Design and Implementation of a GPON-Based Virtual OpenFlow-Enabled SDN Switch

Steven S. W. Lee, *Member, IEEE*, Kuang-Yi Li, *Student Member, IEEE*, and Ming-Shu Wu

***Abstract*—Passive optical networks (PON) have become a promising solution for accessing networks because of the advantages they offer, such as high efficiency, security, and cost reduction. However, network management in PON is not yet automated and needs network operator intervention. In recent years, software-defined networking (SDN) has become an emerging technology. Through the separation of control and data plane in SDN switches, SDN provides dynamically fine-grained traffic control that enhances total network controllability and manageability. In this paper, we leverage the benefits of gigabit-capable passive optical network (GPON), while enhancing its capabilities on traffic management to the same level as an SDN switch. More specifically, we abstract the underlying physical GPON into an OpenFlow-enabled virtual SDN switch. The virtual switch can be used to connect multiple sites in widespread geographic locations. Similar to a real OpenFlow switch, a GPON virtual switch can be controlled by a standard OpenFlow controller. In our design, an embedded OpenFlow agent resides in the optical line termination (OLT) of the underlying GPON. The agent communicates with the external OpenFlow controller and simultaneously uses optical network unit management and control interface inside the OLT to manage ONUs. We created a prototype system based on a commodity GPON network. In the virtual switch, we implemented all the OpenFlow functions, including packet forwarding, bandwidth metering, statistical data collection, and status reporting. The experimental results show that the GPON virtual switch can correctly perform all the functions defined in the OpenFlow 1.3 specification. Its performance on flow entry modification time, dynamic bandwidth control, and switch status monitoring are comparable to the performance of a real OpenFlow switch.**

***Index Terms*—Gigabit-capable passive optical network, OpenFlow, software defined networking, virtual Switch.**

I. Introduction

SOFTWARE defined networking (SDN) is an emerging technology used to provide more flexible controls on networks by abstracting lower-level switch functionality [1], [2]. In SDN, the control plane and data plane are separate. The controller directly manages SDN aware devices via a well-defined interface and control protocol. The central control paradigm of SDN enables the controller to utilize network resources efficiently and facilitate QoS differentiation for services through network-wide optimization.

Manuscript received December 11, 2015; revised February 14, 2016 and March 7, 2016; accepted March 7, 2016. Date of publication March 8, 2016; date of current version April 11, 2016. This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant 103-2221-E-194-003-MY2.

The authors are with the Department of Communications Engineering, National Chung Cheng University, Min-Hsiung 62102, Taiwan (e-mail: ieeswl@ccu.edu.tw; nickykyli@gmail.com; f16cdf16cd@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JLT.2016.2540244

OpenFlow is the most popular technology for implementing SDN [3]. Many successful efforts have used OpenFlow to improve network performance [4]–[8]. The OpenFlow specification gives special consideration to L2/L3 switching and routing in the electronics domain [9]. Recently, much attention has focused on applying SDN to control optical networks [10]–[21]. To standardize software defined optical networking, ONF created the Optical Transport Working Group, whose goals are to study use cases [22], define a target reference architecture for controlling optical transport networks incorporating OpenFlow, and identify and create OpenFlow protocol extensions [23]. The requirements for optical transport controlled by OpenFlow/SDN can be found in [24].

Gigabit-capable passive optical network (GPON) is a cost effective access network technology that can support widespread users within a 20 km radius from the optical line termination (OLT) at the central office [25]. GPON is well known for its network management capability. In addition to providing physical OAM functions for critical signal timing, GPON also uses the ONU Management and Control Interface (OMCI) to perform statistical data collection, service provisioning, and fault management. GPON is able to accurately control bandwidth usage for each subscriber. The dynamic bandwidth allocation (DBA) algorithm in the OLT performs scheduling to determine the start time and duration for each ONU to access the network in the upstream direction. Any remaining bandwidth not consumed by an ONU is shared by the other ONUs in the network. Although GPON has strong management capabilities, its configuration and service provisioning are basically static. In other words, the bandwidth quota assigned to each ONU cannot be changed dynamically on demand.

Because GPON is not a switch, it is not included in the OpenFlow specification. In this work, we abstract the entire underlying GPON into an OpenFlow enabled virtual switch by implementing an embedded GPON agent that resides in the OLT. The agent itself interacts simultaneously with the external OpenFlow controller and the GPON OMCI module. For an OpenFlow controller, there is no difference between controlling a GPON virtual switch and a real switch. The benefit of the GPON virtual switch is that it is able to connect multiple sites spread across diverse locations—a capability that a real switch does not support.

A comprehensive survey for designing SDN network virtualization can be found in [26]. The development of network virtualization techniques allows network operators to generate a virtual network with a desired number of nodes and links. The highest levels of virtualization can even abstract the entire physical network topology as a single virtual node; however, those virtual networks are based on a real substrate consisting of real SDN switches. To the best of our knowledge, no research work thus far has focused on abstracting a GPON network into a virtual SDN switch.

