JOURNAL OF
Optical Communications
and Networking

Software-defined networking agent for integrating voice, data, and video services into XGS-PON architectures

Noemí Merayo,* David de Pintos, Juan Carlos Aguado, Ignacio de Miguel, Ramón J. Durán Barroso, and Rubén M. Lorenzo

Optical Communications Group, Universidad de Valladolid, Valladolid, Spain
**noemer@uva.es*

Received 24 May 2024; revised 29 July 2024; accepted 9 September 2024; published 24 September 2024

Software-defined networking (SDN) provides an efficient framework for managing passive optical networks (PONs) with equipment from different manufacturers, technologies, and standards. Thus, we recently proposed a robust and flexible SDN-OpenFlow agent to configure and manage 10 gigabit symmetric passive optical networks (XGS-PONs) to support Internet (data) services. That SDN agent communicates with the SDN controller via the OpenFlow protocol and with the optical line terminal (OLT) of the PON via the manufacturer's chipset-specific API (application programming interface). In this paper, we significantly extend that SDN agent by incorporating the capability to support two essential services for network operators and Internet service providers, specifically voice over IP (VoIP) and video services. The required extensions and configuration procedures on the different layers that compose the SDN agent are described, and an experimental validation of the extended SDN agent in an XGS-PON is conducted, demonstrating its effectiveness in the integration of those two services. The source code of the SDN agent has been made open and available on GitHub.

https://doi.org/10.1364/JOCN.531051

1. INTRODUCTION

Recent market studies show the constant penetration rate growth of fiber to the home/building (FTTH/B) technologies. By the end of 2023, the Europe region could have 230 million homes passed by full-fiber infrastructure, and the forecasts predict 310 million by 2028. At the same time the number of subscribers will increase from 108 million in 2023 to 196 million in 2028. This way, the coverage rate is expected to increase from 65% nowadays to 88% by 2028. In this context, passive optical network (PON) technologies are gradually gaining market share over point-to-point fiber technology. Some reasons for this are the strong focus on sustainability and power consumption reduction together with the continuous improvements of PON technology [1]. These access technologies consist of an optical line terminal (OLT) located at the central office and an optical network unit (ONU) at the user end, typically in a tree-like architecture. They operate as a point-to-multipoint network in the downstream link and employ multiple access through time-division in the upstream link.

In this way, the global PON market is nowadays dominated by GPON technology (PONs based on the gigabit standard), which delivers 2.5G downstream and 1.25G upstream. However, 10-GPON technology is currently engaged in a process of equipment cost reduction. Moreover, this technology is back compatible with GPON, making it possible to make a smooth transition from one technology to the other. As a consequence, although XGS-PON (10G PON) [2] only represents 15% of the overall PON market in the USA in 2021, it is expected to reach 55% by 2026. This is also encouraged by the opportunity of reducing costs in other services by incorporating XGS-PON, as for example, serving dense urban areas with many small cell 5G antennas [3]. Therefore, a smooth, gradual, and flexible transition from GPON to XGS-PON is very likely to occur when QoS (quality of service) requirements and bandwidth demands require it. One of the advantages of PON is that operators can use the same optical fiber infrastructure and a portion of the existing equipment, meaning that the migration does not result in an increase in OPEX expenses [3]. In fact, it is not necessary to upgrade both ends of the network (OLT and user-side ONU) simultaneously. By replacing OLT cards with "combo" PON cards at the central office, operators can offer both XGS-PON and GPON services simultaneously. This enables ONU devices to be upgraded to XGS-GPON-compatible devices when required.

On the other hand, this convergence of various PON technologies and the growing demand for services with stringent QoS standards have increased the complexity of configuring

PON infrastructures. Furthermore, PONs are frequently overseen by proprietary and rigid network management systems. The integration of software-defined networking (SDN) into PONs can effectively tackle these issues, virtualizing control and management PON functions to enable network optimization, streamline operations, and foster the creation of new services [4]. SDN separates the control plane from the data plane, providing centralized and intelligent network management by using protocols like OpenFlow [5], NETCONF [6], or RESTCONF [7]. This way, SDN empowers network operators with centralized control over network management operations and automation capabilities for intricate and time-consuming tasks, minimizing the need for manual interventions. In addition, SDN technology enables rapid network configuration and supports PON devices from different manufacturers and technologies, and the use of software-based controllers to communicate with the underlying hardware infrastructure means that network operators can monitor the network independently of the underlying technology [8].

Given these benefits, in the literature we can find multiple proposals exploring the integration of SDN into PON architectures to achieve different objectives. In this way, a high number of studies propose that bandwidth management (DBAs, dynamic bandwidth algorithms) and service configuration (QoS) policies are performed by an SDN controller external to the PON, both in simulation scenarios [9–11] or in experimental settings using testbeds [12,13]. Alternative SDN implementations concentrate on overseeing residential network management within PONs [14]. Concerning energy protection in PONs, there are proposals integrating SDN, such as [15–17]. Additionally, SDN has been employed in virtual PONs (VPONs) to allocate bandwidth [18] or to coordinate diverse services [19]. Furthermore, various proposals recommend extending current OpenFlow protocols for direct implementation within PON technologies, aligning with their protocols [20,21]. On the other hand, some research focuses on enabling PON devices (OLTs and ONUs) to be managed through SDN [22–24]. Indeed, numerous proposals introduce an abstraction layer to enable SDN-based control of PON devices. These methods involve the deployment of a virtual layer within PON devices that communicates with OLTs and ONUs, which adds complexity to the network and deviates from an integrated, inherent solution. As a result, alternative approaches opt to sidestep this complexity by designing SDN agents capable of translating SDN commands into the inherent PON configuration. Within this line of research, the VOLTHA (Virtual OLT Hardware Abstraction) project [25], which abstracts the PON to a programmable switch managed by an SDN controller and interacts with the PON devices through the manufacturer proprietary language, stands out. VOLTHA is part of the SEBA (SDN Enabled Broadband Access) project [26], an open-source initiative supported by the ONF (Open Networking Foundation). SEBA uses a unified API (application programming interface) to abstract OLT and ONU devices and manage them through an SDN controller, which is why some proposals use it [27,28]. OB-BAA (Open Broadband-Broadband Access Abstraction), on the other hand, is an initiative led by the Broadband Forum organization that seeks to standardize and define open interfaces (specifications and APIs) for the management of broadband access devices, i.e., multi-vendor devices in a uniform way. Thus, OB-BBA and VOLTHA differ in their approaches and main functionalities [29]. Such approaches hold great promise due to the rapid proliferation and coexistence of diverse PON technologies (GPON, EPON, 10G PON) and numerous PON device manufacturers. Indeed, this proliferation has also introduced compatibility challenges when managing these devices collectively. Although all PON devices adhere to the same standards, there are no universally standardized PON devices. This is because each manufacturer employs specific proprietary software to manage PONs, resulting in substantial variations in the way access methods and the configuration of PON device chipsets (OLTs and ONUs) are implemented. Hence, SDN provides an efficient solution to this issue [4,8]. It offers an efficient way of transparently and concurrently managing PON devices from various manufacturers, diverse technologies (e.g., GPON, 10GPON), and different standards (IEEE, ITU-T).

As a result, close to the latter approaches (more specifically VOLTHA), we have recently developed and experimentally validated a robust SDN OpenFlow agent that communicates directly with the APIs of OLT chipsets, namely, in XGS-PON infrastructures [30]. Our approach is scalable, thanks to its modular design composed of differentiated blocks. This modular structure facilitates the seamless integration of new SDN functionality within a single block, requiring minimal adjustments to other blocks. This contrasts with approaches such as VOLTHA, where the addition of new functionalities requires modifications in multiple blocks. In addition, the agent shows great flexibility, as although it can be managed by tools such as SADIS (Subscriber and Device Information Service) or ONOS (Open Network Operating System), it also offers a user-friendly menu-driven interface. This design ensures that the learning curve for managing the OpenFlow-based SDN agent is relatively low. It is also easy to install, requiring only a few Docker containers to deploy, unlike other methods, such as VOLTHA, which require a larger number of containers, increasing complexity. In summary, the main advantage of this proposal compared to VOLTHA is its simplicity. As a result, our approach offers a fast SDN solution for configuring and testing new functionalities with minimal programming changes required at specific layers of the developed SDN agent. This approach significantly reduces the time and effort required to make network adjustments, making it ideal for small and medium-sized operators who may have limited resources and technical staff. By minimizing programming changes, our solution lowers the barrier to entry for smaller operators, allowing them to implement advanced network functionalities without requiring extensive technical expertise. In addition, the modular design allows for targeted upgrades and rapid rollbacks if required, giving small and medium-sized operators greater control and flexibility in managing their network infrastructure without substantial investment. Furthermore, the open-source code of this SDN solution allows network designers and researchers to easily manage XGS-PON infrastructures, and test new SDN functionalities and its integration with other infrastructures. However, the SDN agent that we previously presented in [30] only supports Internet (data) services. Nowadays, there is a trend towards

OTT (over-the-top) services, such as streaming TV services (Netflix, YouTube, Prime Video, or HBO), and thus, the proposal in [30] is a valid management solution for those OTT services via Internet data. Nevertheless, OTT services have many limitations compared to traditional video and voice services. For instance, compared to IPTV (Internet Protocol television) services, OTT services lack dedicated bandwidth, which can affect reception quality, while IPTV offers consistent quality thanks to its reserved bandwidth. Additionally, IPTV set-top boxes often feature extra functions like program recording and rewind. Security is another advantage of IPTV, with dedicated infrastructures and networks provided by operators, in contrast to the operator independence in OTT services. Consequently, many large and important operators currently offer IPTV services or even provide both types (IPTV and OTT). Similarly, when it comes to voice services (VoIP), having a dedicated bandwidth guarantees a high quality of service, unlike OTT-based solutions, which require bandwidth sharing between all services. Therefore, any pragmatic proposal on network management should be able to efficiently manage both types of services, OTT and traditional ones, and not just one of them, as many operators still offer those traditional services. As a consequence, in this paper we significantly extend the SDN agent by incorporating the capability to support these essential services for network operators or Internet service providers (ISPs). Therefore, the solution proposed in this paper fully supports various services for residential subscribers on XGS-PON architectures, not only data services, but also IPTV (multicast/video) and VoIP (voice), which are in fact services defined in the ITU-T G.9807 standard (XGS-PON) [2] and must be operational for use by network operators. Then, our proposal is fully aligned with the services and requirements set out in the XGS-PON standard itself. Furthermore, extending the agent to support and configure these services is not trivial, as it involves an in-depth analysis of the XGS-PON standard, in order to determine which ONU management and control interface (OMCI) entities of the ITU-T G.988 standard [31] are necessary to configure the ONUs, as well as to determine the order in which these entities must be created and their subsequent programming in the SDN agent. In addition, on the OLT side, chipset API functions must also be integrated to support and configure these services, which is a more complex procedure than for Internet data services. In summary, this paper presents a significant extension of the base model presented in [30] allowing the integration of VoIP and video services, which are key services for network operators and ISPs. In addition, we have conducted an experimental validation of the extended SDN agent within an XGS-PON, proving its effectiveness in the integration of those two services. Moreover, we have made the source code of the SDN agent open and available on GitHub, so that it can be easily used by network designers and researchers.

