, the agent transforms the commands sent by the SDN controller via OpenFlow into commands interpretable by the API of the OLT chipset. The reason for selecting RPC is its utilization of a client-server model that offers remarkable adaptability to the system. This enables the management of OLTs from different vendors utilizing the same RPC configuration, as the identical functions can be employed to control different devices [38].

On the other hand, as each OLT may have a different OLT chipset, the only change when managing XGS-PON equipment from different manufacturers would be the internal API of the OLT (OLT chipset). It will therefore be necessary to program an adaptation layer inside the OLT that receives the RPC instructions and consequently manages the network through the particular management API of the OLT chipset. An important feature of the proposed OpenFlow-based SDN agent is that it can be deployed at software level (outside the OLT) to control XGS-PON OLTs from different manufacturers, as it is an external software layer implemented as a bridge between an SDN controller and an XGS-PON infrastructure. In this way, the same SDN controller (ONOS in this case) could control

PON equipment from different manufacturers using the same SDN agent. Figure 1 shows a flexible network scenario with the proposed OpenFlow-based SDN agent, where ONOS can be able to interact with several XGS-PONs simultaneously through the same SDN agent. By utilizing only the SDN agent code, the RPC, and the ONOS-interacting graphical user interface (GUI), our proposal enables the management of any XGS-PON through a unified code. The adaptation layer (inside the OLT in Fig. 1) responsible for direct interaction with the API of the OLT chipset vendor is the only component that needs modification, as different OLTs can have distinct chipsets.

In addition, our proposal is scalable and flexible, as it is divided into simple and highly differentiated blocks, ONO adapter, translation layer, and OLT adapter, which makes it easy to add new SDN functionalities to one of the blocks without affecting the rest. New functionalities or types of services can be easily integrated by simply defining that type of flow or schema in the ONOS adapter and generating the service in the translation layer. In this way, the new service would already be configured without the need to modify the OLT adapter layer.

4. COMMUNICATION OF THE OPENFLOW AGENT WITH THE SDN CONTROLLER AND XGS-PON

This section explains how the OpenFlow agent interacts, on one hand, with the ONOS controller and, on the other hand, with the API of XGS-PON OLTs. The OpenFlow agent was implemented using the python-openflow library, developed by Kytos [41], which is compatible with the openflow standard version 1.3/1.0.

A. Communication between the SDN Controller and the OpenFlow Agent: ONOS Adapter

The first step is to establish an initial connection between ONOS and the OpenFlow agent. This connection is established through a socket that connects the agent to the ONOS port associated with the OpenFlow protocol (6633). The proposed OpenFlow-based SDN agent emulates the behavior of an OpenFlow switch, creating real OpenFlow messages (using the Kytos python-openflow library) that will be exchanged with ONOS according to requests from the SDN controller. When an OpenFlow switch connects to an SDN controller, sets of OpenFlow messages are exchanged. The specific OpenFlow messages exchanged between the OpenFlow agent and ONOS are shown in Fig. 2, using Wireshark [42] to capture this sequence of messages. In Fig. 2, the green frames represent OpenFlow messages used for service configuration, while the red frames belong to messages responsible for initiating communication and exchanging information between ONOS and the agent. The following sections explain the OpenFlow messages exchanged between ONOS and the SDN agent during communication.

1. Connection Establishment

When an OpenFlow connection is first established, each side of the connection shall immediately send an OFPT_HELLO message with the highest OpenFlow protocol version supported by the sender. On receipt of this message, the recipient must select the version of the OpenFlow protocol to be used. If the negotiated version is compatible with the recipient, the connection is established. As shown in Fig. 2, once the OpenFlow agent and ONOS have exchanged OFPT_HELLO messages and successfully negotiated a common version, the connection is established and they start exchanging OpenFlow messages.

2. Initial Communication

Once the connection has been established, ONOS periodically requests information from the OpenFlow agent through different types of messages. The first thing ONOS does is to send an OFPT_FEATURES_REQUEST message (Fig. 2), and the OpenFlow agent responds to this request with an OFPT_FEATURES_REPLY message, which includes information that allows the controller to identify the switch and its basic capabilities.

Next, ONOS requests a description of all the ports registered in the OpenFlow agent. These ports are the network-to-network interface (NNI) port of the OLT and the user network interface (UNI) ports of all connected ONTs. For this purpose, ONOS sends an OFPT_MULTIPART_REQUEST message of type OFMP_PORT_DESC, as shown in Fig. 2, and the OpenFlow agent responds with an equivalent OFPT_MULTIPART_REPLY, which contains the information of all registered ports, but since the OLT has been recently added, there is not any ONU discovered yet. For that reason, the SDN agent initially includes only the OLT NNI port information on the OFMP_PORT_DESC message type. Once the NNI port is registered, the network is ready to register the ONTs, sending an OFPT_PORT_STATUS message for each UNI port of each discovered ONU, as shown in the Wireshark screenshot in Fig. 2, to register these ports in ONOS.

After these initial messages, ONOS and the OpenFlow SDN agent exchange further OpenFlow messages. One of the requests of ONOS is the characteristics of the supported

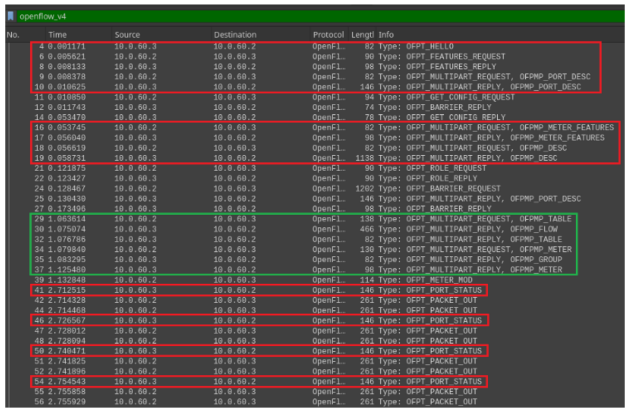


Fig. 2. OpenFlow messages between the OpenFlow-SDN agent and ONOS.

meters. Meters enable OpenFlow to implement simple QoS operations, such as rate limiting. In fact, a meter is an element in the OpenFlow standard that can measure and control the rate of packets in flows. A flow is an element used to match and process packets. The flows and meters are used in our proposal to configure different types of services (e.g., Internet, video, voice). Thus, ONOS requests this information by means of an OFPT_MULTIPART_REQUEST message of type OFMP_METER_FEATURES and the OpenFlow agent responds with an OFPT_MULTIPART_REPLY message of type OFMP_METER_FEATURES.

3. Network Management and Configuration with ONOS

Once the connection has been established, ONOS periodically requests information from the OpenFlow agent through different types of messages. The first thing ONOS does is to send an OFPT_FEATURES_REQUEST message (Fig. 2), and the OpenFlow agent responds to this request with an OFPT_FEATURES_REPLY message, which includes information that allows the controller to identify the switch and its basic capabilities.

Once the initial communication is established, the controller periodically requests flow, table, and meter statistics from the OpenFlow agent via OFPT_MULTIPART_REQUEST messages. Since a flow is used to match and process packets, it contains a set of match fields for matching packets, a priority for matching precedence, a set of counters to track packets, and a set of instructions to apply [43]. In addition, meters measure and control the rate of packets in flows. The meter triggers a meter band if the packet rate or byte rate passing through the meter exceeds a predefined threshold. If the meter band drops the packet, it is called a rate limiter [43]. As mentioned above, our proposal uses flows and meters to configure services. In this version, the OpenFlow-SDN agent is able to configure Internet services, in particular single and double services. In this way, within XGS-PONs, traffic must always be simply tagged using VLANs. This tagging is used to differentiate services (data, voice, video, etc.) that share the same transmission medium. The VLAN tag is an identifier that must be known by both ends of the communication. The tag used to identify each of these services in single tagging is called C-tag (customer tag). When traffic comes from the transport network, it may also have a double VLAN tag (QinQ), which can be used to differentiate several operators arriving at the same PON (in our case, XGS-PON). This second tag is called S-tag (service tag). Therefore, the S-tag must be assigned a value when double tagging is required and will be applicable only in the transport network, while the C-tag is for single tagging in the transport network and the XGS-PON and must always be specified. Thus, to configure a single tag or double tag Internet service, both will have a single tag configured in the access network (C-tag), but in the double tag scenario, a VLAN tag (S-tag) will be added in the OLT interface (adding the S-tag in the upstream direction and removing it in the downstream direction), as can be observed in Fig. 3.

Thus, the most important messages exchanged between ONOS and the OpenFlow agent for configuring, updating, or deleting services are as follows:

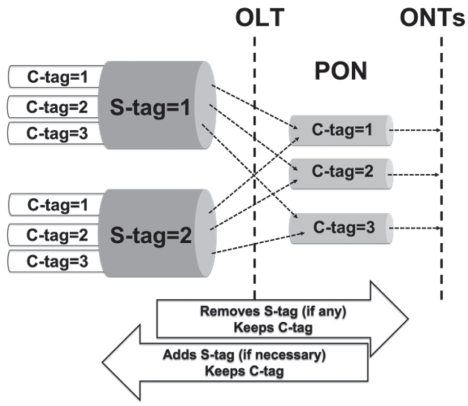


Fig. 3. Explanation of S-tag and C-tag in PONs in XGS-PON services.