A software defined passive optical network (PON) for intra-datacenter communications is proposed in [27]. The network is based on the TWDM-PON architecture [28]. A controller connects all the OLTs in the network, collects resource usage information, and monitors traffic conditions to determine bandwidth allocation. In [29], the authors propose a SDN-controlled GPON called OpenFlowPLUS for business to business applications. The controller in OpenFlowPLUS can control both OLTs and ONUs in the GPON network. OpenFlowPLUS introduces new functions to map flows to GEM ports and to map GEM ports to T-CONT instances, so it must modify the OLTs and ONUs to support those new functions. In [30], the specific requirements for an ISP GPON-based network are considered. It also proposed possibilities for integrating OpenFlow in GPON, in which an OpenFlow controller directly performs the DBA function. In addition, ONUs would have to be enhanced to provide OpenFlow-like matching and processing functions. A detailed implementation of the system is not addressed in that work.

To enable GPON to be controllable by a programmable controller, Amokrane *et al.* proposed an architecture called software-defined edge networks (SDEN) [31]. SDEN has two modules: one for the controller and the other for the OLT. The SDEN module (called SDEN agent) is appended to the SDN controller to call the module APIs located in the edge network. The SDEN agent translates high level flow control requests from a controller into a set of native PON configuration commands, and sends those commands to the OLT via a command line interface (CLI). In this way, the SDN controller can control and manage flows in GPON. In [32], the capacity steering problem in the SDEN is formulated as an ILP problem, the solution of which is used to determine DBA. In this work, we take these ideas a step further and transform the entire GPON system into a virtual switch. The virtual switch is completely compatible with the OpenFlow specification. Similar to controlling a real switch, an OpenFlow controller can communicate with the virtual switch using a standard OpenFlow protocol. In our system, no modifications to the controller are necessary. Because the controller does not differentiate between a real switch and a virtual switch, real switches and virtual switches can be intermixed to constitute a larger SDN network.

Fig. 1 depicts two architectures to realize a GPON-based SDN virtual switch. In Fig. 1(b), the virtual switch consists of only a single OLT. The access ports in the ONUs and the backhaul ports in the OLT become the ports of the virtual switch. The virtual switch can be used to interconnect multiple sites which are widely spread in different locations. Typically, the distance between the OLT and the ONU can be up to 20 km, and there is no other constraint for the distance between ONUs. Fig. 1(c) illustrates another architecture in which multiple GPONs are cascaded to form a large virtual switch by extending coverage and increasing the number of ports. Although there are only two GPON systems in the figure, even larger virtual switches can be constructed by connecting more GPONs in the network.

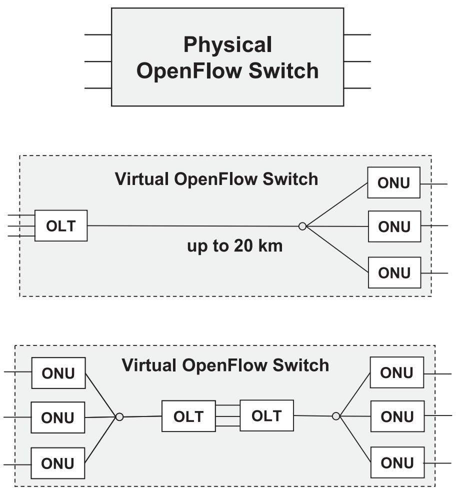


Fig. 1. Physical OpenFlow switch and two types of GPON virtual switches. (a) Physical switch. (b) A virtual switch with single GPON. (c) A virtual switch with multiple GPONs.

The virtual switch performs using standard OpenFlow protocols and is able to communicate with an outside OpenFlow controller. In other words, the OpenFlow controller can configure the flow table in the virtual switch. The switching functions of the virtual switch include packet forwarding, statistical data collection, and bandwidth control—the same functions that are provided by a physical switch. The only difference between a physical switch and a virtual switch is that the port used to connect the controller is restricted. Unlike a real physical switch, which can use any port to connect to the controller, due to limitations of the GPON system, only the port provided by the OLT can become the control port in a virtual switch.

Recently, AT&T and Open Networking Lab jointly proposed a new project named Central Office Re-Architected as Datacenter (CORD) [33]. In this project, a whole new architecture for the telecommunication central office is present. CORD leverages cloud and SDN technologies in the central office while using GPONs as low cost access networks. Since our GPON-based virtual switch is totally compatible with an OpenFlow switch, our virtual switch can help to realize CORD under a unified control of an OpenFlow protocol.

The remainder of this paper is organized as described below. In Section II, we present the architecture design for implementing the virtual switch system. In Section III, we describe our design and discuss some important implementation issues in detail. In Section IV, we demonstrate the experimental results. Finally, concluding remarks are made in Section V.