This paper is structured as follows. First, Section 2 reviews the fundamentals of the OpenFlow-based SDN agent for XGS-PONs presented in [30]. Then, Section 3 proposes the set of extensions to be added so that the SDN agent supports voice and video services, and Section 4 validates the effectiveness of those extensions by reporting the results of an experimental study conducted in an XGS-PON testbed. Finally, Section 5 summarizes the main conclusions.

2. REVIEW OF THE OPENFLOW-BASED SDN AGENT FOR XGS-PONS

The SDN proposal presented in [30] and summarized in this section allows for the configuration of XGS-PON using an SDN controller (ONOS [32]) and an OpenFlow-based SDN agent. This section summarizes the layers and functionalities of the agent and how it interacts with ONOS and the XGS-PON OLT's API to configure services.

A. Global Description of the OpenFlow Agent

Figure 1 illustrates a network scenario featuring the proposed OpenFlow-based SDN agent, enabling ONOS to interact simultaneously with multiple XGS-PONs through the same SDN agent. In essence, this agent translates commands originating from the SDN controller via OpenFlow into commands that can be interpreted by the API of the OLT chipset. These commands are transmitted to the OLT through a management interface port, isolated from the PON traffic, so as not to be affected by network congestion. The OpenFlow agent has been developed using the Python-OpenFlow Library [33], and is fully compatible with versions 1.3/1.0 of the OpenFlow standard, requiring no additional extensions to the OpenFlow standard itself. More specifically, the OpenFlow-based SDN agent intercepts messages sent by ONOS, which contain OpenFlow commands. These commands are then used to configure the OLT and ONUs connected to the XGS-PON, as well as to manage end-user services and profiles through the OLT. Consequently, the agent comprises three distinct layers, as illustrated in Fig. 1:

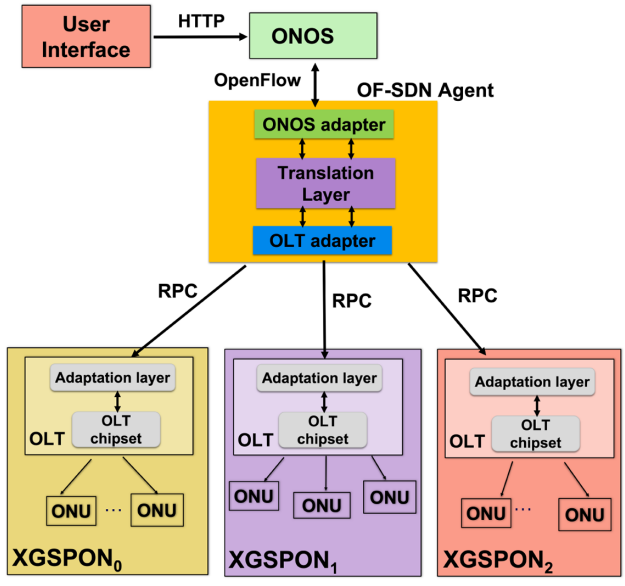

Fig. 1. Global design of the OpenFlow agent to manage XGS-PONs (Reprinted with permission from [30] © Optica Publishing Group).

- *ONOS adapter layer*: It interfaces with ONOS, capturing messages from the controller and responding to them as if the SDN agent were an OpenFlow virtual switch [34].
- *Translation layer*: It is a bridge between the other two layers, responsible for translating ONOS instructions into XGS-PON configurations.
- *OLT adapter layer*: It manages and configures XGS-PON OLTs in their native language, specifically using the API provided by the OLT chipset vendor. The approach employs RPC (remote procedure call) [35] connections to facilitate interaction between the OpenFlow agent and the OLT chipset's API. Subsequently, the OLT then configures the devices and services (e.g., Internet) subscribed to the users.

Furthermore, a menu-driven user interface has been developed, which simplifies the creation of ONOS API instances for configuring PON parameters. Thus, the key features of the whole system include the following:

- *Scalability and flexibility*: The system is structured into well-defined layers, allowing for the easy integration of new SDN functionalities into one specific layer without affecting the others. This can be achieved by defining the flow or schema in the ONOS adapter and generating the service in the translation layer. Consequently, new functionalities can be integrated with little change to the OLT adapter layer. In addition, all SDN configurations on the OLT are done through a port that is physically separated from network traffic, so the functionality of the SDN proposal and its scalability is not affected by network congestion, as this separation ensures that OLT network configurations will reach the OLT through a physically separated port.
- *Interoperability*: The solution acts as an intermediary element between an SDN controller and the XGS-PON equipment. This design enables a single SDN controller to manage PON equipment from various manufacturers using the same SDN agent. By leveraging the SDN agent code, RPC, and ONOS-interacting menu-driven interface, the proposal streamlines the management of any XGS-PON through a unified codebase. The adaptation layer within the OLT (illustrated in Fig. 1), responsible for direct interaction with the API of the OLT chipset vendor, is the only component that requires modification, as different OLTs may use distinct chipsets.

The OpenFlow-SDN agent defined in [30] only supports the configuration of Internet (data) services. This can be done with either single or double tags, as traffic on PONs must be tagged via VLANs (virtual local area networks) to differentiate services (voice, data, video). 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 tag (QinQ), which can be used to differentiate several operators arriving at the same PON. This second tag is called S-Tag (service tag).

B. Description of the OpenFlow Agent Layers

This section provides an overview of the main functionalities of the SDN layers of the OpenFlow agent. A more detailed analysis of the SDN agent can be found in [30].

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

The SDN agent emulates an OpenFlow switch [34], creating real OpenFlow messages to be exchanged with the ONOS controller. When an OpenFlow switch connects to an SDN controller, a set of OpenFlow messages is exchanged via a socket, namely, the following:

- **Connection Establishment**: When an OpenFlow connection is first established, each side of the connection immediately sends OFPT_HELLO messages.
- **Initial Communication**: ONOS periodically requests information from the OpenFlow agent. First, it requires information regarding the basic capabilities by means of OFPT_FEATURES_REQUEST and OFPT_FEATURES_REPLY messages. Then, information about the registered OpenFlow ports, the NNI (network-to-network interface) port of the OLT, and the UNI (user network interface) ports of the ONUs is required using OFPT_MULTIPART_REQUEST, OFMP_PORT_DESC, OFPT_MULTIPART_REPLY, and OFPT_PORT_STATUS messages.
- **Service Configuration**: Flows and meters are used to configure services. A flow is an element used to match and process packets, so it mainly contains match fields for matching packets, a priority for matching, and a set of instructions to apply [36]. In addition, meters permit QoS operations, such as rate-limiting, to be implemented, so they measure and control the rate of packets in flows. Thus, the messages exchanged between ONOS and the OpenFlow agent for configuring, updating, or deleting services are as follows:
  - **OFPT_FLOW_MOD**: The flows are used to configure a service for a specific user, UNI port of an ONU. Since services are filtered by VLANs, the criteria and instructions of the flows are based on this parameter.
  - **OFPT_METER_MOD**: The meters are used to associate a maximum bandwidth to a service. This maximum bandwidth is a guaranteed bandwidth plus excess bandwidth, only offered when the network has bandwidth available. Hence, meters measure the packet rate of a flow, so that when this rate is higher than a maximum bandwidth, packets are dropped.

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

In the translation layer, the OpenFlow-SDN agent translates the commands received from ONOS into commands interpretable by the OLT. In this way, this layer checks the received ONOS flows to assign them to an OpenFlow table [36] according to the rules described in [30]. First, the agent checks the matching fields and looks for the identifier of an incoming port, a UNI port of an ONU (traffic comes from the ONU, upstream traffic) or the NNI port of the OLT (traffic comes from outside the XGS-PON, downstream traffic). The SDN agent then looks for matching flows to configure the service.

For each new service, the SDN agent gathers flow parameters and stores them in a service list, ready to be transmitted

to the OLT through the OLT adapter layer. Each entry in the service list comprises the following fields: port (UNI port of the ONU), VLAN, and meter IDs. On the other hand, the translation layer also contains information about the XGS-PON and its devices (OLT, ONUs), so that it relates the instances and data coming from ONOS to the specific parameters of the PON, such as ONU data, service parameters [bandwidth, Alloc-ID, GEM (GPON encapsulation method) ports, ...], and OMCI configuration [31]. In addition to defining these service parameters, several QoS elements, including bandwidth policies, priorities, and weights, are configured at this layer for both traffic schedulers and queues. Consequently, all services are initially configured with "Best Effort" bandwidth policies, while priorities and weights are set to zero. Moreover, the current design of the OpenFlow SDN agent allocates a single GEM port for each service. Nevertheless, other configurations of these QoS parameters can be managed in this layer.

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

This layer connects the OpenFlow agent with the XGS-PON, to deploy/delete services in the ONUs using commands understandable by the OLT chipset. This communication is carried out through RPC connections, which requires an RPC server in the OLT and an RPC client within the OLT adapter layer. Since all OLTs will use the same RPC functions, an RPC server has to be integrated and deployed within each OLT with the same functions defined in the RPC client. These functions, defined in depth in [30], allow the network operator to configure the access network by acting directly on the OLT driver API. Although the OLT adapter layer uses these RPC functions to send the configurations to the OLT, the translation layer orchestrates the entire configuration process, due to its knowledge of the state of each PON device (OLT, ONUs). More details about these procedures can be found in [30].