- OFPT_FLOW_MOD: ONOS sends this message to add or delete a flow. The flows are used to configure a service on a specific user (UNI port of an ONT). In the OLT, services are filtered by VLANs, so the criteria and instructions of the flows are based on VLANs too. Additionally, flows are created both upstream and downstream, which requires a set of matching flows to create a service. In fact, for single tag (C-tag) services, two flows should be created for each service, as the XGS-PON operates in two channels: one for managing the downstream QoS requirements and another for upstream QoS requirements. In the case of double tag services (S-tag), four flows are needed because of the two VLANs, so a pair of flows should be created for each VLAN (upstream, downstream).
- OFPT_METER_MOD: Meters are used to associate a maximum bandwidth with a service. In this approach, QoS requirements are linked to maximum bandwidth levels. This maximum bandwidth is composed of guaranteed bandwidth plus excess bandwidth, which is offered when the network has enough bandwidth available. Thus, the maximum bandwidth of a specific service is controlled by meters, so it is necessary to assign a meter to each flow, one to control the maximum downstream bandwidth and one to control the maximum upstream bandwidth. Then, the band rate defines the maximum bandwidth associated with the service at both channels (upstream, downstream). Therefore, a meter measures the packet rate of a flow and allows its rate to be controlled, so that when this rate is higher than a maximum value (maximum bandwidth) set in the meter, packets are dropped.

Finally, the designed OpenFlow agent collects statistics of the flows and counters of the configured services, so that it responds to the controller with OFPT_MULTIPART_REPLY messages of type OFMP_FLOW and OFMP_METER, as shown in Fig. 3.

B. Translation of OpenFlow Commands to XGS-PON Commands: Translation Layer

In the previous section, we detailed the communication of the SDN agent with the ONOS controller (ONOS adapter), and at this point, the OpenFlow agent must translate the commands received from ONOS into commands that are

understood by the OLT, so that the OLT adapter sends this information to the OLT by means of RPC instructions.

Thus, the first step is to check the received ONOS flows, which are assigned to an OpenFlow table [43]. First, when a new flow is received, the agent checks the matching fields and looks for the identifier of an incoming port, which will either be a UNI port of an ONT (traffic comes from the ONT, upstream) or the NNI port of the OLT (traffic comes from outside the XGS-PON, downstream). The SDN agent will then search for matching flows to create and configure a service. The agent supports up to two tables, so any flow should be on table "0" or "1." Thus, two tables have been defined in this proposal because the mapping fields are based on VLANs, one for C-tagged services (table 0) and two for S-tagged services (table 0, table 1). In addition, PONs use VLANs to differentiate services (data, voice, video), which can be single or double tagged. Accordingly, the SDN agent arranges these services (C-tag, S-tag) in the tables according to the following rules:

- *Single tagged services*: The C-tag filter is set to table 0 and only one flow is set to each traffic direction (one for downstream and one for upstream). In that case, a valid flow must have an OpenFlow port in the "OUTPUT" instruction, namely, a UNI port (of an ONU) in the downstream direction or the NNI port of the OLT in the upstream direction.
- *Double tagged services*: These services have two VLAN IDs (S-tag and C-tag), so each traffic channel has two flows configured, one for each filtering VLAN, requiring a total of four flows. Because of these two filtering VLANs, two OpenFlow tables are needed. With this in mind, a valid double tagged service is formed by matching flows from different tables, starting from table 0 and progressing to table 1 where an "OUTPUT" instruction must be set as before: a UNI port of an ONU in the downstream or the NNI port of the OLT in the upstream.

For each new service, the SDN agent collects the flow parameters and stores them in a list of services, waiting to be sent to the OLT through the OLT adapter. Each entry in the list of services consists of the following fields:

- *Port*: It identifies the UNI port of the ONU on which the service shall be deployed. This field is obtained from the "OUTPUT" instruction of the flows for the OpenFlow downstream channel or from the "IN_PORT" instruction in the case of flows for the upstream channel.
- *NNI*: It is the ID of the OLT NNI port.
- *VLAN*: It contains the VLAN identifiers C-tag and S-tag. The S-tag field would be "null" in the case of single-tag services. These parameters are taken from the flows, extracting the C-tag from the "VLAN_VID" match filter and in the case of S-tag from the "VLAN POP" (remove VLAN S-tag) or "VLAN_PUSH" (add VLAN S-tag), depending on the downstream or upstream channel, respectively.
- *Meters ID*: It contains two lists of meter IDs, one for upstream direction and another for downstream.

Additionally, the translation layer contains information about the PON and its devices (OLT, ONUs), so that it relates the instances and data coming from ONOS to the specific parameters of the PON. Thus, the types of data stored in the translation layer are defined as follows:

- *ONU data*: ONU data, such as ONU identifier in the PON, are stored for the purpose of activation and management of the ONU.
- *Service parameters (Internet):* Parameters of a service (such as bandwidth, Alloc-ID, GEM ports, etc.) are stored in the translation layer to be managed through the OLT adapter layer (install or remove services). However, other QoS parameters such as additional bandwidth policies, priorities, and weights in both traffic schedulers and queues are set by default in this layer, so all services are configured with the bandwidth policies set to "Best Effort," and the priority and weights are set to zero. Furthermore, the proposed OpenFlow SDN agent has been currently designed to assign a single GEM port for each service.
- *Default OMCI configuration*: It defines and stores the relationships between any type of service (Internet, multicast, voice) and the ONT management and control interface (OMCI) configuration to be installed at any ONU.

C. Communication between the OpenFlow-SDN Agent and the XGS-PON: OLT Adapter

At this point, the OpenFlow agent is able to receive the ONOS flows and meters and store the services and parameters to be configured in the XGS-PON in lists. So, the next step is to connect the OpenFlow agent to the XGS-PON and configure the services (voice, data, video) and the ONTs in their language (commands interpretable by chipsets). The communication with the XGS-PON is done via an RPC connection, which includes an RPC server on the OLT and an RPC client on the SDN agent inside the OLT adapter. Since all OLTs will use the same RPC functions to be managed, the RPC server configuration will be unique and shared between OLTs. Thus, each OLT has an internally installed RPC server with the same functions defined in the RPC client, which will be used by the SDN agent to configure the PON. To achieve maximum compatibility between different OLTs, the same configuration of the VOLTHA RPC server will be used, defining the following functions:

- *GetDeviceInfo*: to get OLT information (model, vendor ID, number of PON ports, MAC address);
- *HeartbeatCheck*: to check the status of the OLT ("OK" if enable);
- *EnableIndication*: to create a channel to send indications and alarms detected in the OLT (the SDN agent shall listen on that channel);
- *ActivateOnu*: to activate an ONU by setting it in Operational State, Physical Layer Operation Administration and Maintenance (PLOAM) activation operation, after assigning an internal identifier (ONU-ID) [16];
- *CreateTrafficSchedulers*: to configure the bandwidth of the service, traffic schedulers, both downstream and upstream along with the Alloc-ID;
- *RemoveTrafficSchedulers*: to remove existing traffic schedulers;
- *CreateTrafficQueues*: to create priority traffic queues in both downstream and upstream channels;

- *RemoveTrafficQueues*: to remove the traffic queues;
- *FlowAdd*: to join the service configuration in a flow, one for each channel (upstream, donwstream);
- *FlowRemove*: to remove the service flow;
- *OmciMsgOut*: to send OMCI messages to a specific ONU for configuration, as the OLT API supports only sending OMCI messages to ONUs;
- *Reboot*: to reboot the OLT.

These functions allow the network operator to configure the XGS-PON, acting directly on the OLT driver API. Therefore, in the initialization process, the agent connects to the OLT, collects the OLT device information with GetDeviceInfo, and calls EnableIndication to create the communication channel and listen for indications from the OLT.

The following sections will explain some issues related to the activation of an ONU and how to set up a service, in particular Internet services. Although the OLT adapter layer uses RPC functions to send these configurations to the OLT, it is important to note that the translation layer coordinates the whole process, leveraging its knowledge of the state of each device (PON, ONU) and their associated parameters. This approach allows for modifications and improvements of services and functionalities mainly at the translation layer.

1. ONU Activation Process

When the SDN agent receives, in the OLT adapter, the indication that an ONU has been discovered (ONU Discovery Indication), it transfers it to the translation layer. This layer will then activate the ONU by sending the corresponding parameters and calling the RPC ActivateOnu function through the OLT adapter layer. Once the ONU is activated, the SDN agent must create the following entities according to the ITU-T G.988 standard [44] by means of OMCI messages using the OmciMsgOut function:

- *MAC Bridge Service Profile*: It implements a bridge that may have any number of ports associated.
- *GAL (GEM adaptation layer) Ethernet Profile*: It defines the maximum size of the payload generated in the entity.
- *MAC Bridge Port Configuration Data*: It creates an instance of this entity for each UNI port, modeling a port on the MAC bridge.
- *Extended VLAN Tagging Operation Configuration Data*: It classifies and executes operations on the VLAN tagging of MAC Bridge Port traffic, whereby an instance of this entity is associated with each MAC Bridge Port Configuration Data.

Thus, Fig. 4 shows the log in the SDN agent of this OMCI configuration and the described entities, with one instance of these entities for each registered UNI port.

2. Service Configuration

Once the ONU is activated and initialized, new services (Internet in our case) can be created, and the SDN agent must configure it in both the OLT and the ONU. As previously explained, the ONOS adapter receives the OpenFlow messages from ONOS, and the translation layer will store this information. Then, when the translation layer knows the type of service to be created, it organizes the necessary data and sends the appropriate requests to the OLT adapter layer, which sends the configuration to the OLT. In this version, the OpenFlow-SDN agent has been designed to configure and remove Internet services.