II. System Architecture

Fig. 2 depicts the system architecture of a GPON virtual switch. The design shown in Fig. 2(a) is based on a common commercially available GPON OLT and ONU system. Because the OLT does not understand the SDN protocol, an additional agent controller is responsible for communicating with the SDN controller on behalf of the OLT. The embedded agent controller processes the commands coming from the OpenFlow controller

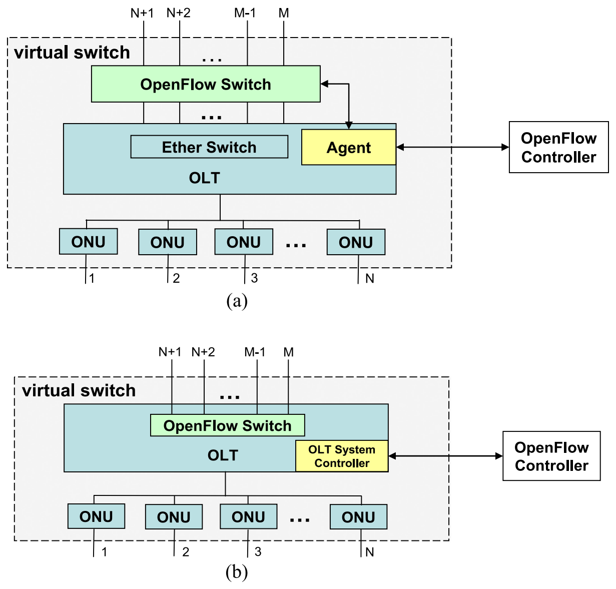


Fig. 2. Possible system architectures for a GPON virtual switch. (a) An implementation based on a current GPON system. (b) An implementation using the SDN switch fabric inside the OLT.

and then uses local commands to control the entire GPON through the OMCI. Most commodity OLTs include an Ethernet switch chip. However, because the Ethernet switch does not support functions defined by OpenFlow, the simplest way to create an OpenFlow virtual switch is to include an additional add-on OpenFlow switch in the system. The add-on OpenFlow switch in Fig. 2(a) provides functions for packet forwarding among ONUs. It also collects statistical data, attaches VLAN tags, and performs metering functions for bandwidth control. Please note that the Ethernet switch is not able to perform packet forwarding between two ONU ports because of the loop-free prevention property in an Ethernet switch. Fortunately, including an add-on OpenFlow switch inside the system can resolve that problem easily. In addition, the Ethernet switch inside the OLT usually does not perform bandwidth management. Although a GPON system is able to enhance bandwidth sharing through DBA in the upstream direction, it is not able to dynamically change the quota for each ONU. The add-on OpenFlow switch also resolves that problem.

Fig. 2(b) depicts a different architecture that replaces the embedded Ethernet switch chip in the OLT with an OpenFlow switch chip. The OLT controller directly controls the OpenFlow switch chip to perform OpenFlow functions. From the viewpoint of the controller, both architectures are the same. As more OpenFlow switches become available in the market, the cost of an OpenFlow switch chip will drop. We can foresee that the second architecture would be feasible in the future; however, in this work, we focused on the GPON virtual switch design based on a commodity system and used the first architecture in our implementation.

A. The Hardware Architecture

Fig. 3 depicts the hardware architecture of the OLT used in our implementation. It consists of a system controller (Marvell

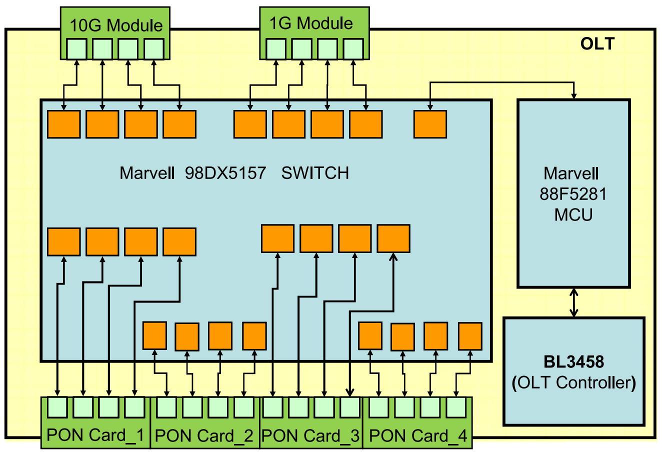


Fig. 3. Hardware architecture.

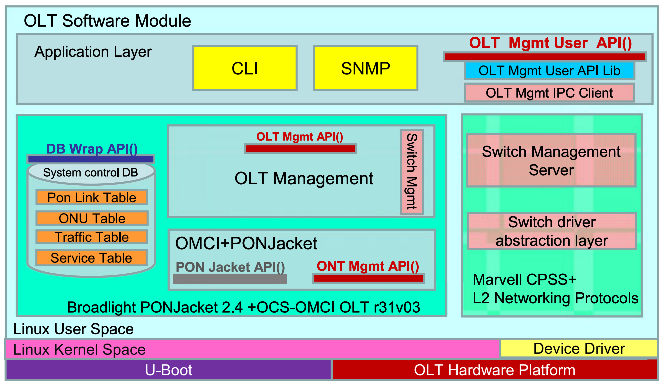


Fig. 4. Software architecture.