3. EXTENSIONS TO THE OPENFLOW-BASED SDN AGENT TO SUPPORT VOICE AND VIDEO SERVICES

Once we have described the base OpenFlow-SDN agent presented in [30], we now extend that work by presenting a set of extensions to support VoIP and video (multicast) services, which are key services to be offered by ISPs and network operators. Integrating the configuration of voice and video services into the SDN agent has required a thorough analysis of the XGS-PON standard to identify the required OMCI entities to configure these services in the ONUs. This also involved establishing the correct order of creation and scheduling of these entities within the SDN agent. In addition, on the OLT side, API functions (in the chipset) must be integrated to complete the configuration of these services. Nevertheless, we have attempted to simplify service configuration, abstracting as much of the complexity as possible for the user through the chosen configuration tool, whether it be SADIS, ONOS, or the menu-driven user interface. Therefore, this section explains the required changes and additions in each layer of the OpenFlow-based SDN agent, and in the following section we experimentally validate these extensions. It is worth noting that the extended OpenFlow-SDN agent that we present next, with support for VoIP, video, and data services, has been made available on GitHub [37]. Specifically, the management system provided there consists of a set of four different Docker containers, one for ONOS, one for the SDN agent, another one for a menu-driven user interface to facilitate the configuration of the XGS-PON, and finally a VoIP server in another container.

A. Extensions to Support the VoIP Service

This section explains the extensions implemented within the OpenFlow agent to configure VoIP services.

1. ONOS Adapter Layer

This layer does not require any extensions or modifications, as the configuration of VoIP services only requires the creation and configuration of OFPT_FLOW_MOD and OFPT_METER_MOD messages with the same rules as described in [30].

2. Translation Layer

This layer stores services and device status information (OLTs, ONUs) to relate data coming from ONOS to specific OLT parameters. This layer requires some extensions, although it reuses some functionalities of the Internet (data) services. It needs to detect that it is a VoIP service in order to make the correct request to the OLT adapter layer. For this purpose, the following additional matching parameters (in the ONOS flows) have been defined in this layer:

- **IP (Internet protocol):** It matches the protocol ID for UDP (user datagram protocol) traffic, used for SIP (session initiation protocol) VoIP messages.
- **UDP destination:** This is the upstream destination UDP port, set to 5060, the port for SIP connections to a VoIP server.
- **UDP source:** This is the downstream source UDP port, which must be set to 5060.

However, there are other parameters that do not appear in the ONOS flows, but must be defined to configure this type of service, such as the VoIP extension, which is automatically selected by the SDN agent based on a configuration file (`openFlow_agent.config`) associated with the translation layer of the agent. In addition, the username will have the same name as the VoIP extension value, and the password will be a random set of four characters generated by the SDN agent and stored in another file (`subscribers.info`), also associated with the translation layer of the SDN agent. This file stores all parameters in JSON (JavaScript Object Notation) format, including the ONU ID, the PON port, and the list of the configured POTS (plain old telephone service) ports in the following format: "<ID_POTS>": [<username/extension>, "<password>"].

3. OLT Adapter Layer

First, with respect to the RPC server and the defined functions, the integration of VoIP services does not require the definition of new RPC functions. Moreover, the configuration of the VoIP service inside the OLT is identical to the configuration of an Internet service. Although this configuration is defined in depth in [30], in general terms, the SDN agent calls a function to configure bandwidth profiles. Thus, the most important bandwidth parameters in the upstream are Alloc-ID, committed information rate (CIR) (guaranteed bandwidth), and peak information rate (PIR) (maximum bandwidth), while in the downstream they are the T-CONTs (transmission containers), which correspond to priority queues in the ONUs.

However, the configuration of the VoIP service inside the ONU, which is done through OMCI entities in accordance with ITU-T G.988 [31], is considerably more complex than for Internet services, as it involves many more entities. Although the VoIP configuration is different from that of the Internet service, it is also necessary to configure the Alloc-ID, GEM ports, and VLAN tags, so the agent creates these same OMCI entities as for the Internet service [30]. In addition, for the specific configuration of the VoIP service, and following the ITU-T G.988 standard [31], the ONU first needs an IP address configured with an associated local UDP port. Then, the next step is the VoIP configuration, generating both voice and SIP configurations. Once this configuration is completed, the SDN agent sets up the VLAN configuration. Therefore, the agent has been extended by creating the following OMCI entities (defined in ITU-T G.988 [31]) in the OLT adapter layer:

IP address configuration

- **IP VoIP Config Data:** Used to select the desired signaling protocol (SIP) and configuration method (OMCI).
- **IP Host Config Data:** The ONU generates as many of these entities as VoIP services it could support. The SDN agent configures this entity with an IP address.
- **TCP (Transmission Control Protocol)/UDP Config Data:** It defines the local UDP port to be used for VoIP connections.
- **Extended VLAN Tagging Oper Config Data:** Associated with the IP Host Config Data entity. It is created but configured at the end of the OMCI process.

Voice and SIP configurations

- **Voice Service Profile and RTP (Real Time Transport Protocol) Profile Data:** It organizes the data describing the voice service functions and RTP functions.
- **VoIP Media Profile:** It joins up the voice service profile and RTP profile data entities and sets the required voice codecs.
- **TCP/UDP Config Data:** It defines the local UDP port to be used on VoIP connections.
- **SIP Agent Config Data:** It defines the VoIP SIP server and points to the IP configuration to open new connections.
- **SIP User Data:** It performs the SIP configuration of the user, i.e., the user's extension and the credentials to authenticate the user to the SIP VoIP server.

```
02-06-2023 13:23:09 | DEBUG:    ****VOICE SERVICE PROFILE ID 513:

02-06-2023 13:23:09 | DEBUG:    ****RTP PROFILE DATA ID 513:

02-06-2023 13:23:09 | DEBUG:    ****VOIP MEDIA PROFILE ID 513:
                                ********VoIP service profile pointer = 513
                                ********RTP profile pointer = 513

02-06-2023 13:23:10 | DEBUG:    ****NETWORK ADDRESS ID 513:
                                ********Username = 1111
                                ********Password = ooza
                                ********Address = 192.168.6.1

                                ****SIP AGENT CONFIG DATA ID 513:
                                ********Proxy server = 192.168.6.1
                                ********Outbound Proxy server = 192.168.6.1
                                ********Primary DNS = None
                                ********Secondary DNS = None
                                ********SIP username = 1111
                                ********SIP password = ooza
                                ********SIP registrar server = 192.168.6.1
                                ********TCP/UDP pointer = 36000

                                ****SIP USER DATA ID 513:
                                ********SIP Agent pointer = 513
                                ********User part AOR = 1111
                                ********Username and Pass pointer = 513
                                ********PPTP pointer = 513

02-06-2023 13:23:10 | DEBUG:    ****PPTP POTS UNI ID 513:
                                ********Admin state = 0
                                ********Transmission path = 0

02-06-2023 13:23:10 | DEBUG:    ****VOIP VOICE CTP ID 513:
                                ********User Protocol pointer = 513
                                ********PPTP pointer = 513
                                ********VoIP Media Profile pointer = 513
```

Fig. 2. Log status of the OLT with voice service OMCI configuration.

- **PPTP POTS UNI:** It is the termination point of a POTS port that identifies the physical port where the telephones will be connected.
- **VoIP Voice CTP:** It associates the PPTP POTS UNI with the VoIP configuration.

VLAN configuration

- **VLAN Tagging Filter Data:** To perform traffic filtering tasks by VLAN identifier.
- **Extended VLAN Tagging Oper Config Data:** It is configured with the VoIP service VLAN.

As an example, Fig. 2 shows the log of the SDN agent inside the OLT with the successful configuration of the OMCI entities of the voice service.

B. Extensions to Support the Multicast (Video) Service

This section explains the extensions implemented within the OpenFlow agent to support multicast services (video).

1. ONOS Adapter Layer

The support of multicast flows (video services) requires the configuration of a group element in addition to flows and meters (OFPT_FLOW_MOD, OFPT_METER_MOD). This group element must be associated with the flow and allows grouping several subscribers for the same flow. In this way, the messages exchanged between ONOS and the OpenFlow agent to configure this group element are as follows:

- **OFPT_GROUP_MOD**: It is used to join different users (UNI port of an ONU) on the same service (video service). These groups are associated with downstream flows so that the traffic of a flow will reach all users defined in the group

(members), for broadcasting purposes. This configuration is used for multicast services, as a single traffic flow is directed to several destination users (ONUs in our case). Thus, only those ONUs registered in a specific group will listen to the multicast service.

It is important to note that no extensions to ONOS or OpenFlow are proposed in the ONOS adapter layer, but rather existing and standardized methods are used but adapted to our use case, which gives potential to this SDN proposal as the same OpenFlow configurations can be used with other SDN controllers.

2. Translation Layer

In this layer the services and status information of the devices (OLTs, ONUs) are stored, so the parameters that differentiate the video service from the other integrated services must be added. In addition, this service has a parameter that makes it unique, as only multicast traffic uses groups in the configuration. Then, this layer must store the groups and all the members that compose it, as well as detect the flows that have an associated group in the matching fields, storing these types of services as multicast services.

3. OLT Adapter

As far as RPC functions are concerned, the integration of video services requires the definition of the following new functions:

- **PerformGroupOperation()**: It creates a new multicast group or modifies the members of one existing group.
- **DeleteGroup()**: It removes an existing multicast group.

Furthermore, at the OLT side, the configuration shown in Fig. 3 must be accomplished. First, the SDN agent generates the bandwidth profile (Downstream TrafficScheduler in Fig. 3) with the configuration of the guaranteed bandwidth (CIR) and the maximum bandwidth (PIR). Next, an empty Group is created, to which the Action is added, which is responsible for performing VLAN operations on the traffic. Finally, the SDN agent creates a Flow (Downstream Flow in Fig. 3) that points to the generated Group and defines the shape of the traffic through the Classifier fields. Then, all the members (specific ONUs) are added to the group, which will be composed of a PON interface and a GEM port, so that the ONUs added to the group can listen to the multicast service on that PON interface and that GEM port. Consequently, only ONUs added to the group will listen to the multicast service.