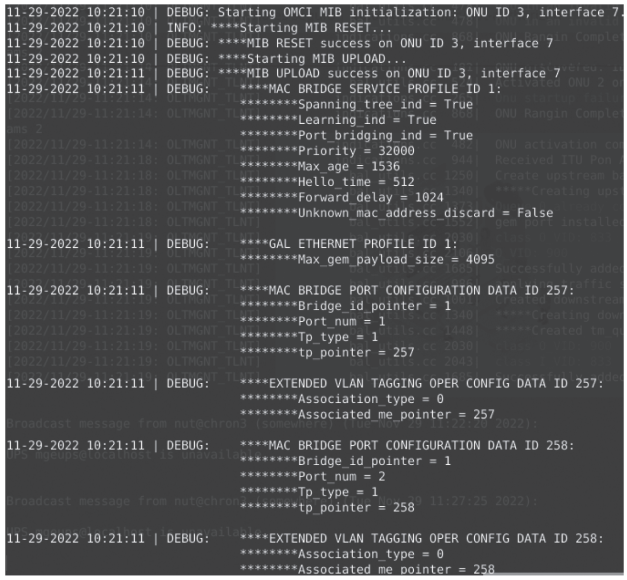


Fig. 4. ONU OMCI initialization process.

On the OLT side, Figs. 5 and 6 describe the OLT configuration of an Internet service inside the OLT. Figure 5 shows the downstream configuration (from the OLT to the PON), while Fig. 6 shows the upstream configuration (from the OLT out of the PON). As can be observed, the SDN agent calls the function CreateTrafficSchedulers to configure service parameters, such as the downstream and upstream bandwidth profiles, that is, Downstream TrafficScheduler in Fig. 5 and Upstream TrafficScheduler in Fig. 6 together with the Alloc-ID. These bandwidth profiles (in upstream and downstream) contain the committed information rate (CIR) parameter, which corresponds to the guaranteed bandwidth, and the peak information rate (PIR) parameter, which corresponds to the maximum bandwidth. Downstream TrafficScheduler is associated with the Queue 0, so that these queues correspond with the T-CONTs of the PON standards. In the next step, the agent calls to the FlowAdd method by means of the OLT adapter, to join all service parameters (ONU ID, GEM port, Ingress/Egress interface) in a flow to be sent to the OLT. Moreover, as PONs filter different types of traffic by VLANs,

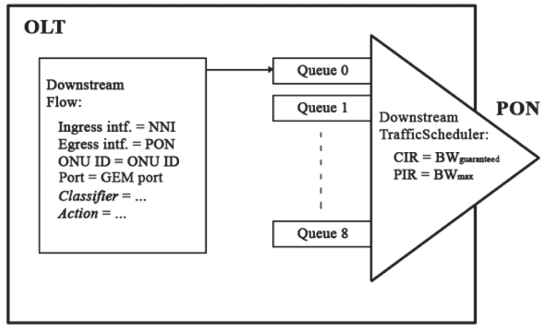


Fig. 5. Diagram of downstream service configuration.

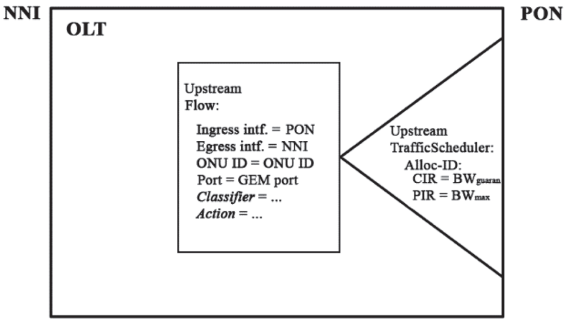


Fig. 6. Diagram of upstream service configuration.

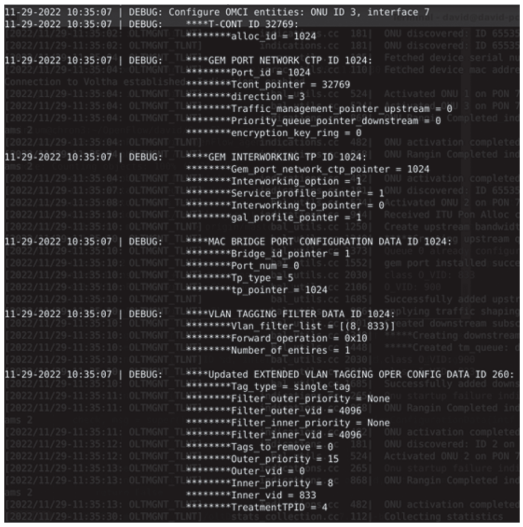


Fig. 7. ONU OMCI service configuration.

the flow must contain the Classifier and Action structures (Figs. 5 and 6) related to filter traffic based on VLANs and how to manage these VLANs (push and pop actions).

On the other hand, as far as the ONU configuration is concerned, the corresponding OMCI entities shall be implemented in accordance with ITU-T G.988 [44]. Figure 7 shows a screenshot of the status of the log related to the OMCI entities deployed when configuring an Internet service. First, the service Alloc-ID is set to the first available T-CONT. Then, the agent creates the following entities:

- *GEM Port Network Connection Termination Point (CTP)*: to represent the GEM port termination of the service at the ONU, in this case, GEM port 1024 (Fig. 7);
- *GEM Interworking TP*: to link the GEM port network CTP and the MAC Bridge configuration;
- *MAC Bridge Port Configuration Data*: entity described on the ONU activation section;
- *VLAN Tagging Filter Data*: to perform traffic filtering tasks by the VLAN identifier.

Finally, since the Extended VLAN Tagging Operation Configuration Data entity is created at ONU initialization, in the service configuration, the agent should update only this entity.

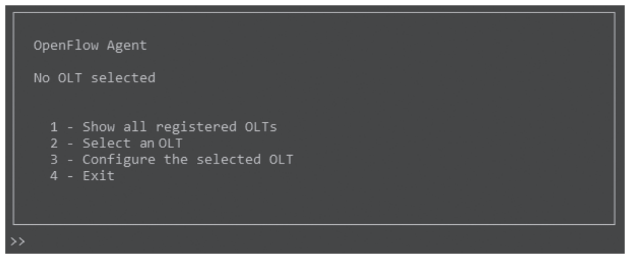


Fig. 8. Main menu of the menu-driven user interface.

5. MENU-DRIVEN USER INTERFACE

To simplify the management of the XGS-PON and the configuration of services (for instance, Internet) on the XGS-PON, a menu-driven user interface has been developed using OpenFlow. This application, coded in Python, can be used to create flows and meters in the controller, instead of directly using the ONOS API to configure services in the ONUs. To do this, the menu-driven user interface communicates internally with the ONOS API through HTTP requests using the GET, POST, and DELETE methods. Figure 8 shows the main menu of this menu-driven user interface, through which the OLT and the ONUs connected to this OLT are configured. The main menu consists of four options: "Show all registered OLTs," "Select an OLT," "Configure the selected OLT," and finally, "exit," to quit the user application. A notable distinction from other GUIs, such as SEBA's Network Edge Mediator (NEM) software, is that NEM requires configuring and activating all settings (services, bandwidths, OLTs, and ONUs) primarily through the Topology and Orchestration Specification for Cloud Applications (TOSCA) language. TOSCA is an open source language utilized for describing relationships and dependencies between services and applications. Once these configurations are dispatched, they can be viewed within NEM's graphical user interface. In contrast, our proposal offers a simpler approach in which configurations do not need to be programmed with code. Instead, parameter values (services, profiles, bandwidths, VLANs, OLTs, ONUs) are entered directly when requested by the menu-driver interface and transmitted transparently to ONOS.

To configure a registered OLT, the user must first select it through the second menu option ("Select an OLT"). To do so, the user can view all registered OLTs using the first menu option, i.e., "Show all registered OLTs," which also shows more information on the registered OLTs. Figure 9 shows information of the OLTs currently registered in the user interface, in this case, a single OLT, the Edgecore ASXvOLT16 Whitebox.

Once the OLT is selected, all configurations are made through the third option menu (Configure selected OLT). This option of the main menu opens a new menu with several functionalities, as shown in Fig. 10. Within this submenu, the user can view all UNI ports of all ONUs connected to the OLT through the first option, which also displays detailed information. The user can also obtain the current configuration of each UNI port from the ONUs using the second option. In addition, the user can create or delete a service from an ONU (associated with a UNI port) using the third and fourth options, respectively. To obtain the packet statistics of the ONUs (for each UNI port), the fifth option can be used.

Fig. 9. Information related to registered OLTs in the user interface.

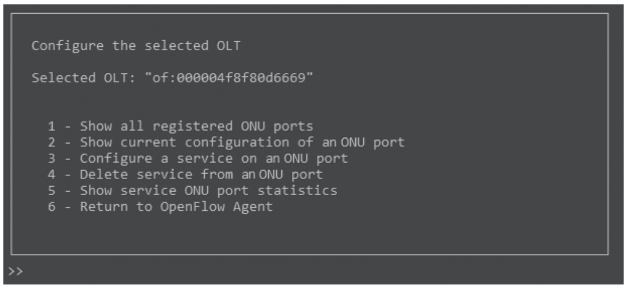


Fig. 10. Information related to registered OLTs in the user interface.

Finally, the sixth option shall be used to return to the main menu.

Furthermore, it is important to note that the menu-driven user interface can be easily modified to integrate new services (voice, video) or functionalities or even modify existing ones (adding IP filtering, including additional priority queues for the services, among others).