88F5281), an OLT controller (Broadlight BL3458), and an Ethernet switch (Marvell 98DX5157). The Marvell 88F5281 is an ARM-based controller running the Linux operating system. It hosts the OLT management program and the OpenFlow agent program. The BL3458 is an OLT controller responsible for controlling the GPON system. The Marvell 98DX5157 Ethernet switch is used to forward packets among different ports. Conventionally, a GPON system uses different VLAN IDs to differentiate traffic belonging to different ONUs. Because our goal is to use conventional GPON to build up a virtual switch, we follow VLAN tagging rules inside the GPON. We take full advantage of the multiple flow tables supported by the add-on SDN switch to delete unnecessary VLAN tags in the virtual switch. The detailed operations for VLAN processing are introduced in Section III.

B. The Software Architecture

Fig. 4 shows the software architecture. The program code executes in the user space of the Linux system. The Marvell CPSS+ module provides the driver for the layer 2 Ethernet switch. The Broadlight PONJACKET module includes an API that provides interfaces for GPON management. The most important set of functions inside the PONJACKET is the

OMCI. We have developed code in the application layer for an OpenFlow agent program that acts as a bridge between the OpenFlow controller and the GPON management module.

The OpenFlow agent executes in the application layer. The functions of the OpenFlow agent are divided into three parts as described below.

1) Interface With the OpenFlow Controller: The agent communicates with the external OpenFlow controller according to the standard OpenFlow 1.3 specification [9]. The OpenFlow controller can query system features, configure the flow tables, and request statistical data from the virtual switch. The virtual switch also responds to port status changes when any ONU ports are activated or deactivated.

2) Interface With OLT: In GPON, OLT uses physical layer OAM and OMCI to control ONUs. To facilitate programmatic control of the GPON, we prepared a set of API functions in the OLT which the agent uses to control and manage the GPON system.

3) Interface With the Add-On OpenFlow Switch: As shown in Fig. 2(a), the external OpenFlow controller does not directly control the add-on OpenFlow switch inside the system. In fact, the OpenFlow controller does not know about the existence of the add-on switch; instead, the add-on OpenFlow switch is controlled by the agent. The agent relays the OpenFlow messages coming from the external OpenFlow controller to the add-on OpenFlow switch, and vice versa. The technique has been used in many virtual network designs (e.g., [34], [35]). Using this technique, the external controller does not need to know the internal architecture of the virtual switch, which simplifies the configuration management of the external controller. Because the external controller does not directly interact with the add-on OpenFlow switch, the port IDs viewed from the OpenFlow controller are different from the port IDs of the GPON ONUs and the add-on OpenFlow switch. The agent performs port ID mapping between them for the external controller.

III. The Design and Implementation of the Agent

In this section, we introduce the implementation of the agent. All the design issues are explained in detail.

A. Initialization and Handshake

The agent must maintain connections with the OpenFlow controller and the add-on OpenFlow switch. When the agent is turned ON, in addition to activating the GPON system, its most important job is to set up a connection between the agent and the add-on OpenFlow switch. To the add-on OpenFlow switch, the agent is its controller, so the OpenFlow switch is responsible for reporting its switch features to the agent using standard OpenFlow protocol. Moreover, by detecting the status of the GPON system, the agent determines how many ONUs are available. After finishing the initialization processes for both GPON and the add-on OpenFlow switch, the agent follows the OpenFlow specification to set up a secure TCP connection to the external OpenFlow controller. The agent also must provide the mapping between ports in the virtual switch and the physical devices, including the ONU, OLT, and add-on OpenFlow switch. When the connection process between the agent and the external controller completes, the agent is ready to provide services for the external OpenFlow controller.

B. Symmetric Message Handling

According to the OpenFlow specification, messages can be classified into symmetric messages, asymmetric messages, and control-to-switch messages. We have implemented multiple handlers for processing all those messages in the virtual switch. The symmetric message handler maintains the connection between the controller and the switch. It performs functions to generate *hello* messages to the controller. It also receives *echo request* messages from the controller and generates *echo reply* messages in response.

C. Asymmetric Message Handling

Asymmetric messages are used by the virtual switch to actively inform the controller of particular events. When the virtual switch receives an unknown packet, the *Packet In* event generator delivers the packet to the controller. Similarly, a *Flow Removed* event generator notifies the controller when a flow entry is removed. Finally, a *Port Status change* event generator notifies the controller of any change in the ON/OFF state of a port.

D. Control-to-Switch Message Handling

The control-to-switch messages include commands for *Handshake Maintenance, Switch Configuration, Flow Entry Modification, Meter Modification, Switch Description, Individual Flow Statistics Data Collection, Port Statistics Data Collection, and Packet-Out*.

In OpenFlow, a controller can use the *packet out* message to deliver a packet to a port in the switch. In our system, the agent becomes a relay node that receives the desired packet-out packet sent from the OpenFlow controller to the add-on OpenFlow switch. Because the port ID used in the OpenFlow controller and the add-on OpenFlow switch are different, the agent modifies the port ID before delivering the received packet-out packet to the add-on OpenFlow switch.