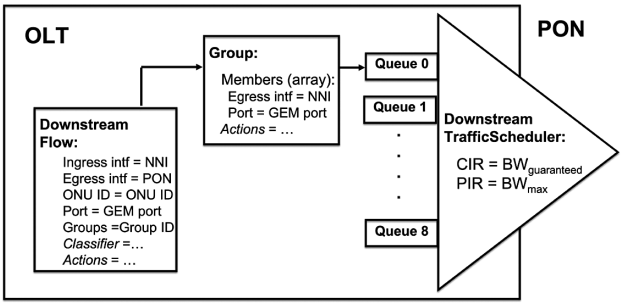

Fig. 3. Diagram of downstream multicast service configuration.

```
02-09-2023 08:42:33 | DEBUG:    ****GEM PORT NETWORK CTP ID 4094:
                                ********Port_id = 4094
                                ********Tcont_pointer = 0
                                ********direction = 3
                                ********Traffic_management_pointer_upstream = 0
                                ********Priority_queue_pointer_downstream = 0
                                ********encryption_key_ring = 0

02-09-2023 08:42:33 | DEBUG:    ****MULTICASTGEM INTERWORKING TP ID 4094:
                                ********Gem_port_network_ctp_pointer = 4094
                                ********Interworking_option = 1
                                ********Service_profile_pointer = 1
                                ********Interworking_tp_pointer = 0
                                ********gal_profile_pointer = 1

02-09-2023 08:42:33 | DEBUG:    ****MAC BRIDGE PORT CONFIGURATION DATA ID 4094:
                                ********Bridge_id_pointer = 1
                                ********Port_num = 254
                                ********Tp_type = 6
                                ********tp_pointer = 4094

02-09-2023 08:42:33 | DEBUG:    ****MULTICAST OPERATIONS PROFILE ID 4094:
                                ********Upstream Vlan = (8, 806)

02-09-2023 08:42:33 | DEBUG:    ****MULTICAST SUBSCRIBER CONFIG INFO ID 260:
                                ********OperationsProfileID = 4094
```

Fig. 4. Log status of the OLT with multicast service OMCI configuration.

For the OMCI configuration, each ONU associated with a multicast service must be configured with the GEM port associated with the group. Thus, the SDN agent starts configuring the GEM port, which is the same for all the ONUs (UNI ports) on the same PON interface. Since multicast services only have the downstream direction, all the entities related with the upstream channel are avoided. Thus, the entities to be configured are those defined for Internet traffic plus the following set of entities specific to the multicast service:

- **Multicast Operations Profile**: It defines the multicast access control configuration and IGMP (Internet group management protocol) parameters, such as the IGMP version and the upstream VLAN used for IGMP.
- **Multicast Subscriber Config Info**: It organizes data associated with multicast management at subscriber ports, relating Mac Bridge Port Configuration Data with the multicast operations profile generated.

As an example, Fig. 4 shows the status of the OLT log and the correct deployment of the previously explained OMCI entities for the multicast service configuration.

4. VALIDATION ANALYSIS OF THE EXTENSIONS ON AN XGS-PON TESTBED

In this section, we perform a set of experiments to validate the extensions of the SDN agent to support VoIP and multicast (video) services on a legacy XGS-PON.

A. Architecture of the XGS-PON Testbed

Figure 5 illustrates the experimental setup, consisting of an XGS-PON architecture, where the OpenFlow-SDN agent has been deployed in a Docker container, the ONOS controller (version 2.5.9) in another container, a menu-based user interface in a third container, and finally a VoIP server in another container. For video services we have used the VideoLAN Client (VLC) media player as the video server. On the other hand, all containers are hosted on a server that connects to the OLT through the management port using RPC. To facilitate the deployment process, a Docker-compose file has been used to enable the whole installation in a single action [37]. The

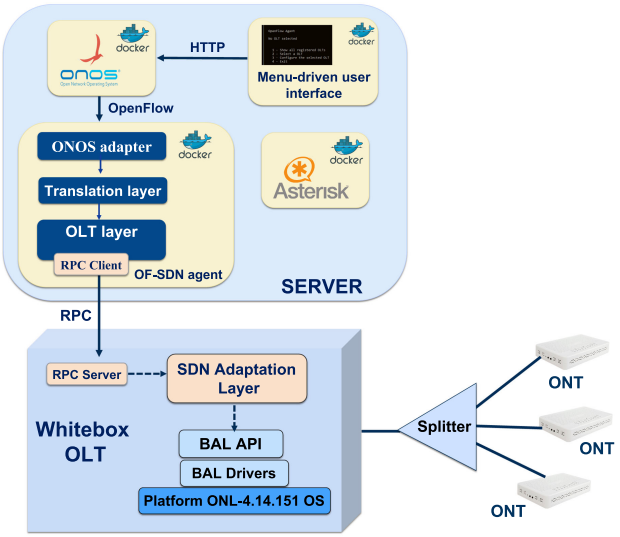

Fig. 5. Block diagram of the SDN-OpenFlow agent deployed on the XGS-PON testbed.

result is a complete and simultaneous deployment of the entire platform, encompassing the SDN agent, ONOS controller, user interface, and VoIP server.

The communication between these components is depicted in Fig. 5. Thus, HTTP connections between the user interface and the ONOS driver, OpenFlow connections between ONOS and the SDN agent, and finally RPC connections to the OLT are observed. In this context, the gRPC library [38] has been employed to link the RPC client developed in the OLT adapter with the RPC server in the OLT. Therefore, when executing any instruction from the RPC client to the RPC server, the adaptation layer will be instantiated within the RPC server, allowing the execution of the OLT chipset API methods. The SDN agent has been developed for the Broadcom Maple BCM68628 System-on-a-Chip, which is compatible with the BCM68620 series [39]. The BCM68620 series supports GPON, XGPON, XGS-PON, NGPON2, EPON, and 10G-EPON protocols, incorporating this support through a common API, enabling a unified software design.

Regarding the hardware, the XGS-PON testbed (Fig. 6) consists of an Optical Line Terminal ASXvOLT16 Whitebox of Edge core (16×10G XGS-PON/NG PON2 ports) [40]. To allow the connection of multiple ONUs to the ASXvOLT16 OLT, a passive 1:8 optical splitter has been deployed, so that the optical output of the OLT is connected directly to the input of the optical splitter, as shown in Fig. 6. In addition, a 15 dB attenuator has been incorporated at the input of the optical splitter to account for the full attenuation of the end-to-end distance in the XGS-PON, i.e., to emulate all fiber and additional losses. At the user end, several ONUs from different manufacturers have been installed, so each output of the optical splitter has been connected to a different ONU, as shown in Fig. 6. Finally, the models of deployed ONUs are summarized as follows:

- **One ONU of Azores (model WAG-8F2W6)**: Four gigabit Ethernet ports, two POTS ports, one RF port, $4 \times 4$802.11ax 5G, and 2×2 802.11ax 2.4G Wi-Fi [41].

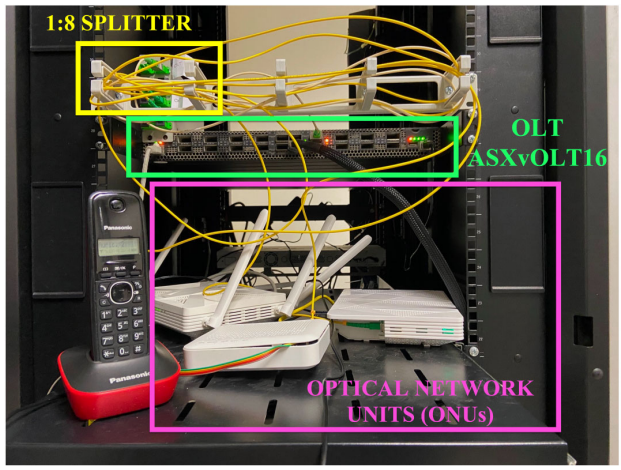

Fig. 6. Experimental deployment of the XGS-PON testbed.

- **One Broadcom ONU**: Four gigabit Ethernet ports, two POTS ports, Wi-Fi 802.11ax 2×2 at 2.4 GHz, and 802.11ax 4×4 (prototype).
- **One Bowie ONU (model WAG-D10T)**: One 10 gigabit Ethernet port and/or one 2.5G Ethernet port (prototype).

B. Configuration of the SIP Server: Asterisk

In order to test VoIP services, a VoIP server is needed. For that aim, Asterisk has been chosen [42]. Asterisk is a free and easy to use program that can be easily installed on Linux, deploying a SIP server on the network. The SIP server controls user registration and call setup, creating a point-to-point connection when the call is established.

For the case of study, Asterisk has been deployed on a Docker container, directly connected to the OLT NNI port; therefore, all the users on the network have access to it. Thus, all VoIP users of the network must authenticate against the Asterisk SIP server. Moreover, the username and password with which each user is configured must be registered as a new user in the Asterisk configuration files. For example, Figs. 7 and 8 show the configuration of the extension "1111," where the first image (Fig. 7) represents the username and password configuration, but also the type of extension configured as follows:

- *Type friend*: It denotes that the user is allowed to make and receive calls.

```
[uva-template](!)    ; UVA template
type=friend      ; Send and receive calls
host=dynamic         ; Any IP allowed
context=etsit        ; Context etsit

; Extension 1111 - uva
[1111](uva-template)
username=uva
secret=ooza
```

Fig. 7. Asterisk SIP user configuration.

```
[etsit]
exten => 1111,1,Dial(SIP/1111,30)
    same => n,Hangup()
```

Fig. 8. Asterisk SIP user dialplan configuration.

- *Host dynamic*: Used to filter the user connecting from a single IP address.
- *Context*: It provides the name of the network.

In addition, Fig. 8 shows the dialplan configuration for extension "1111," for which it has been configured in a simple call procedure, allowing testing of VoIP services.