6. DESCRIPTION OF THE SDN PROPOSAL IMPLEMENTED ON AN XGS-PON TESTBED

A. XGS-PON Testbed Scenario

Our first step in demonstrating the feasibility and validity of our proposal is to describe the experimental XGS-PON architecture that has been deployed in the laboratory (Fig. 11). The XGS-PON infrastructure comprises a virtual OLT ASXvOLT16 Whitebox of Edge core (16x10G XGS-PON/NG PON2 ports) [45]. After the OLT ASXvOLT16, a 1:8 passive optical splitter is placed between the OLT and the ONUs to connect several ONUs to the XGS-PON. To account for the end-to-end attenuation due to distance in the XGS-PON, a 15 dB attenuator is connected. Finally, on the user side, we place sets of ONUs from different manufacturers. Specifically, the following ONUs are used:

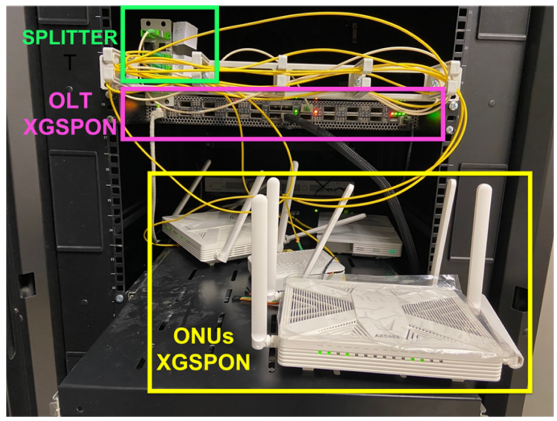


Fig. 11. Picture of the experimental XGS-PON testbed.

- Two ONUs of the manufacturer Azores model WAG-8F2W6 XGS/GPON: They provide four gigabit Ethernet ports, two plain old telephone service (POTS) ports, one RF port, 4×4 802.11ax 5G, and 2×2 802.11ax 2.4G Wi-Fi [46].
- One Broadcom ONU: It provides gigabit Ethernet ports, two POTS ports, 2×2 802.11ax at 2.4 GHzWi-Fi, and 4×4 802.11ax (prototype).
- One Bowie ONU model WAG-D10T: It offers one 10GE Ethernet port and/or one 2.5G Ethernet port (prototype).

All connected devices comply with the ITU-T G.9807.1 XGS-PON recommendations [4], supporting 9.95328 Gbps in the downstream and 9.95328 Gbps in the upstream.

B. XGS-PON Testbed Scenario

The SDN-OpenFlow agent has been set up inside a docker container to manage the XGS-PON configuration via the OLT, as shown in Fig. 12. Additionally, we have installed the ONOS controller (version 2.5.9) and a menu-driven user interface for streamlined network management and service configuration, each in their own docker containers. All three containers are hosted on a server connected directly to the OLT through its management port using the RPC protocol. To simplify the deployment process, a docker-compose file has been used to consolidate all configurations into a single action, resulting in the entire platform (agent, ONOS, user interface) being deployed simultaneously and directly.

This architecture is shown in Fig. 12, where communication between the different elements is described. Specifically, HTTP connections are observed between the menu-driven user interface and ONOS, OpenFlow connections between ONOS and the SDN agent, and, finally, the connection with the OLT is made by means of RPC. Specifically, the gRPC library has been used [47], connecting the RPC client developed in the OLT adapter with the RPC server in the OLT. When any instruction is executed from the RPC client to the RPC server, the adaptation layer will be instantiated by the RPC server, which will proceed to execute the required methods in the API of the OLT

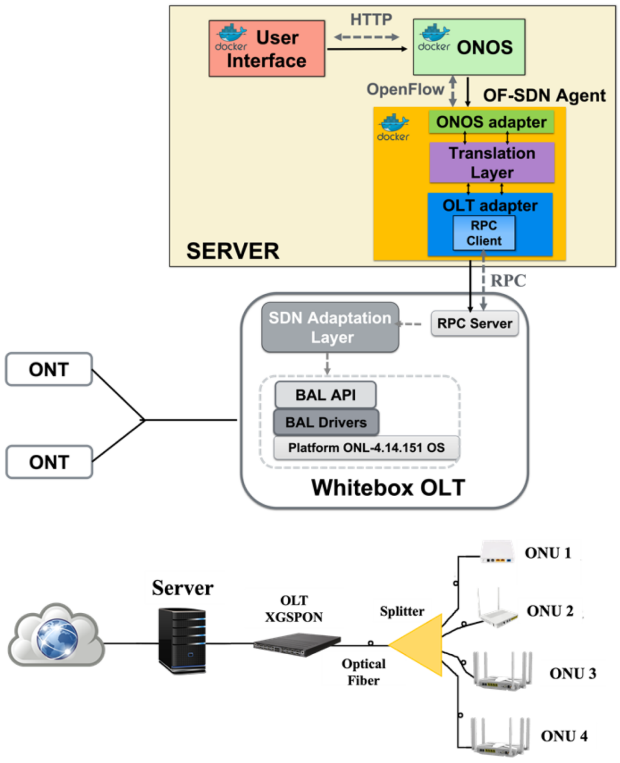


Fig. 12. Schema of the deployed XGS-PON testbed.

Fig. 13. Initial configuration with one ONU of two UNI ports.

chipset. For example, if you wish to set up a new Internet service, the SDN agent should configure a new bandwidth profile and an associated Alloc-ID, so the translation layer of the SDN agent assigns the corresponding values and sends the command to the OLT adapter to execute the CreateTrafficSchedulers function in the RPC client. Then, this information is received by the RPC server in the OLT through an RPC connection and calls the adaptation layer, in charge of creating the new Internet service, according to the specific API of the OLT provider.

Consequently, the adaptation layer must be modified depending on the OLT chipset and its API. However, our approach provides great potential and flexibility, as our SDN agent has been developed and tested for the Broadcom Maple BCM68628 driver (OLT driver), which is compatible with all BCM68620 series drivers [48]. The BCM68620 series is a high performance OLT PON SoC (System-on-a-Chip), supporting GPON, XGPON, XGS-PON, NGPON2, EPON, and 10G-EPON. But it also integrates support for existing and next-generation PON technologies through a common API, thus enabling a unified software design that can be applied to multiple PON protocols and technologies.

7. VALIDATION ANALYSIS AND RESULTS

This section explains the experimental validation of different functionalities of the SDN proposal on a legacy XGS-PON. Specifically, the registration of new ONUs will be tested experimentally, as well as the configuration/deletion of different types of Internet services (single and double tag).

A. Registration of New ONUs

The first functionality to be validated is to connect a new ONU to the XGS-PON, verifying that the OpenFlow agent activates it and registers a new OpenFlow port for each UNI port of the new ONU. Thus, at the beginning of the test, an ONU with two UNI ports was already registered (port_1, port_2), and one of the NNI ports (port_0) of the OLT was also registered, as shown in Fig. 13. This figure shows this initial configuration on the user interface, which is the output of the option "Show all registered ports" on the "Configure the selected device" menu. Afterwards, a new ONU (Azores model) with four UNI ports is connected, so the OpenFlow agent discovers it and activates it, associating an ONU ID. As shown in Fig. 14, the OpenFlow agent informs ONOS of the newly registered ONU with its associated UNI ports, and they all appear in the menu-driven user interface.

These registered ports have several important attributes, for example, the port numbered "port_1" shows the following:

- element: ID of the OLT;
- port: ID given to the port by the OpenFlow agent;
- isEnabled: informs that the port is enabled;
- Type: connection type (copper, optical fiber);
- portSpeed: link speed;
- portName: name given by the OpenFlow agent to the port, generated by concatenating the ONU serial number and the UNI port number on the ONU.

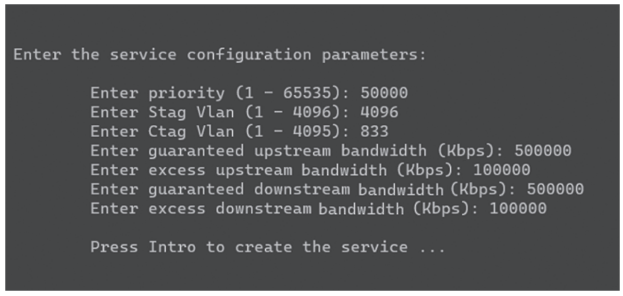
```
{
  "port_0": {
    "element": "of:000004f8f80d6669",
    "port": "20",
    "isEnabled": true,
    "type": "fiber",
    "portSpeed": 100,
    "annotations": {
      "adminState": "enabled",
      "portMac": "04:f8:f8:0d:66:69",
      "portName": "NNI"
    }
  },
  "port_1": {
    "element": "of:000004f8f80d6669",
    "port": "1879245060",
    "isEnabled": true,
    "type": "copper",
    "portSpeed": 1000,
    "annotations": {
      "adminState": "enabled",
      "portMac": "00:00:00:00:00:00",
      "portName": "AZRS6f47f15e-4"
    }
  },
  "port_2": {
    "element": "of:000004f8f80d6669",
    "port": "1879245059",
    "isEnabled": false,
    "type": "copper",
    "portSpeed": 0,
    "annotations": {
      "adminState": "enabled",
      "portMac": "00:00:00:00:00:00",
      "portName": "AZRS6f47f15e-3"
    }
  },
  "port_3": {
    "element": "of:000004f8f80d6669",
    "port": "1879245058",
    "isEnabled": false,
    "type": "copper",
    "portSpeed": 0,
    "annotations": {
      "adminState": "enabled",
      "portMac": "00:00:00:00:00:00",
      "portName": "AZRS6f47f15e-2"
    }
  },
  "port_4": {
    "element": "of:000004f8f80d6669",
    "port": "1879245057",
    "isEnabled": false,
    "type": "copper",
    "portSpeed": 0,
    "annotations": {
      "adminState": "enabled",
      "portMac": "00:00:00:00:00:00",
      "portName": "AZRS6f47f15e-1"
    }
  },
  "port_5": {
    "element": "of:000004f8f80d6669",
    "port": "1879179522",
    "isEnabled": false,
    "type": "copper",
    "portSpeed": 1000,
    "annotations": {
      "adminState": "enabled",
      "portMac": "00:00:00:00:00:00",
      "portName": "BRCM23456789-2"
    }
  },
  "port_6": {
    "element": "of:000004f8f80d6669",
    "port": "1879179521",
    "isEnabled": false,
    "type": "copper",
    "portSpeed": 10000,
    "annotations": {
      "adminState": "enabled",
      "portMac": "00:00:00:00:00:00",
      "portName": "BRCM23456789-1"
    }
  }
}

Press Intro to continue ...
```


Fig. 14. New configuration with one new ONU registered.