OpenFlow uses a metering function to perform dynamic bandwidth control for each port in a switch. If a flow is associated with a meter value at a port, it is the maximum bandwidth the flow can use. Packets from traffic flows that use excessive bandwidth are dropped. Although it is a very useful tool for flow control, test results have shown that most of the commercial switches do not implement the function [36]. Due to the importance of the function, we have implemented it in our virtual switch. For the upstream direction in the GPON system, the metering function of the virtual switch is achieved by GPON DBA. For the downstream direction, the bandwidth control is realized in the add-on OpenFlow switch. In Section IV, we demonstrate the performance of the metering function in our prototype system.

The flow entry modification requires the virtual switch to identify the correct physical port inside the virtual switch. It also needs to address the VLAN tagging problem in the system.

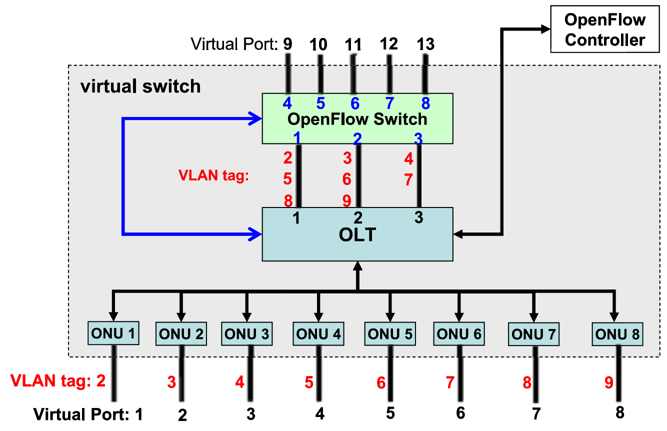


Fig. 5. VLAN tag mapping in the system.

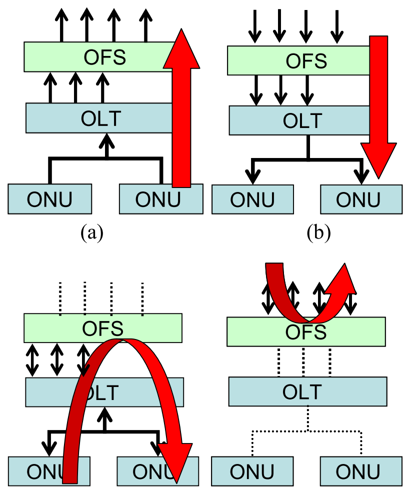


Fig. 6. VLAN tag mapping in the system. (a) Type 1: ONU port to OFS external port. (b) Type 2: OFS external port to ONU port. (c) Type 3: ONU port to ONU port. (d) Type 4: OFS external port to OFS external port.

We explain how to handle the flow entry modification problems in Section III-E.

The controller uses *port statistics* and *flow statistics* messages to query for statistical data on an individual port and/or flow. Because all the flows in the virtual switch go through the add-on OpenFlow switch, we can retrieve all the required data from the switch. Internal filters are applied to obtain the correct data. We present our implementation for statistical data collection in Section III-F.

E. VLAN Handling

Using VLAN to differentiate flows is a common approach for most commercial GPON systems. Similar to a typical port-based Ethernet, in the upstream direction, any incoming packet generated by an end user is tagged with the assigned VLAN ID of the ONU. In the downstream direction, the VLAN tag is attached by the gateway devices. The tag is removed by the ONU before forwarding the packet to the end user. However, for an OpenFlow switch, there is no VLAN tagging for packets unless the controller installs a *PUSH VLAN* action in a flow table. Our virtual switch has to handle the VLAN tagging problem to make a GPON virtual switch works similar to a real one. Fig. 5 illustrates an example in our system. As an ONU joins the system, it is assigned a unique VLAN ID. VLAN 1 is reserved in our system. Here we assign VLAN 2- VLAN 9 to the 8 ONUs shown in Fig. 5.

At the OLT, the output port used by the embedded Ethernet switch chip to forward an incoming ONU packet to the add-on OpenFlow switch is also determined by the VLAN ID. In Fig. 5, the VLAN IDs are equally assigned to all the ports connecting the OLT to the add-on OpenFlow switch for load balancing. The eight VLANs are assigned to the three ports such that each port handles at most three VLANs. The port-based VLAN system simplifies flow classification in the OLT. Note that in a commercial GPON system, the VLAN tags of the packets are not removed in the OLT so that the gateway at the central office can provide service differentiations based on the VLAN tags. Although VLAN tags help with service differentiations in GPON, they violate the operations of an OpenFlow switch; More specifically, the target of our design is to make a GPON network behaves like an OpenFlow-enabled switch. Therefore, the packets must not be modified outside of the installed flow-table rules. To this end, the additional VLAN tag generated in the GPON system must be removed before packets leave the virtual switch.