C. VoIP Service Configuration and Validation

To configure a VoIP service on a specific POTS port of an ONU, the parameters to be configured in the user interface are for both downstream and upstream (symmetric): guaranteed bandwidth of 256 kbps, excess bandwidth of 256 kbps (a maximum of 512 kbps), and C-tag VLAN set to 60. Then, this configuration is encapsulated in JSON format and sent from this application to ONOS via an HTTP POST request. For this service, two flows are created in ONOS, one for downstream and another one for upstream, and only one meter is


```
{
    "priority": 50000,
    "timeout": 0,
    "isPermanent": true,
    "deviceId": "of:000004f8f80d6669",
    "tableId": 0,
    "treatment": {
        "instructions": [
            {
                "type": "L2MODIFICATION",
                "subtype": "VLAN_ID",
                "vlanId": 0
            },
            {
                "type": "OUTPUT",
                "port": "2013463041"
            },
            {
                "type": "METER",
                "meterId": "2"
            }
        ]
    },
    "selector": {
        "criteria": [
            {
                "type": "ETH_TYPE",
                "ethType": "0x0800"
            },
            {
                "type": "IP_PROTO",
                "protocol": 17
            },
            {
                "type": "IPV4_SRC",
                "ip": "192.168.6.1/32"
            },
            {
                "type": "IPV4_DST",
                "ip": "192.168.6.130/32"
            },
            {
                "type": "UDP_SRC",
                "udpPort": 5060
            },
            {
                "type": "UDP_DST",
                "udpPort": 36000
            },
            {
                "type": "IN_PORT",
                "port": 20
            },
            {
                "type": "VLAN_VID",
                "vlanId": 60
            }
        ]
    }
}
```


Fig. 9. JSON representation of the configured flow in the user application for the VoIP service.


```
{
  "deviceId": "of:000004f8f80d6669",
  "unit": "KB_PER_SEC",
  "burst": true,
  "bands": [
    {
      "type": "DROP",
      "rate": "256",
      "burstSize": "2004",
      "prec": "0"
    },
    {
      "type": "DROP",
      "rate": "512",
      "burstSize": "2004",
      "prec": "0"
    }
  ]
}
```


Fig. 10. JSON representation of the configured meter in the user application for the VoIP service.

associated with both flows. Figure 9 shows the JSON configuration of the downstream flow (the upstream is similar). This JSON has an *instructions* block and a *criteria* block; this last one defines the VoIP service. The SDN agent differentiates VoIP services by the transport layer protocol used (UDP), the port on the server side, which must match with the SIP protocol port (5060), and, additionally, it requires an IP configuration, both on the server and ONU sides, specifically:

- *IP PROTO*: The ID 17 indicates UDP protocol.
- *UDP SRC*: It defines the source UDP port and should be equal to the SIP server protocol (5060).
- *UDP DST*: It defines the destination UDP port, the port used by the ONU for VoIP connections, and the common range is 1025-65535.
- *IPv4 SRC*: It defines the SIP VoIP server address that the ONU should connect to for the SIP registration process, in this case 192.168.6.1/32.
- *IPv4 DST*: It defines the IP address the ONU will use on the VoIP connections (192.168.6.130/32).

For the upstream direction, the above parameters are reversed because the source is the destination in that case. The remaining parameters define the VLAN (VLAN_ID = 60), the UNI POTS port to deploy the service (IN_PORT = 2013463041), and finally the meter, which defines the service bandwidth. This meter is associated with the flow via the *instructions* field, as can be seen in Fig. 10, where the meter identifier "`meterId=2`" is shown. In addition, the meter is associated with two bands with a DROP parameter (Fig. 10), one of 256 kbps ("`unit": "Kb_PER_SEC`") for the guaranteed bandwidth and the other of 512 kbps for the maximum permitted bandwidth.

Figure 11 shows a Wireshark [43] capture associated with the communication between ONOS and the OpenFlow SDN agent. Specifically, it shows the meter creation message (OFPT_METER_MOD) by means of the command OFPMC_ADD. This screenshot shows the two bands (`Meter band`) of the meter with the corresponding rates of 256 and 512 kbps, as well as the identifier of the previous

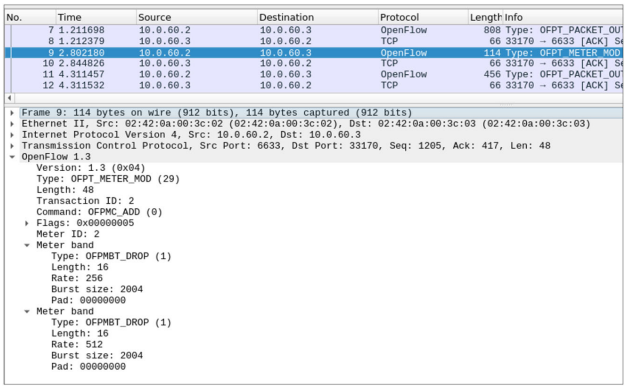

Fig. 11. Capture of the OpenFlow message OFPT_METER_MOD in Wireshark.

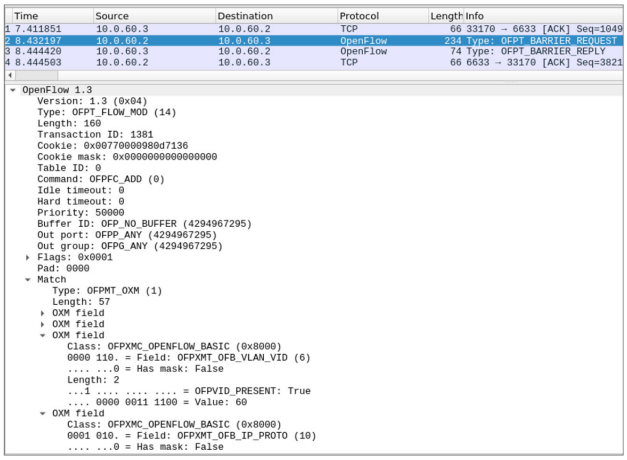

Fig. 12. Capture of the OpenFlow message OFPT_FLOW_MOD message in Wireshark.

meter (`Meter ID: 2`). Additionally, Fig. 12 shows the flow sent using the OFPT_FLOW_MOD message, which includes the command OFPFC_ADD to create the corresponding flow.

Once the service parameters are sent to ONOS, the status of the log inside the OLT corroborates that the service has been successfully installed. In this log (Fig. 13), XGS-PON parameters related to the traffic profiles are observed, including Alloc-IDs, GEM Ports, bandwidths, and VLANs, rather than OpenFlow meters and flows. In fact, the initial line displays the bandwidth profiles, featuring a guaranteed bandwidth of 256 kbps (CIR) and a maximum bandwidth of 512 kbps (PIR), along with the VLAN configuration in line 5 (`Received classifier with O_VID: 60`). Additionally, in the upstream direction, the log reveals the AllocID and the GEM port identified as 1024.

Once the service is configured, the ONU exchanges the registration SIP messages with the VoIP server. This process is validated capturing the messages with Wireshark (Fig. 14). The conversation is initiated by the ONU, sending a "SIP REGISTER" message to the VoIP server with the authentication parameters, such as `username:"1111."` When the user is successfully authenticated, the VoIP server sends an "OK" message to the user, which is acknowledged with another "OK" message, but in Fig. 14 the VoIP server first sends the "OPTIONS" message, announcing the allowed SIP options to the user, e.g., "INVITE," "CANCEL," "BYE," etc.

At this point, the user is ready to make or receive calls. A test call was made to validate the VoIP service, where the two ends are a physical phone connected to an ONU and a virtual phone to the other end outside the network through a free application called Zoiper [44]. The real-time streams of the call are shown in Fig. 15, where the lower table shows the call data, namely, the IP addresses of the users, the number of packets transmitted, and the sampling rate. In addition, the figure shows the real-time speech streams from both ends of the conversation, showing the pauses when the interlocutor stops speaking and the moments when the interlocutor speaks.

On the other hand, some QoS parameters, such as real-time bandwidth, jitter, and delay difference between packets, have been captured with Wireshark in both upstream and downstream directions, but only upstream data is shown to minimize the number of figures. The voice service tests were performed using the iperf tool, considering an iperf client transmitting at 1 Gbps connected to the UNI port of an ONU and an iperf server connected to the other end of the XGS-PON, i.e., to the OLT. Figure 16 shows the measured throughput (using Wireshark) of the VoIP service for the upstream channel. As shown in the figure, the throughput is 85 kbps, well below the maximum allowed bandwidth, due to the fact that the voice service consumes very little bandwidth. Furthermore, Fig. 17 represents the delta time of the conversation delay in the upstream. The delta time in a conversation is the elapsed time from the previous packet to the current packet in a conversation. It is observed that the delay between consecutive packets is fairly constant, so it can be stated that the quality is good. As for the delay difference (which is the difference between consecutive values of delta time), Fig. 18 shows that it is very small, around zero milliseconds. Finally, the jitter has been plotted in the upstream channel (Fig. 19), where it is observed that the values are very small, close to zero milliseconds, showing good quality of service.

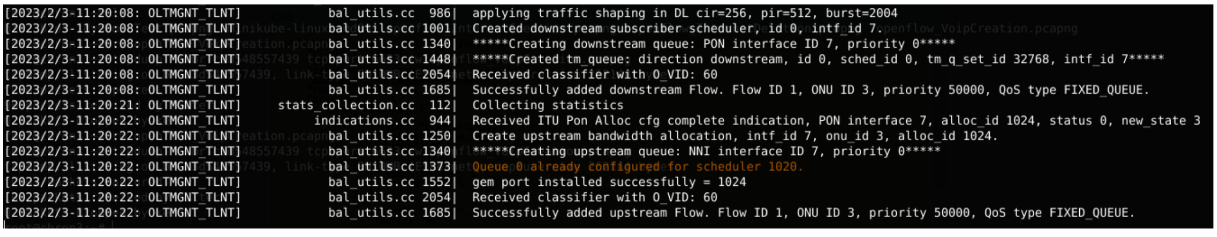

Fig. 13. Status of the log inside the OLT for the VoIP service.