B. Configuration of Internet Services

After registering ONUs in the SDN system, services such as the Internet can be configured and sent to XGS-PON users (UNI ports in ONUs) using OpenFlow flows and meters. These flows and meters can be created through the ONOS web interface or through the menu-driver user interface designed. Specifically, the user interface was used for all tests. To validate Internet services, a single tag and a double tag service will be configured and tested in the XGS-PON.



```
Enter the service configuration parameters:

        Enter priority (1 - 65535): 50000
        Enter Stag Vlan (1 - 4096): 4096
        Enter Ctag Vlan (1 - 4095): 833
        Enter guaranteed upstream bandwidth (Kbps): 500000
        Enter excess upstream bandwidth (Kbps): 100000
        Enter guaranteed downstream bandwidth (Kbps): 500000
        Enter excess downstream bandwidth (Kbps): 100000

        Press Intro to create the service ...
```


Fig. 15. Single tag service configuration.

1. Configuration of a Single Tag Internet Service

For a single tag Internet service on a specific UNI port of one ONU (previously selected), the configuration is shown in Fig. 15. As can be observed, the configured C-tag VLAN is 833, the S-tag VLAN is 4096, which means there is no S-tag, and the guaranteed and excess bandwidths are 500 Mbps and 100 Mbps, respectively (maximum of 600 Mbps), for both downstream and upstream. Indeed, the IEEE 802.1Q-2005 QoS standard specifies that the VLAN identifier is encoded in a 12-bit field between values from 0 to 4095 reserved, so we will use 4096 to indicate the absence of the S-tag [49].

Furthermore, for this service, two flows are created in ONOS, one for downstream and another for upstream, and one meter, which is associated with both flows. The meter will be associated with two bands with a DROP parameter, one of 500,000 Kbps (500 Mbps) for the guaranteed bandwidth and the other of 600,000 (600 Mbps) for the maximum permitted bandwidth, with a meter identifier "id = 2." Thus, this configuration is encapsulated in JSON format and sent from this application to ONOS via an HTTP POST request.

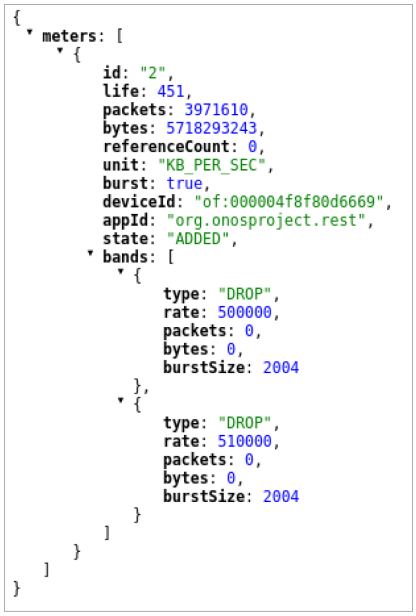
```
{
    "priority": 10000,
    "timeout": 0,
    "isPermanent": true,
    "deviceId": "of:000004f8f80d6669",
    "tableId": 0,
    "treatment": {
        "instructions": [
            {
                "type": "L2MODIFICATION",
                "subtype": "VLAN_ID",
                "vlanId": 833
            },
            {
                "type": "METER",
                "meterId": "2"
            },
            {
                "type": "OUTPUT",
                "port": "20"
            }
        ]
    },
    "selector": {
        "criteria": [
            {
                "type": "IN_PORT",
                "port": 1879245060
            },
            {
                "type": "VLAN_VID",
                "vlanId": 0
            }
        ]
    }
}
```


(a)


```
{
    "priority": 10000,
    "timeout": 0,
    "isPermanent": true,
    "deviceId": "of:000004f8f80d6669",
    "tableId": 0,
    "treatment": {
        "instructions": [
            {
                "type": "L2MODIFICATION",
                "subtype": "VLAN_ID",
                "vlanId": 0
            },
            {
                "type": "OUTPUT",
                "port": "1879245060"
            },
            {
                "type": "METER",
                "meterId": "2"
            }
        ]
    },
    "selector": {
        "criteria": [
            {
                "type": "IN_PORT",
                "port": 20
            },
            {
                "type": "VLAN_VID",
                "vlanId": 833
            }
        ]
    }
}
```


(b)

Fig. 16. JSON representation of the configured flows in the user application (a) upstream and (b) downstream.



```
{
  meters: [
    {
      id: "2",
      life: 451,
      packets: 3971610,
      bytes: 5718293243,
      referenceCount: 0,
      unit: "KB_PER_SEC",
      burst: true,
      deviceId: "of:000004f8f80d6669",
      appId: "org.onosproject.rest",
      state: "ADDED",
      bands: [
        {
          type: "DROP",
          rate: 500000,
          packets: 0,
          bytes: 0,
          burstSize: 2004
        },
        {
          type: "DROP",
          rate: 510000,
          packets: 0,
          bytes: 0,
          burstSize: 2004
        }
      ]
    }
  ]
}
```


Fig. 17. JSON representation of the configured meter in ONOS with guaranteed and maximum bandwidth.

Figures 16(a) and 16(b) show the configuration of both flows, upstream and downstream, respectively.

To demonstrate that the configuration has been correctly set up in ONOS, Fig. 17 shows the meter configuration obtained through a GET request in ONOS.

In addition, Fig. 18 shows an ONOS snapshot of the upstream flow created for the Internet service (the downstream flow is similar). As can be observed, the associated meter previously created "meterId = 2" appears in the flow.

Once ONOS has received each JSON configuration, both meters and flow types, it sends the same configuration to the OpenFlow agent through OpenFlow messages. Figure 19 shows the meter message, which is a message of type "OFPT_METER_MOD," encapsulating the data detailed above in the JSON configuration.

To corroborate that the Internet service has been deployed correctly in the OLT, Fig. 20 shows the status of the log inside the OLT. In this case, we can find XGS-PON parameters related to traffic schedulers, traffic queues, such as Alloc-IDs or GEM ports, instead of OpenFlow meters and flows. In fact, the parameters have a value of Alloc-ID and GEM port equal to 1024 and, after the GEM port configuration, the entry "Received classifier with O_VID: 833" reports the VLAN configuration. In addition, line 8 of Fig. 20 also shows the bandwidth configuration, where the CIR set to 500 Mbps corresponds to the guaranteed bandwidth, and the PIR set to 600 Mbps to the maximum bandwidth.

The Wireshark tool has been used to measure the bandwidth performance of the service. The tests have been carried out using the iperf tool, so an iperf client transmitting at 1 Gbps is connected to the UNI port of this ONU and an iperf server is connected to the OLT. Figure 21 shows the measured throughput of the Internet service using Wireshark

```
{
  flows: [
    {…},
    {…},
    {…},
    {…},
    {
      groupId: 0,
      state: "ADDED",
      life: 1173,
      liveType: "UNKNOWN",
      lastSeen: 1669804458066,
      packets: 154,
      bytes: 9548,
      id: "33495522324823134",
      appId: "org.onosproject.fwd",
      priority: 10000,
      timeout: 0,
      isPermanent: true,
      deviceId: "of:000004f8f80d6669",
      tableId: 0,
      tableName: "0",
      treatment: {
        instructions: [
          {
            type: "L2MODIFICATION",
            subtype: "VLAN_ID",
            vlanId: 833
          },
          {
            type: "OUTPUT",
            port: "20"
          },
          {
            type: "METER",
            meterId: "2"
          }
        ],
        deferred: [ ]
      },
      selector: {
        criteria: [
          {
            type: "IN_PORT",
            port: 1879245060
          },
          {
            type: "VLAN_VID",
            vlanId: 0
          }
        ]
      }
    },
```


Fig. 18. Flow configured in ONOS for the upstream channel.

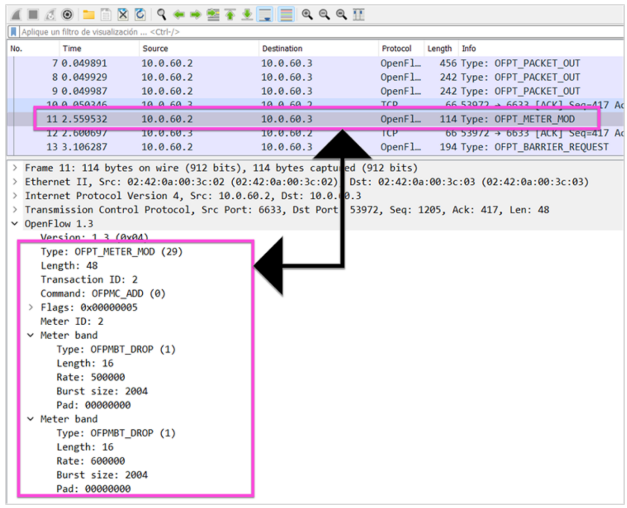


Fig. 19. OpenFlow messages between the OpenFlow agent and ONOS.

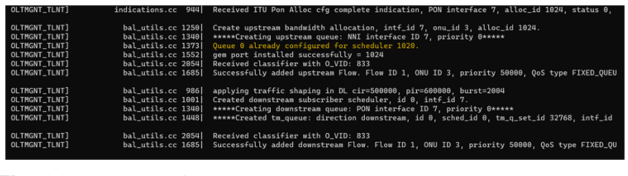


Fig. 20. Status of the log inside the OLT.

for the upstream channel (the downstream bandwidth is similar), which corresponds to the maximum bandwidth set to 600 Mbps.