The OpenFlow switch inside the virtual switch handles the removal of the VLAN tags attached by the ONU. For example, when a packet enters the virtual switch headed from port 1 to port 4, a VLAN 2 tag value is attached at ONU 1. Because the upstream packets generated by ONU 1 are delivered to port 1 of the add-on OpenFlow switch, we install a flow entry in the switch to pop the VLAN 2 tag for each packet coming from port 1.

For a real OpenFlow switch, the controller can install an entry with a *PUSH VLAN* action in the flow table that adds a VLAN tag for a particular flow. This function is also provided by our virtual switch. In that case, our system must to handle two VLAN tags: one is attached by the GPON system using a default VLAN ID and the other is requested by the OpenFlow controller to push a specific VLAN ID for that flow. The virtual switch must perform both port mapping and VLAN tag removal to complete the operation correctly. To simplify the discussion, we can reclassify the ports in the virtual switch to ONU ports into three categories: OLT ports, OFS internal ports (the ports connecting the OLT and the add-on OpenFlow switch), and OFS external ports. As shown in Fig. 5, the ONU ports are labeled 1, 2, . . . , 8 and the OFS external ports are labeled 9, 10, . . . , 13. The ONU ports and the OFS external ports are the ports used to connect outside devices. In addition, there are three OLT ports and three OFS internal ports. Both the OLT ports and OFS internal ports are inside the virtual switch.

As shown in Fig. 6, we categorize each flow in the virtual switch into one of four types according to their incoming and outgoing ports. Each type of traffic requires different handling operations due to the VLAN tagging.

Type 1: ONU port to OFS external port. A Type 1 flow enters the virtual switch from an ONU port and leaves the virtual switch