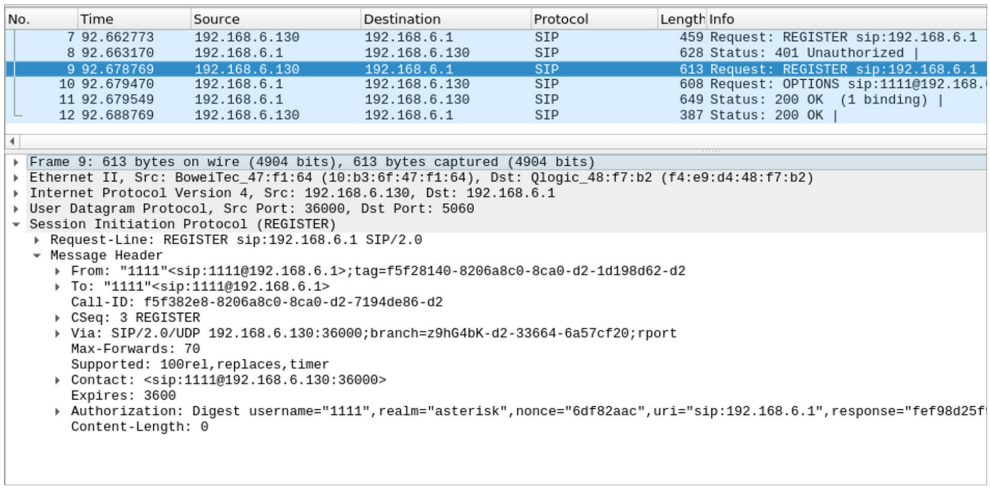

Fig. 14. Registration SIP messages captured in Wireshark.

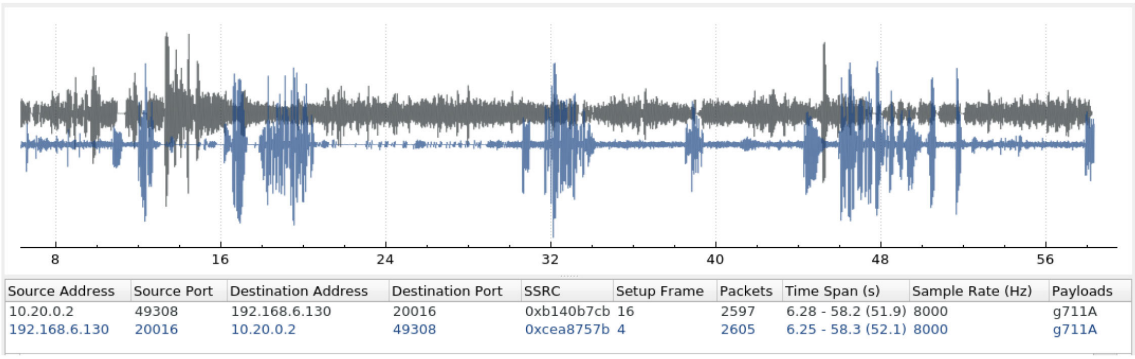

Fig. 15. Real-time voice transmissions from both ends of the conversation.

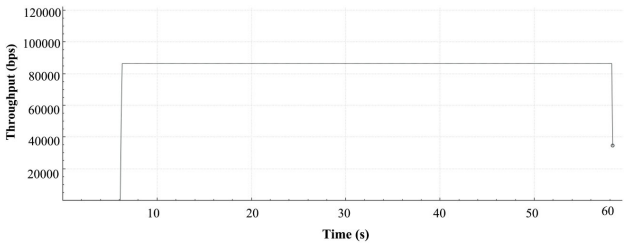

Fig. 16. Real-time throughput performance of the VoIP service measured with Wireshark (upstream).

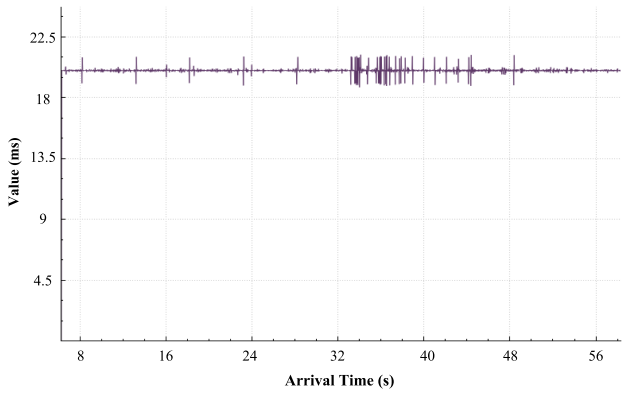

Fig. 17. Real-time evolution of the delta time measured with Wireshark (upstream).

D. Multicast (Video) Service Configuration and Validation

To configure a multicast service (video) on a specific UNI port of an ONU, the parameters were set employing the user interface. First, a multicast group was created to which all the ONUs over which the video service was deployed were added. Figure 20 shows the JSON sent to ONOS from the user interface with a UNI port of one specific ONU configured in the multicast group (`"groupID": "1"`). In this case, the multicast service has been associated with two different ONUs, identified by the port variable in the JSON message, namely, port numbers 1879245060 and 1879310596. In addition, the meter configuration shown in Fig. 21 limits both the guaranteed bandwidth and the maximum bandwidth of the service using two bands, in this case 400 Mbps and 500 Mbps, respectively, which means that the excess bandwidth is set at 100 Mbps.

Since multicast services only work downstream (from OLTs to ONUs), only flows will be created in that channel. Therefore, in a C-tag multicast service (single tag service), only one flow is created, as shown in Fig. 22. The structure of this flow is the same as that of the Internet services, except for the unicast "OUTPUT" field, which is replaced here by

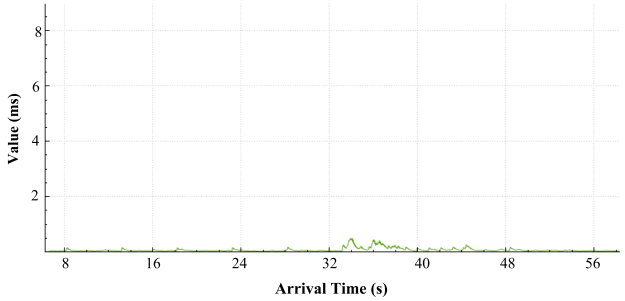

Fig. 18. Real-time evolution of the delay difference (delta difference) measured with Wireshark (upstream).

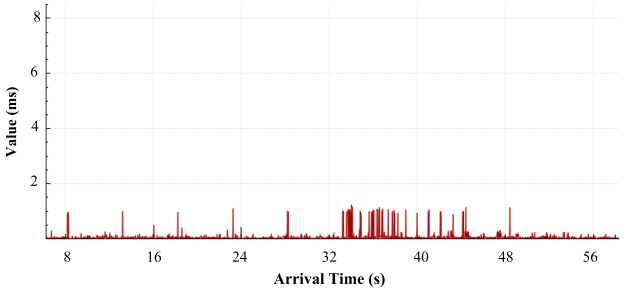

Fig. 19. Real-time evolution of VoIP service jitter measured with Wireshark (upstream).

```
"type": "ALL",
"appCookie": "0x1234abcd",
"groupId": "1",
"buckets": [
  {
    "treatment": {
      "instructions": [
        {
            "type": "OUTPUT",
            "port": "1879245060"
        },
        {
            "type": "OUTPUT",
            "port": "1879310596"
        }
      ]
    }
  }
]
```

Fig. 20. JSON that includes the group configuration for the multicast service.

```
{
  "deviceId": "of:000004f8f80d6669",
  "unit": "KB_PER_SEC",
  "burst": true,
  "bands": [
    {
      "type": "DROP",
      "rate": "400000",
      "burstSize": "2004",
      "prec": "0"
    },
    {
      "type": "DROP",
      "rate": "500000",
      "burstSize": "2004",
      "prec": "0"
    }
  ]
}
```

Fig. 21. JSON that includes the meter configuration for the multicast service.

```
{
    "priority": 50000,
    "timeout": 0,
    "isPermanent": true,
    "deviceId": "of:000004f8f80d6669",
    "tableId": 0,
    "treatment": {
        "instructions": [
            {
                "type": "L2MODIFICATION",
                "subtype": "VLAN_ID",
                "vlanId": 0
            },
            {
                "type": "GROUP",
                "groupId": "1"
            },
            {
                "type": "METER",
                "meterId": "1"
            }
        ]
    },
    "selector": {
        "criteria": [
            {
                "type": "IN_PORT",
                "port": 20
            },
            {
                "type": "VLAN_VID",
                "vlanId": 806
            }
        ]
    }
}
```

Fig. 22. JSON that includes the flow configuration for the multicast service.

the "GROUP" field, which points to the multicast group ("groupID": "1"). In addition, it can be seen that the meter corresponds to the previous meter, identified with ID = 1 ("meterID": "1"). Finally, it can be observed that the VLAN is set to 806.

Once ONOS receives this configuration in JSON format (group, meters, flows), it sends the same configuration to the SDN agent via OpenFlow messages. In Figs. 23 and 24, the communication between ONOS and the SDN agent is analyzed using the Wireshark tool. As an example, Figs. 23 and 24 show the creation of the group and the meter by means of the messages OFPT_GROUP_MOD and OFT_METER_MOD, respectively. Then, the downstream flow is sent through an OFT_FLOW_MOD service, where both the previously sent meter (ID = 1) and the group (ID = 1) are inserted. In addition, in the OFPT_GROUP_MOD, inside Actions, the ports that identify the ONUs can be seen, which are the same as in the JSON message in Fig. 20.

Once the SDN agent receives the OpenFlow configuration with the service parameters, it deploys this multicast service in the OLT and in the corresponding ONU. To verify that the multicast service has been correctly deployed in the OLT, Fig. 25 shows the status of the log inside the OLT. The OLT log corroborates the successful configuration of the service, creating a group (Group 1). Furthermore, within the log file, parameters related to the XGS-PON standard configuration can be observed, such as the GEM Port (4094), traffic schedulers

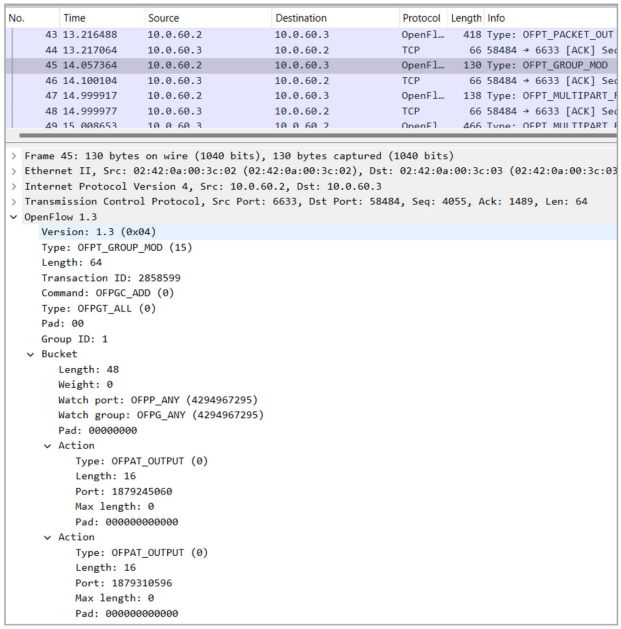

Fig. 23. OpenFlow messages between ONOS and the SDN agent to create a group.