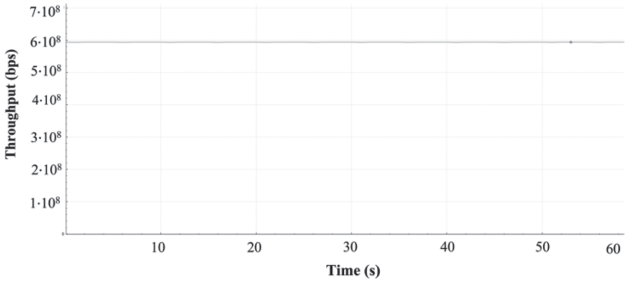


Fig. 21. Throughput of the upstream Internet service measured with Wireshark.

```
Enter the service configuration parameters:

        Enter priority (1 - 65535): 50000
        Enter ONU port: 1879245060
        Enter Stag Vlan (1 - 4096): 4096
        Enter Ctag Vlan (1 - 4095): 833
        Enter guaranteed upstream bandwidth (Kbps): 500000
        Enter excess upstream bandwidth (Kbps): 10000
        Enter guaranteed downstream bandwidth (Kbps): 500000
        Enter excess downstream bandwidth (Kbps): 10000

        Press Intro to create the service ...
```

Fig. 22. Configuration of a new Internet service with different bandwidth restrictions.

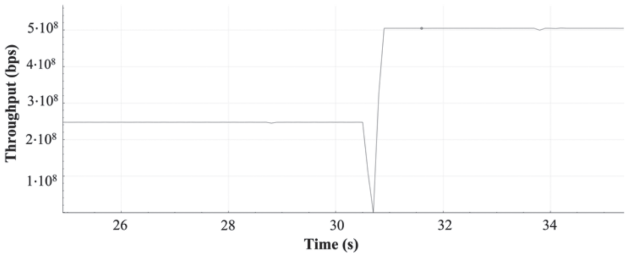


Fig. 23. Real-time throughput performance of a dynamic Internet service measured with Wireshark with the SDN proposal (upstream).

On the other hand, to demonstrate the viability of the SDN proposal in a real network environment, the speed of configuring and deploying the service with the SDN proposal has been analyzed. Thus, the test is performed considering an initial Internet service with a bandwidth of 250 Mbps and then modifying the bandwidth restrictions by increasing it to 510 Mbps, as shown in Fig. 22, in the user interface. Thus, Fig. 23 shows the real-time throughput performance of the dynamic Internet service measured with Wireshark in the upstream channel. As can be observed, the time elapsed from launching the Internet service in the user interface (at 30 s) to deployment (at 31 s) is very short, around 1 s. This leads us to conclude that our proposal is fully feasible in terms of speed.

2. Configuration of a Double Tag Internet Service

For a double tag Internet service on an ONU, the configuration via the menu-driven user interface is shown in Fig. 24, which, as introduced above, has an additional VLAN tag (S-tag) associated with it. The only difference from the single

```
Enter the service configuration parameters:

        Enter priority (1 - 65535): 10000
        Enter Stag Vlan (1 - 4096): 900
        Enter Ctag Vlan (1 - 4095): 833
        Enter guaranteed upstream bandwidth (Kbps): 500000
        Enter excess upstream bandwidth (Kbps): 10000
        Enter guaranteed downstream bandwidth (Kbps): 500000
        Enter excess downstream bandwidth (Kbps): 10000

        Press Intro to create the service ...
```

Fig. 24. Configuration of a double tag Internet service.

Fig. 25. Flows configured in ONOS for the upstream channel.

tag service is the configured S-tag VLAN value, which in this case is 900. The C-tag VLAN is 833, and the guaranteed and excess bandwidths are 500 Mbps and 10 Mbps, respectively (maximum of 510 Mbps), for both downstream and upstream. Therefore, the meter associated with the flows is the one created for the single tag service (Fig. 24).

For this Internet service, four flows must be created in ONOS, two for downstream and two for upstream (one pair for each VLAN). Then, this configuration is encapsulated in JSON format and sent from this application to ONOS via an HTTP POST request. To demonstrate that the configuration has been set up correctly in ONOS, Figs. 25 and 26 show the configuration of the flows in the upstream (two flows) and downstream (two flows), respectively, obtained through GET requests in ONOS.

To check the correct deployment of the Internet service in the OLT, Fig. 27 shows the corresponding log in the OLT. In the log status, parameters of the XGS-PON standard related

```
▾ flows: [
  ▸ {…},
  ▾ {
      groupId: 0,
      state: "ADDED",
      life: 82,
      liveType: "UNKNOWN",
      lastSeen: 1669807508070,
      packets: 3971556,
      bytes: 5718288923,
      id: "33495525160523057",
      appId: "org.onosproject.fwd",
      priority: 10000,
      timeout: 0,
      isPermanent: true,
      deviceId: "of:000004f8f80d6669",
      tableId: 0,
      tableName: "0",
    ▾ treatment: {
      ▾ instructions: [
        ▾ {
            type: "L2MODIFICATION",
            subtype: "VLAN_POP"
          },
        ▾ {
            type: "TABLE",
            tableId: "1"
          }
        ],
        deferred: [ ]
      },
    ▾ selector: {
      ▾ criteria: [
        ▾ {
            type: "IN_PORT",
            port: 20
          },
        ▾ {
            type: "VLAN_VID",
            vlanId: 900
          }
        ]
      }
    },
  ▸ {…},
```

```
▾ flows: [
  ▾ {
      groupId: 0,
      state: "ADDED",
      life: 273,
      liveType: "UNKNOWN",
      lastSeen: 1669807708071,
      packets: 3971559,
      bytes: 5718289109,
      id: "33495532362349 87",
      appId: "org.onosproject.fwd",
      priority: 10000,
      timeout: 0,
      isPermanent: true,
      deviceId: "of:000004f8f80d6669",
      tableId: 1,
      tableName: "1",
    ▾ treatment: {
      ▾ instructions: [
        ▾ {
            type: "L2MODIFICATION",
            subtype: "VLAN_ID",
            vlanId: 0
          },
        ▾ {
            type: "OUTPUT",
            port: "1879245060"
          },
        ▾ {
            type: "METER",
            meterId: "2"
          }
        ],
        deferred: [ ]
      },
    ▾ selector: {
      ▾ criteria: [
        ▾ {
            type: "IN_PORT",
            port: 20
          },
        ▾ {
            type: "VLAN_VID",
            vlanId: 833
          }
        ]
      }
    },
  ▸ {…}
```


Fig. 26. Flows configured in ONOS for the downstream channel.

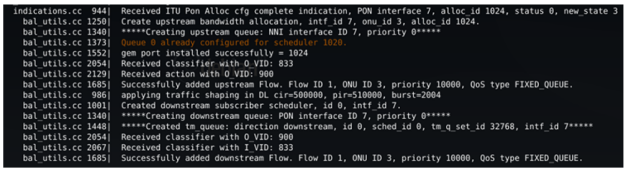


Fig. 27. Status of the log inside the OLT.

to the configuration of services can be observed, such as both VLANS (900, 833), Alloc-IDs (1024), GEM ports (1024), or the bandwidth requirements in the upstream and downstream channels.

Finally, Fig. 28 shows the measured throughput of the Internet service using Wireshark for the upstream direction (the same performance for the downstream). The tests have been carried out using the iperf tool, so an iperf client transmitting at 1 Gbps is connected to the UNI port of this ONU, and an iperf server is connected to the OLT. In Fig. 28, it can be seen that the ONU is allocated the maximum bandwidth of 510 Mbps previously configured using the menu-driver user interface.

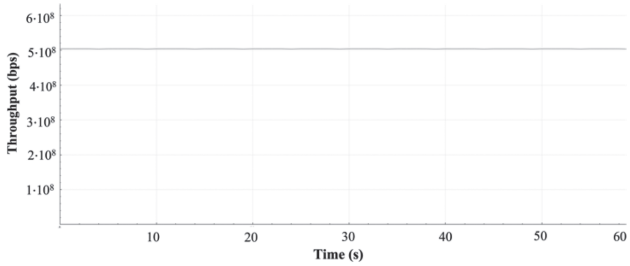


Fig. 28. Throughput of the upstream Internet service measured with Wireshark.

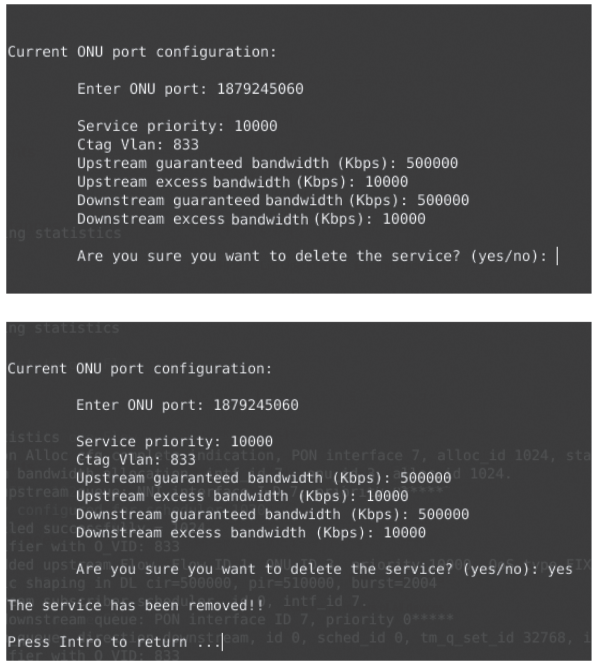


Fig. 29. Deletion of the Internet service in the menu-driven user interface.