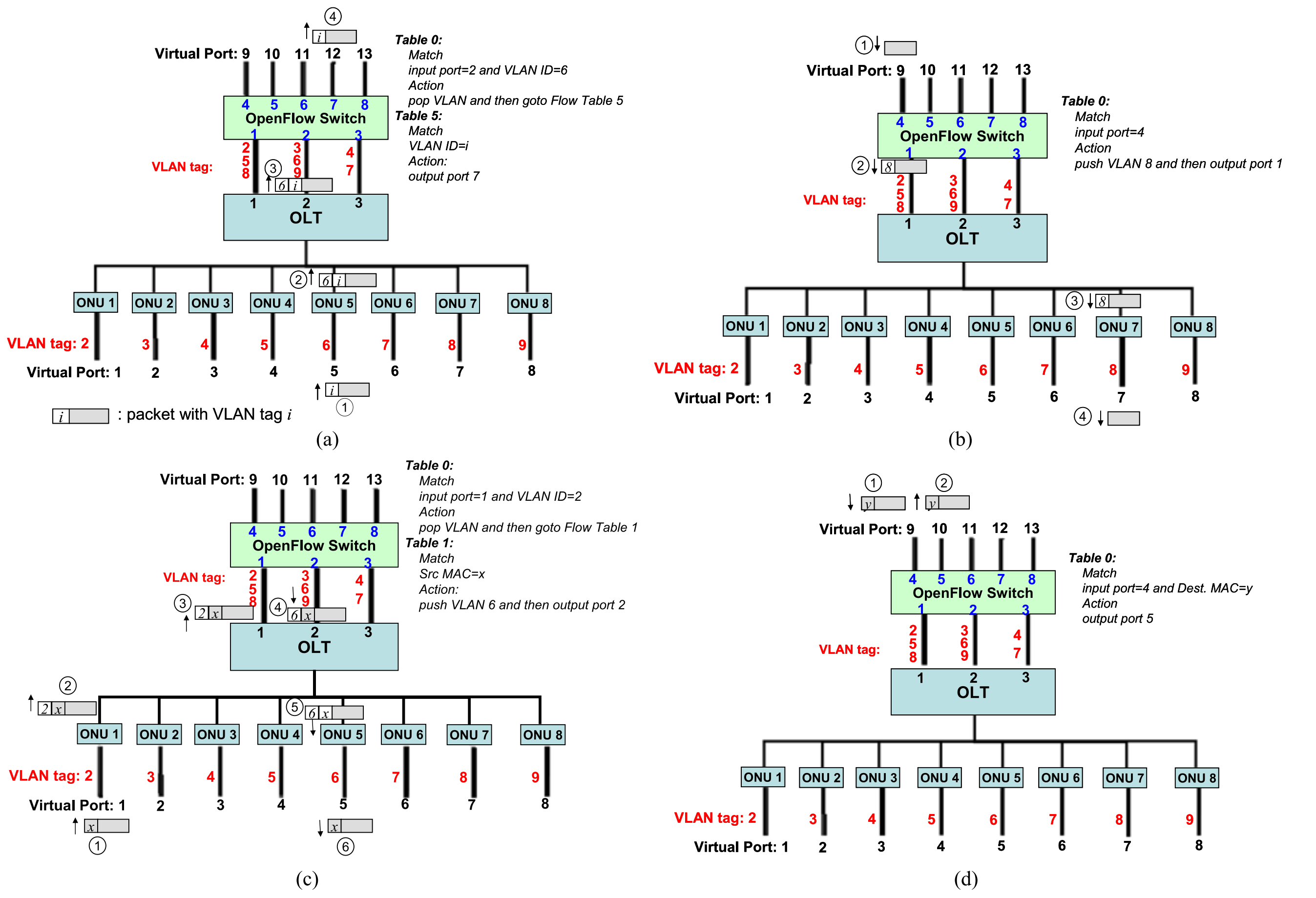

Virtual Port: 9 10 11 12 13
OpenFlow Switch
OLT
VLAN tag:
ONU 1 ONU 2 ONU 3 ONU 4 ONU 5 ONU 6 ONU 7 ONU 8
VLAN tag: 2 3 4 5 6 7 8 9
Virtual Port: 1 2 3 4 5 6 7 8
: packet with VLAN tag i
Table 0:
Match
input port=2 and VLAN ID=6
Action
pop VLAN and then goto Flow Table 5
Table 5:
Match
VLAN ID=i
Action:
output port 7
(a)
Table 0:
Match
input port=4
Action
push VLAN 8 and then output port 1
(b)
Table 0:
Match
input port=1 and VLAN ID=2
Action
pop VLAN and then goto Flow Table 1
Table 1:
Match
Src MAC=x
Action:
push VLAN 6 and then output port 2
(c)
Table 0:
Match
input port=4 and Dest. MAC=y
Action
output port 5
(d)

Fig. 7. Examples of VLAN tagging. (a) Type 1 traffic. (b) Type 2 traffic. (c) Type 3 traffic. (d) Type 4 traffic.

at an OFS external port. For this type of traffic, a VLAN tag is attached to the packet at the ONU port. In addition to performing the command requested by the OpenFlow controller, the virtual switch must also remove the VLAN tag before it leaves the virtual switch. However, it would be problematic to directly implement the required flow actions and default VLAN removal using a single flow table. For example, in Fig. 7(a), the controller would download the following command:

Match: "input port = 5" and "VLAN ID = I," Action: "output port 12."

The command requests that the switch forward all incoming packets tagged with VLAN tag *i* by the world outside the virtual switch to output port 12. Because port 5 is an ONU port, the switch will attach an additional VLAN ID 6 to the incoming packet, which means there are two stacked VLAN tags attached to the packet. All the incoming flows from port 5 of the virtual switch enter the add-on OpenFlow switch from its second port. Because there are now two VLAN tags that need to be examined, the command cannot be translated into a single flow entry for the SDN switch.

In our design, we take full advantage of the multiple flow tables feature of the add-on OpenFlow switch to simplify the implementation. When the OpenFlow controller installs a flow entry corresponding to Type 1 traffic, the agent automatically splits that entry into two for the add-on OpenFlow switch. The first entry is stored in the first flow table (Table 0) in which the default VLAN IDs added by the ONUs are examined. The action in this flow entry is to pop the default ONU VLAN tag on the packet and then go to the second flow table for further match-action processing. The ID of the second table is exactly the same as the input port ID. In the second flow table, the command is the same as the one sent by the external OpenFlow controller except for the removal of the matching input port. For the above example, the entries in both of the flow tables in the SDN switch become:

Table 0: Match "input port = 2" and "VLAN ID = 6," Action "pop VLAN" and then "goto Flow Table 5"

Table 5: Match "VLAN ID = I," Action: "output port 7."

Because a packet coming from port 5 of the virtual switch is tagged with VLAN ID 6, the packet will enter the add-on OpenFlow switch from port 2. Thus, in Table 0, we have to match input port 2. Because the incoming port (port 5) is implicitly identified by the ID of the second flow table, the missing input port information does not create any ambiguity.

Here is another example to help clarify the operations of the virtual switch. The command sent by the controller is:

Match: "input port = 6" and "Destination IP = y," Action: "output port 10."

Unlike the previous example, in this example, the packet that enters the ONU port carries no VLAN tag. This command is transformed to use two flow tables:

Table 0: Match "input port = 3" and "VLAN ID = 7," Action "pop VLAN" and then "goto Flow Table 6"

Table 6: Match "Destination IP = y," Action: "output port 5."

Type 2: OFS external port to ONU port. The second type of traffic starts from an external port of the SDN switch and goes to an ONU port. For a Type 2 packet, the add-on OpenFlow switch has to push a VLAN tag with an ID assigned to the destination ONU. As shown in Fig. 7(b), because input port 9 of the virtual switch corresponds to port 4 of the OpenFlow switch and output port 7 of the virtual switch is the ONU that is using default VLAN 8, the OpenFlow command: *Match: "input port 9," Action: output port 7"* is transformed to a single flow entry in Table 0 of the add-on OpenFlow switch below, using the following command:

Table 0: Match "input port 4," Action "push VLAN 8" and then "output port 1."