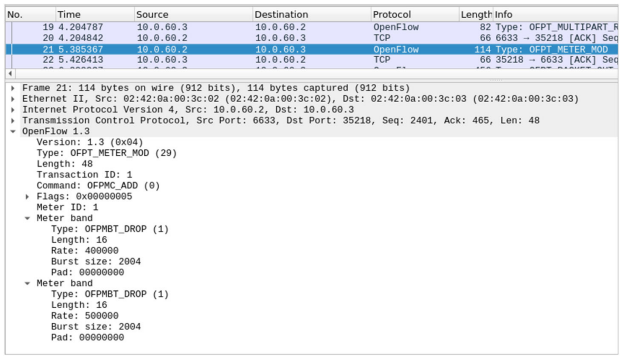

Fig. 24. OpenFlow messages between ONOS and the SDN agent to create a meter.

(`sched_id = 0`), traffic queues (`tm_queue = 0`), the VLAN (806), and guaranteed and maximum bandwidth levels (CIR, PIR).

The actual network bandwidth measurement tests are then carried out. First, multicast traffic is added to the network by configuring the multicast IP to 239.168.1.210, using the iperf tool. Then, capturing the traffic with Wireshark, the user starts listening to the multicast traffic by sending the corresponding IGMP join message with the selected multicast address (239.168.1.210), as shown in Fig. 26. This screenshot also shows the IGMP leave message, which appears when the user leaves the multicast group, i.e., when the user stops listening on the channel, in this case 30 s after starting to listen.

Furthermore, Figs. 27(a) and 27(b) show the real-time evolution of the bandwidth measured with Wireshark to analyze the performance of the multicast service in the downstream channel (the upstream channel is similar). The tests were 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 order to validate the functionality of the multicast service, the bandwidth performance of the two ONUs with which the video service has been associated has been analyzed. As can be seen in Figs. 27(a) and 27(b), the maximum bandwidth in both ONUs corresponds to the values configured, set to a maximum of 500 Mbps. In addition, a real video streaming test was also performed using the VLC media player. A laptop has been used to perform these tests (connected to the ONU), so that we can visualize the video content transmitted. Figure 28 shows a schematic depicting the test scenario. As can be observed in this schematic, the test is performed between a server connected to the OLT and a laptop connected to one ONU. Within this server, we deploy a VLC Media Player server (multicast server) and run an instance to transmit the video from the VLC Media Player server (with CLI commands) to the laptop connected to the ONU (with a VLC media player client). Furthermore, a picture of the real test scenario is shown in Fig. 29. In this picture it can be seen that the OLT is connected to the input of the 1:8 optical splitter and the different outputs are connected to the deployed ONUs. Thus, it can be observed that one ONU has the laptop connected to it through one of its Ethernet ports. The file to test the bandwidth of this service with the Wireshark tool (Fig. 30) is a free online video of 900 MB in MKV (MatrosKa Video) format with an approximate duration of 20 s. A server connected on the OLT side has been used to transmit the video. The VLC command used for the transmission is `vlc -vvv jellyfish-250-mbps-4k-uhd-h264.mkv -sout '#udpdst=239.168.1.210,port=1234'`, so that the multicast traffic is transmitted through the IP address 239.168.1.210 and UDP port 1234. These parameters (IP address and port) are the ones we have to configure in the VLC client to be able to receive the video signal. Besides, the real-time evolution of the bandwidth measured with Wireshark (Fig. 30) shows that the bandwidth

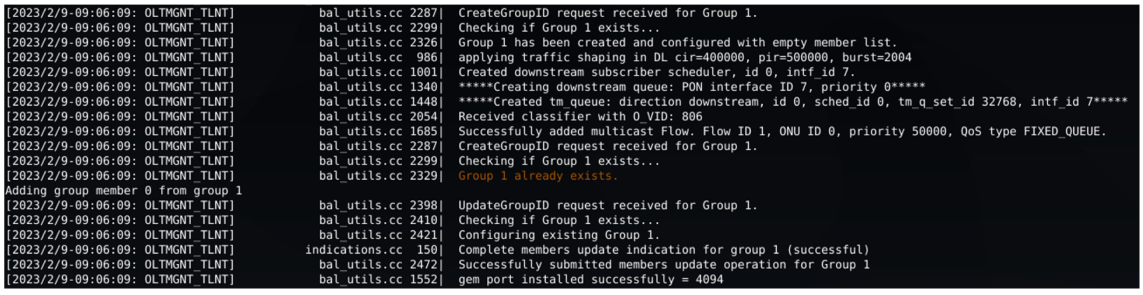

Fig. 25. Status of the log inside the OLT when configuring a multicast service.

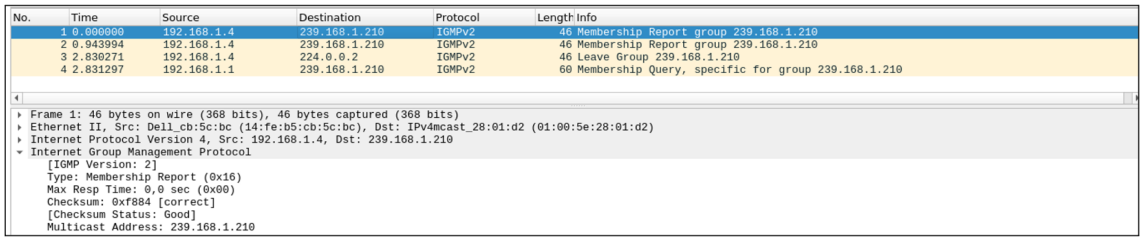

Fig. 26. Real-time bandwidth evolution for multicast service measured with Wireshark (downstream).

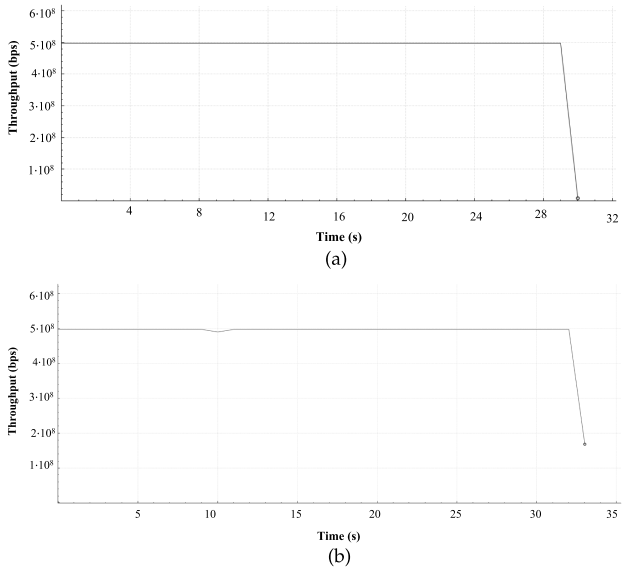

Fig. 27. Real-time bandwidth evolution for multicast service measured with Wireshark (downstream) in both ONUs associated with the multicast video: (a) ONU 1, (b) ONU 2.

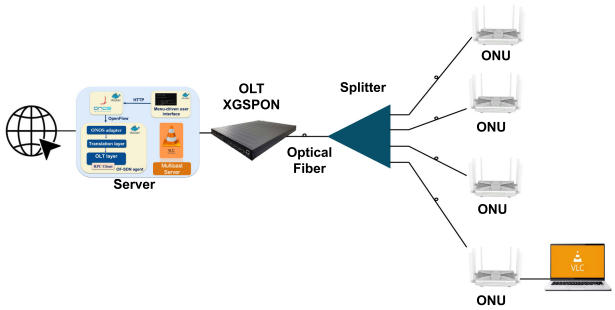

Fig. 28. Schematic depicting the test scenario.

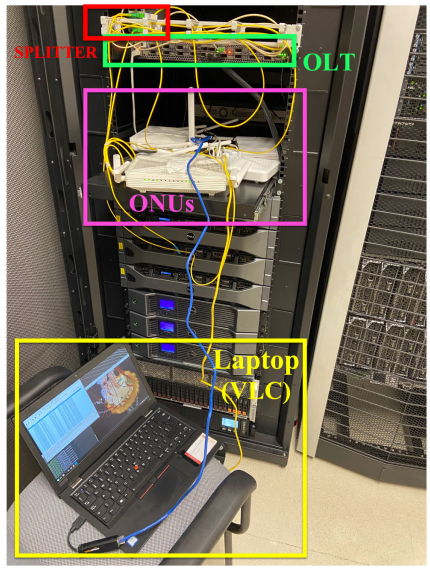

Fig. 29. Image of the real test scenario with the VLC media player.

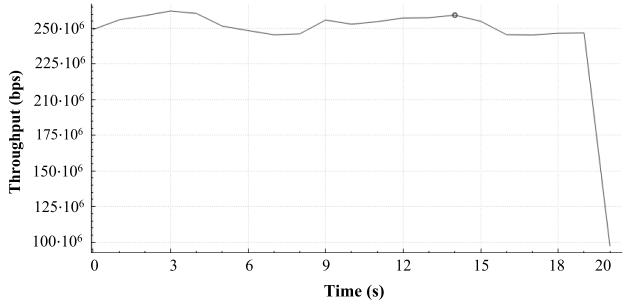

Fig. 30. Real-time bandwidth evolution of a real video stream with VLC measured with Wireshark.

during the 20 s duration is below the maximum (500 Mbps). In fact, the bandwidth corresponds to the transmission rate of this video, which is 250 Mbps.