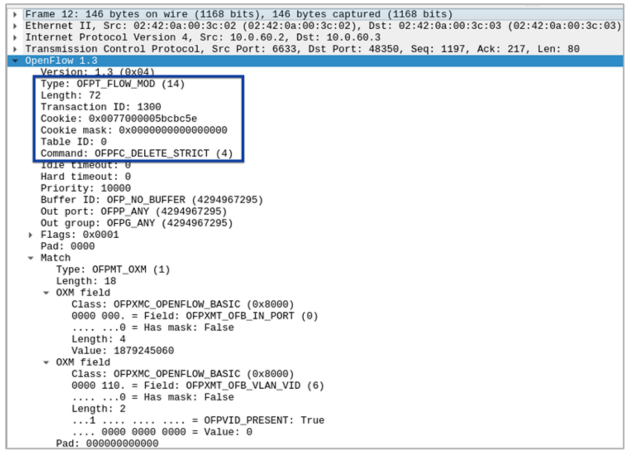


Fig. 30. OpenFlow messages between the OpenFlow agent and ONOS to delete the upstream flow.

C. Deletion of Internet Services

In this section, we test the deletion of an Internet service through the SDN proposal developed. To do so, we will consider that initially we had a service with maximum bandwidth of 510 Mbps (guaranteed plus excess bandwidth), and as shown in Fig. 29, we will proceed to delete this service using the menu-driven user interface.

Once ONOS receives this configuration in JSON format, it sends the same configuration to the OpenFlow agent through OpenFlow messages, namely, the removal of the two flows (upstream, downstream) and the associated meters. Figure 30 shows the flow message (type "OFPT_FLOW_MOD"), which includes the command "OFPFC_DELETE_STRICT," to delete the corresponding flow. In addition, Fig. 31 shows the OpenFlow message (type "OFPT_METER_MOD"), which

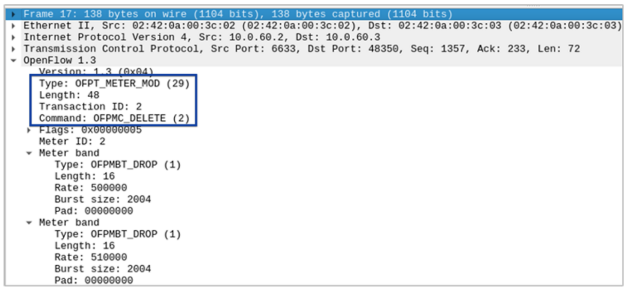


Fig. 31. OpenFlow messages between the OpenFlow agent and ONOS to delete the associated meter of the flow.

Fig. 32. Status of the log inside the OLT after the deletion of a service.

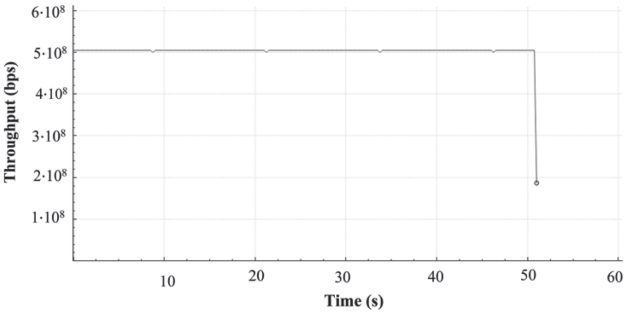


Fig. 33. Throughput of the upstream Internet service measured with Wireshark using the SDN approach (upstream).

includes the command "OFPMC_DELETE," to delete the associated meter.

To test that the Internet service has been correctly deleted in the OLT, Fig. 32 shows the status of the log inside the OLT. In the log status, parameters of the XGS-PON related to the deletion of services can be observed, such as Alloc-IDs (1024), GEM ports (1024), traffic schedulers (sched_id = 0), and traffic queues (tm_queue = 0).

Finally, Fig. 33 shows the real-time evolution of the bandwidth using Wireshark to analyze the removal of the Internet service in the upstream channel. As can be seen in this graph, the removal of the service takes place almost automatically, about 1–2 s after sending the command from the user interface (second 48).

8. CONCLUSIONS

This paper presents the development and design of an SDN OpenFlow agent for managing and configuring PONs, specifically XGS-PON architectures, with the ONOS controller. The OpenFlow agent behaves like an OpenFlow switch, enabling communication with ONOS using OpenFlow messages. Additionally, it communicates with OLTs using the vendor-specific APIs of the OLT chipset, without the need to emulate SDN abstract layers in hardware devices. To evaluate the proposal, we conducted experiments using OpenFlow and ONOS on a White Box OLT for XGS-PON architectures. Our results demonstrate that the proposed solution allows for the real-time configuration and deletion of various types of Internet services, catering to the QoS requirements of service providers and operators. Furthermore, the proposed SDN OpenFlow agent is both scalable and flexible. It is composed of distinct, simple blocks, namely, the ONOS adapter, translation layer, and OLT adapter, which facilitates the seamless integration of new SDN functionalities. The integration of new functionalities does not usually require modifications in all blocks, but only in some of them, mainly ONOS and the translation layer. As a result, our OpenFlow-based SDN agent has a comparatively smooth learning curve, and since its deployment involves fewer containers, it makes it less complex compared to existing alternative approaches.

In the future, we plan to flexibly integrate other critical services for operators and service providers, such as multicast (video) or voice, into the SDN agent. It is worth noting that the menu-driven user interface is highly flexible and can be readily adapted to incorporate new services or functionalities, as well as modify existing ones (additional IP filtering, additional service priorities, and so on). In fact, our menu-driven user interface offers a simpler approach in which configurations are entered directly when requested by the applications and transmitted transparently to ONOS. Another research line is focused on employing the SDN-enabled XGS-PON testbed as the supporting wired infrastructure for a set of use cases related to connected vehicles, edge computing, and computation offloading, which are currently under development.

Funding. Consejería de Educación, Junta de Castilla y León (VA231P20); European Regional Development Fund (VA231P20); Ministerio de Ciencia e Innovación (PID2020-112675RB-C42); Agencia Estatal de Investigación (PID2020-112675RB-C42).

Acknowledgment. We are grateful to Telnet R.I. for the supply of XGS-PON equipment (OLT, ONUs).

Data availability. The code and the instructions to install and deploy the SDN agent are available on GitHub at [39].

REFERENCES

1. FTTH Council, "FTTH market forecasts 2022–2027" [Accessed 1 March 2023], https://www.ftthcouncil.eu/knowledge-centre/all-publications-and-assets/1462/ftth-market-forecasts-2022-2027.
2. Broadband Market Trends, "Global fiber-to-the-home/building (FTTH/B) industry" [Accessed 1 March 2023], https://www.reportlinker.com/p05817990/Global-Fiber-to-the-Home-Building-FTTH-B-Industry.html?utm_source=GNW.
3. Global Market Insights, "GPON market" [Accessed 10 January 2023], https://www.gminsights.com/industry-analysis/gigabit-passive-optical-network-equipment-market?gclid=Cj0KCQiA-oqdBhDfARIsAO0TrGHBClfRzrbJv25T9rCqME-mIyYJytRvrsoh91FeFi-SCVf9byApzM8aAgVLEALw_wcB.
4. "10-gigabit-capable symmetric passive optical network (XGS-PON)," ITU-T Recommendation G.9807.1 (2023) [Accessed 10 February 2023], https://www.itu.int/rec/T-REC-G.9807.1/en.
5. K. Wieland, "XGS-PON moves center stage," Light Reading (2022) [Accessed 10 February 2023], https://www.lightreading.com/partner-perspectives-(sponsored-content)/xgs-pon-moves-center-stage/a/d-id/775326.