Finally, to show the practicality of our SDN approach, we have evaluated the speed of configuration and deployment of a multicast service using our SDN agent. The test involves initiating a multicast service with an initial bandwidth (500 Mbps) and then modifying the bandwidth restriction approximately every 10 s, starting with 500 Mbps and moving to 600 Mbps, 200 Mbps, and ending with 500 Mbps, i.e., four bandwidth transitions. This assessment aims to ascertain whether our proposal can promptly and effectively meet these dynamic requirements. Figure 31 depicts the real-time performance of the dynamic video service, as measured by Wireshark. Notably, the duration between initiating the video service through the user interface (every 10 s) and its deployment is remarkably brief, approximately 1 s. This underlines the viability of our SDN agent in terms of speed and service deployment even under conditions of highly dynamic QoS requirements. It is important to note that these values are affordable for network operators since they deploy services only when users subscribe or contract new services. Therefore, these types of changes in a real network context are not very frequent and are not very

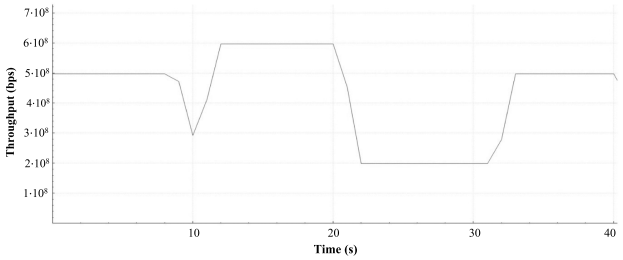

Fig. 31. Real-time bandwidth evolution for multicast service measured with Wireshark in a high dynamic network scenario.

dynamic, so the delay is not such a critical parameter, although our proposal proves to have very low values.

5. CONCLUSION

In this paper, we have extended an OpenFlow-based SDN agent for managing and configuring Internet services in XGS-PONs, by incorporating the capability to support VoIP and multicast (video) services. The agent communicates, on the one hand, with an ONOS controller via OpenFlow messages and, on the other hand, interacts directly with the OLT via the chipset manufacturer-specific APIs. Extending the agent to support VoIP and video services has required a detailed examination of the XGS-PON and ITU-T G.988 standards to determine the necessary OMCI entities to configure the ONUs, including the specific order of their creation and their scheduling in the SDN agent. Additionally, on the OLT side, the integration of chipset API functions has been essential to configure these services. In this way, the extensions required to support the VoIP service and the multicast service at the three layers of the SDN agent (ONOS adapter layer, translation layer, and OLT adapter layer) have been described. Furthermore, these extensions have been validated on an XGS-PON testbed, where VoIP and video services have been configured and demonstrated by means of iperf transmissions, a phone call, and a video transmission. The successful configuration of the network has been verified, and different performance metrics have been analyzed with the help of the Wireshark tool and log files. The experimental results confirm that the extended SDN agent allows for real-time configuration of voice and video services efficiently, meeting quality of service requirements. Moreover, these extensions have also demonstrated the adaptability of the base agent model presented in [30], thanks to its structure of clearly differentiated and independent layers.

In the future, we plan to employ the SDN-enabled XGS-PON testbed as the supporting wired infrastructure for use cases related to connected vehicles, edge computing, and computation offloading.

Funding. Agencia Estatal de Investigación (PID2020-112675RB-C42, RTC2019-007043-7); Consejería de Educación, Junta de Castilla y León (VA231P20); European Regional Development Fund (VA231P20).

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

Disclosures. The authors declare no conflicts of interest.

Data availability. The code to install and deploy the SDN agent with the extensions described in this manuscript (using Docker containers) is available on GitHub [37].

REFERENCES

1. "FTTH/B market forecasts 2023-2028" (2022) [accessed 10 March 2023], https://www.ftthcouncil.eu/committees/market-intelligence/1709/ftth-market-forecasts-2023-2028.
2. "10-gigabit-capable symmetric passive optical network (XGS-PON)," ITU-T Recommendation G.9807.1 (2020) [accessed 15 March 2023], https://www.itu.int/rec/T-REC-G.9807.1/en.
3. "XGS-PON moves center stage" (2022) [accessed 12 September 2023], https://www.lightreading.com/business-management/xgs-pon-moves-center-stage.
4. K. J. Kerpez, J. M. Cioffi, G. Ginis, *et al.*, "Software-defined access networks," IEEE Commun. Mag. **52**(9), 152–159 (2014).
5. "OpenFlow: Open Networking Foundation (ONF)" (2023) [accessed 10 May 2023], https://www.opennetworking.org/.
6. Network Configuration Working Group, "NETCONF configuration protocol" (2023) [accessed 15 February 2023], https://tools.ietf.org/wg/netconf.
7. Internet Engineering Task Force (IETF), "RESTCONF protocol" (2023) [accessed 15 February 2023], https://tools.ietf.org/html/rfc8040.
8. A. Thyagaturu, A. Mercia, M. McGarry, *et al.*, "Software defined optical networks (SDONs): a comprehensive survey," IEEE Commun. Surv. Tutorials **18**, 2738–2786 (2016).
9. F. Wang, B. Liu, L. Zhang, *et al.*, "Dynamic bandwidth allocation based on multiservice in software-defined wavelength-division multiplexing time-division multiplexing passive optical network," Opt. Eng. **56**, 036104 (2017).
10. C. Li, W. Guo, W. Wang, *et al.*, "Programmable bandwidth management in software-defined EPON architecture," Opt. Commun. **370**, 43–48 (2016).
11. H. Khalili, S. Sallent, J. Piney, *et al.*, "A proposal for an SDN-based SIEPON architecture," Opt. Commun. **403**, 9–21 (2017).
12. C. Centofanti, A. Marotta, D. Cassioli, *et al.*, "Slice management in SDN PON supporting low-latency services," in *European Conference on Optical Communication (ECOC)* (IEEE, 2022).
13. N. Merayo, D. de Pintos, J. Aguado, *et al.*, "An experimental OpenFlow proposal over legacy GPONs to allow real-time service reconfiguration policies," Appl. Sci. **11**, 903 (2021).
14. R. Flores, D. Fernández, N. Merayo, *et al.*, "NFV and SDN-based differentiated traffic treatment for residential networks," IEEE Access **8**, 34038–34055 (2020).
15. S. McGettrick, F. Slyne, N. Kitsuwan, *et al.*, "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).
16. B. Yan, J. Zhou, J. Wu, *et al.*, "SDN based energy management system for optical access network," in *9th International Conference on Communications and Networking in China* (IEEE, 2014), pp. 658–659.
17. M. Wang, G. Simon, I. Amigo, *et al.*, "SDN-based RAN protection solution for 5G, an experimental approach," in *International Conference on Optical Network Design and Modeling (ONDM)* (IEEE, 2021).
18. C. Qian, Y. Li, O. Zhang, *et al.*, "Staged priority-based dynamic bandwidth allocation in software-defined hybrid passive optical network," Opt. Eng. **57**, 126101 (2018).
19. L. Zhou, G. Peng, and N. Chand, "Demonstration of a novel software-defined flex PON," Photonic Netw. Commun. **29**, 282–290 (2015).
20. P. Parol and M. Pawlowski, "Future proof access networks for B2B applications," Informatica **38**, 193–204 (2014).
21. A. Amokrane, J. Hwang, J. Xiao, *et al.*, "Software defined enterprise passive optical network," in *10th International Conference on Network and Service Management (CNSM) and Workshop* (IEEE, 2014), pp. 406–411.

22. S. Lee, K. Li, and M. Wu, "Design and implementation of a GPON-based virtual OpenFlow-enabled SDN switch," J. Lightwave Technol. **34**, 2552–2561 (2016).
23. S. Lee, K. Li, W. Liu, *et al.*, "Embedding bandwidth-guaranteed network-based virtual Ethernet switches in SDN networks," J. Lightwave Technol. **35**, 5041–5055 (2017).
24. N. Merayo, D. de Pintos, J. Aguado, *et al.*, "Experimental validation of an SDN residential network management proposal over a GPON testbed," Opt. Switching Netw. **42**, 100631 (2021).
25. "VOLTHA: the open source project VOLTHA" (2023) [accessed 1 September 2023], https://opennetworking.org/voltha.
26. S. Das, "From CORD to SDN Enabled Broadband Access (SEBA) [Invited Tutorial]," J. Opt. Commun. Netw. **13**, A88–A99 (2021).
27. T. Suzuki, Y. Koyasako, K. Nishimoto, *et al.*, "Demonstration of IEEE PON abstraction for SDN enabled broadband access (SEBA)," J. Lightwave Technol. **39**, 6434–6442 (2021).
28. Y. Koyasako, T. Suzuki, T. Hatano, *et al.*, "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).
29. "Open Broadband Software" (2023) [accessed 1 April 2024], https://www.broadband-forum.org/open-broadband/open-broadband-software.
30. D. de Pintos, N. Merayo, C. Sangrador, *et al.*, "Software defined networking agent demonstration to enable configuration and management of XGS-PON architectures," J. Opt. Commun. Netw. **15**, 620–637 (2023).
31. "ONU management and control interface (OMCI) specification," ITU-T Recommendation G.988 amendment (2020), [accessed 3 September 2023], https://www.itu.int/rec/T-REC-G.988/.
32. "ONOS: the ONOS SDN controller" (2022) [accessed 13 October 2023], https://opennetworking.org/onos.
33. "Python-OpenFlow library, Kytos SDN" (2021) [accessed 20 September 2023], https://github.com/kytos/python-openflow.
34. "Open vSwitch: production quality, multilayer open virtual switch" (2020) [accessed 3 February 2023], https://www.openvswitch.org.
35. "Open systems interconnection-remote procedure call (RPC)" (2021) [accessed 5 October 2023], https://www.iso.org/standard/2229.html.
36. "OpenFlow Switch Specification, Version 1.3.1" (2020) [accessed 20 September 2023], https://opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf.
37. D. de Pintos and N. Merayo, "OpenFlow SDN agent in GPONs," GitHub (2023) [accessed 20 October 2023], https://github.com/GCOdeveloper/OpenFlow_Agent.
38. "gRPC protocol" (2020) [accessed 15 September 2023], https://grpc.io.
39. "Broadcom BCM68620 universal OLT PON MAC" (2020) [accessed 20 June 2023], https://www.broadcom.com/products/broadband/xpon/bcm68620.
40. "Edge core networks" (2020) [accessed 10 February 2023], https://www.edge-core.com.
41. "ONT Azores WAG-8F2W6" (2020) [accessed 10 February 2023], https://azoresnetworks.com/products/wag-8f2w6.
42. Asterisk (2020) [accessed 15 September 2023], https://www.asterisk.org.
43. Wireshark (2021) [accessed 20 October 2023], https://www.wireshark.org.
44. "Zoiper application" (2020) [accessed 10 February 2023], https://www.zoiper.com/.