6. K. J. Kerpez, J. M. Cioffi, G. Ginis, M. Goldburg, S. Galli, and P. Silverman, "Software-defined access networks," IEEE Commun. Mag. **52**(9), 152–159 (2014).
7. A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, "Software defined optical networks (SDONs): a comprehensive survey," IEEE Commun. Surv. Tutorials **18**, 2738–2786 (2016).
8. Open Networking Foundation (ONF), OpenFlow [Accessed 20 January 2023], https://www.opennetworking.org/.
9. Network Configuration Working Group, NETCONF Configuration Protocol [Accessed 20 January 2023], https://tools.ietf.org/wg/netconf.
10. A. Bierman, M. Bjorklund, and K. Watsen, "RESTCONF Protocol," IETF RFC 8040 (2017) [Accessed 20 January 2023], https://tools.ietf.org/html/rfc8040.
11. S. S. W. Lee, K.-Y. Li, and M.-S. Wu, "Design and implementation of a GPON-based virtual OpenFlow-enabled SDN switch," J. Lightwave Technol. **34**, 2552–2561 (2016).
12. S. S. W. Lee, K. Li, W. Liu, C. Chen, H. Fang, and T. Wong, "Embedding bandwidth-guaranteed network-based virtual Ethernet switches in SDN networks," J. Lightwave Technol. **35**, 5041–5055 (2017).
13. R. G. Clegg, J. Spencer, R. Landa, M. Thakur, J. Mitchell, and M. Rio, "Pushing software defined networking to the access," in *3rd European Workshop on Software Defined Networks* (2014), pp. 31–36.
14. N. Merayo, D. de Pintos, J. C. Aguado, I. de Miguel, R. J. Durán, P. Fernández, R. M. Lorenzo, and E. J. Abril, "Experimental validation of an SDN residential network management proposal over a GPON testbed," Opt. Switching Netw. **42**, 100631 (2021).
15. N. Merayo, D. de Pintos, J. C. Aguado, I. de Miguel, R. J. Durán, P. Fernández, R. M. Lorenzo, and E. J. Abril, "An experimental OpenFlow proposal over legacy GPONs to allow real-time service reconfiguration policies," Appl. Sci. **11**, 903 (2021).
16. The Open Source Project VOLTHA [Accessed 22 January 2022], https://opennetworking.org/voltha.
17. S. Das, "From CORD to SDN Enabled Broadband Access (SEBA) [Invited Tutorial]," J. Opt. Commun. Netw. **13**, A88–A99 (2021).
18. N. Merayo, J. C. Aguado, I. De Miguel, R. J. Durán Barroso, P. Fernández, R. M. Lorenzo, and E. J. Abril, "Deployment of an SDN-based GPON control agent to manage network configurations," in *International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, Maldives, 2022.
19. "The ONOS SDN controller" [Accessed 22 January 2022], https://opennetworking.org/onos/.
20. H. Khalili, S. Sallent, J. R. Piney, and D. A. Rincón, "A proposal for an SDN-based SIEPON architecture," Opt. Commun. **403**, 9–21 (2017).
21. C. Li, W. Guo, W. Wang, W. Hu, and M. Xia, "Programmable bandwidth management in software-defined EPON architecture," Opt. Commun. **370**, 43–48 (2016).
22. H. Yang, J. Zhang, Y. Zhao, J. Wu, Y. Ji, Y. Lin, J. Han, and Y. Lee, "Experimental demonstration of remote unified control for OpenFlow-based software-defined optical access networks," Photon. Netw. Commun. **31**, 568–577 (2016).
23. X. Liang, Q. Tian, F. Wang, W. Yu, and X. Xin, "A dynamic resource allocation based on network traffic prediction for sliced passive optical network," in *19th International Conference on Optical Communications and Networks (ICOCN)* (2021).
24. F. Wang, B. Liu, L. Zhang, F. Jin, O. Zhang, Q. Tian, F. Tian, L. Rao, and X. Xin, "Dynamic bandwidth allocation based on multiservice in software-defined wavelength-division multiplexing time-division multiplexing passive optical network," Opt. Eng. **56**, 036104 (2017).
25. I.-S. Hwang, A. Rianto, and A. F. Pakpahan, "Software-defined peer-to-peer file sharing architecture for TWDM PON," in *Proceedings of the 27th Wireless and Optical Communication Conference (WOCC)*, Hualien, Taiwan, 2018.
26. A. T. Liem, I.-S. Hwang, E. Ganesan, O. Lengkong, and L. Jallow, "Enabling SDN in hybrid Xhaul-TWDM-PON networks for interactive video applications," in *2nd International Conference on Cybernetics and Intelligent System (ICORIS)* (2020).
27. C. Centofanti, A. Marotta, D. Cassioli, F. Graziosi, N. Sambo, L. Valcarenghi, C. Bernard, and H. Roberts, "Slice management in SDN PON supporting low-latency services," in *European Conference on Optical Communication (ECOC)* (2022).
28. R. Flores Moyano, D. Fernández, N. Merayo, C. M. Lentisco, and A. Cárdenas, "NFV and SDN-based differentiated traffic treatment for residential networks," IEEE Access **8**, 34038–34055 (2020).
29. B. Yan, J. Zhou, J. Wu, and Y. Zhao, "SDN based energy management system for optical access network," in *Proceedings of the 9th International Conference on Communications and Networking in China*, Shanghai, China, 2014.
30. C. Quian, Y. Li, O. Zhang, B. Cao, Z. Li, and M. Wang, "Staged priority-based dynamic bandwidth allocation in software-defined hybrid passive optical network," Opt. Eng. **57**, 126101 (2018).
31. S. McGettrick, F. Slyne, N. Kitsuwan, D. B. Payne, and M. Ruffini, "Experimental end-to-end demonstration of shared N:M dual-homed protection in SDN-controlled long-reach PON and pan-European core," J. Lightwave Technol. **34**, 4205–4213 (2018).
32. M. Wang, G. Simon, I. Amigo, L. A. Neto, L. Nuaymi, and P. Chanclou, "SDN-based RAN protection solution for 5G, an experimental approach," in *International Conference on Optical Network Design and Modeling (ONDM)* (2021).
33. L. Zhou, G. Peng, and N. Chand, "Demonstration of a novel software-defined Flex PON," Photon. Netw. Commun. **29**, 282–290 (2015).
34. P. Parol and M. Pawlowski, "Future proof access networks for B2B applications," Informatica **38**, 193–204 (2014).
35. A. Amokrane, J. Hwang, J. Xiao, and N. Anerousis, "Software defined enterprise passive optical network," in *IEEE International Conference on Network Service Management* (2014), pp. 406–441.
36. Open vSwitch, "Production quality, multilayer open virtual switch" [Accessed 10 January 2022], https://www.openvswitch.org/.
37. T. Suzuki, Y. Koyasako, K. Nishimoto, K. Asaka, T. Yamada, J.-I. Kani, T. Shimada, and T. Yoshida, "Demonstration of IEEE PON abstraction for SDN enabled broadband access (SEBA)," J. Lightwave Technol. **39**, 6434–6442 (2021).
38. Y. Koyasako, T. Suzuki, T. Hatano, T. Shimada, and T. Yoshida, "Demonstration of real-time motion control on a 10G-EPON edge computing platform with SDN enabled broadband access," J. Opt. Commun. Netw. **14**, 951–959 (2022).
39. N. Merayo, "OpenFlow SDN agent in GPONs," GitHub (2023) [Accessed 25 April 2023], https://github.com/GCOdeveloper/OpenFlow_Agent.
40. "Information technology—Open Systems Interconnection—Remote Procedure Call (RPC)," ISO/IEC 11578:1996 (1996) [Accessed 10 September 2022], https://www.iso.org/standard/2229.html.
41. "Python-OpenFlow Library Kytos SDN" [Accessed 10 October 2022], https://github.com/kytos/python-openflow.
42. Wireshark [Accessed February 2023], https://www.wireshark.org/.
43. Open Networking Foundation, "OpenFlow Switch Specification, Version 1.3.1," ONF TS-007 (2012) [Accessed January 2023], https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf.
44. "ONU management and control interface (OMCI) specification," ITU-T Recommendation G.988, Amendment 3 (2020).
45. Edge Core Networks [Accessed 10 February 2023], https://www.edge-core.com/.
46. ONT Azores WAG-8F2W6 [Accessed 1 February 2023], https://azoresnetworks.com/products/wag-8f2w6.
47. "gRPC protocol" [Accessed 10 October 2022], https://grpc.io.
48. Broadcom, "BCM68620 Universal OLT PON MAC" [Accessed 1 February 2023], https://www.broadcom.com/products/broadband/xpon/bcm68620.
49. "IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks," IEEE 802.1Q-2005 (2006) [Accessed 3 February 2022], https://standards.ieee.org/ieee/802.1Q/3495/.

David de Pintos received a degree in telecommunication engineering in 2019 from the University of Valladolid, Spain. He has worked as a technical operator at the company SATEC and as a research fellow at the University of

Valladolid for two years. Currently, he is working at CTAG in the embedded software department.

Noemí Merayo received a degree in telecommunication engineering from the University of Valladolid, Spain, in February 2004 and the Ph.D. degree from the Optical Communication Group at the University of Valladolid in July 2009. Since 2005, she has worked as a lecturer at the University of Valladolid. She has also been a visiting research fellow at the University of Hertfordshire in the Optical Networks Group, Science and Technology Research Institute (STRI); at the TOyBA research group of the University of Zaragoza; and more recently in the Technical University of Munich (TUM). Her research focuses on the design and performance evaluation of optical networks, especially passive optical networks, and the application of artificial intelligence techniques. Dr. Merayo is currently coordinating the master's degree in physics and technology of lasers at the University of Valladolid and the University of Salamanca.

Carlos Manuel Sangrador received a degree in telecommunication engineering in 2021 at the University of Valladolid. He has worked as a research fellow at the University of Valladolid for two years. Currently, he works at the GMV company in the Verification and Validation Department as a software tester of ITS systems, and he is also studying for a master's degree in telecommunications engineering at the Open University of Catalonia.

Juan Carlos Aguado received the Telecommunication Engineer and Ph.D. degrees from the University of Valladolid, Spain, in 1997 and 2005, respectively. Since 1998, he has been working as a junior lecturer at the University of Valladolid, where he is currently an associate professor. His current research interests focus on design and performance evaluation of optical networks and the application of artificial intelligence techniques. He has also been a postdoctoral researcher with the Group of Transmisiones Ópticas de Banda Ancha (TOyBA) at University of Zaragoza.

Ignacio de Miguel (Senior Member, IEEE) received a degree in telecommunication engineering and the Ph.D. degree from the University of Valladolid (UVa), Spain, in 1997 and 2002, respectively. He is currently an associate professor at UVa and the coordinator of the master's degree in telecommunication engineering. He has also been a visiting research fellow at University College London, UK. His main research interests include the design, control, and performance evaluation of communication infrastructures, optical networks, and edge computing and the application of artificial intelligence techniques in these environments. He has published more than 40 papers in international journals and more than 170 conference papers. Dr. de Miguel has been a member of the Technical Program Committee of several international conferences, besides being the Chair of the TPC and the Local Organizing Committee of NOC 2009. He was a recipient of the Nortel Networks Prize to the Best Ph.D. Thesis on Optical Internet in 2002, awarded by the Spanish Institute and the Association of Telecommunication Engineers (COIT/AEIT).

Ramón J. Durán Barroso received his Telecommunication Engineer degree in 2002 and his Ph.D. degree in 2008, both from the University of Valladolid (UVa), Spain. He works as an associate professor at the University of Valladolid. His current research focuses on the use of artificial intelligence techniques for the design, optimization, and operation of future heterogeneous networks, multi-access edge computing, and network function virtualization. Dr. Durán Barroso is the coordinator of the H2020 MSCA IoTalentum project composed of 14 partners, and he has been the coordinator of the Spanish Research Thematic Network "Go2Edge: Engineering Future Secure Edge Computing Networks, Systems and Services" composed of 15 entities. He has been the UVa leader in another six European and national projects. The dissemination activity of his research has produced 48 papers in JCR journals and more than 130 conference papers.