Type 3: ONU port to ONU port. The operations for handling type 3 traffic are the most complex because the old VLAN attached by the ONU has to be removed and a new VLAN tag that corresponds to the destination ONU has to be attached. Similar to Type 1 traffic, we use two flow tables to implement the function in the OpenFlow switch. The example in Fig. 7(c) shows the use of the two flow tables.

Match: "input port 1" and "Source MAC = x," Action: "output port 5" is transformed to:

Table 0: Match "input port 1" and "VLAN ID = 2," Action "pop VLAN" and then "goto Flow Table 1."

Table 1: Match "Source MAC = x," Action "push VLAN = 6" and then "output port 2."

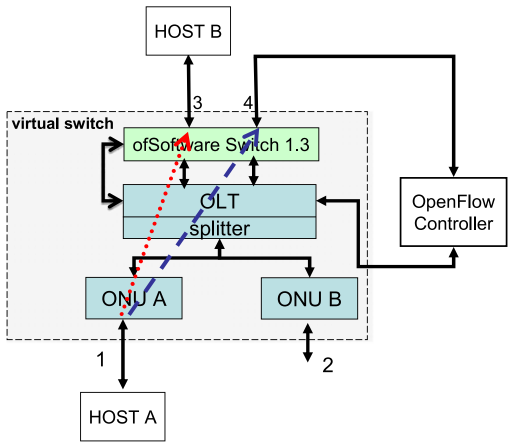
Type 4: OFS external port to OFS external port. For Type 4 traffic, the add-on OpenFlow switch is the only device involved in the process. There are no VLAN tags that need to be handled. We only need to map the port ID specified in the OpenFlow command to the correct port ID used in the add-on switch. The example in Fig. 7(d) shows the command sequences.

Match: "input port 9" and "Destination MAC = y", Action: "output port 10" is transformed to:

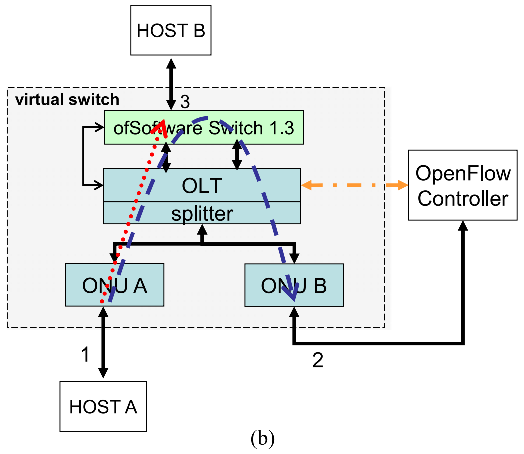
Table 0: Match "input port 4 and "Destination MAC = y", Action "output port 5"

F. Statistics Data Collection

The controller uses *Port Statistics* and *Flow Statistics* messages to query statistical data for a port and a flow, respectively. Because all the flows in the virtual switch go through the add-on OpenFlow switch, we can obtain all the required data from that switch. However, correct filtering rules must be applied. More specifically, because a port of the add-on OpenFlow switch may receive flows coming from multiple sourced ONUs, we have to maintain a data structure to differentiate the flow sources. Because the OpenFlow controller queries a specific flow, the agent transforms the command into a valid query for the add-on OpenFlow switch. The statistical data obtained from the add-on OpenFlow switch is used to derive the final result. For example, in Fig. 5, because port 1 of the OpenFlow switch is shared by ONU 1, ONU 4, and ONU 7, the query for the total byte count of packets entering port 4 of the virtual switch can be obtained by summing all the flows entering port 1 of the add-on OpenFlow switch with VLAN 5 as the filter.



(a)



(b)

Fig. 8. Response time to modify a flow entry. (a) Port 1→Port 3 changed to Port 1→Port 4. (b) Port 1→Port 3 changed to Port 1→Port 2.

IV. Experimental Results

We performed several experiments to evaluate the performance of the proposed GPON based virtual switch. The test system used a Broadlight BL3458 OLT with two TECOM GP6100 ONUs. The add-on OpenFlow switch was a PC running of-Softswitch [37]. The external OpenFlow controller was a PC running the NOX control program [38].

A. Switching Time

First, we evaluated the response time to modify a flow entry in the virtual switch. We conducted two experiments to examine the time required to change the output ports for a particular flow. In the first experiment, shown in Fig. 8(a), the source port of the flow was an ONU port and the output ports were two OFS external ports. In our experiments, the PC running the controller program had two NIC cards installed. The first card was used

TABLE I
RESPONSE TIME FOR FLOW ENTRY MODIFICATION OF THE VIRTUAL SWITCH

Flow Entry Modification (ms)	Average	Maximum	Minimum
ONU port to SDN_external port (see Fig. 8(a))	0.642	1.175	0.265
ONU port to ONU port (see Fig. 8(b))	0.818	1.492	0.474
OFsoftswitch	0.194	0.346	0.115
EdgeCore AS4600	1.717	1.840	1.537
HP5900	211.807	358.709	34.618

to send OpenFlow commands to the virtual switch, and the second card was used to receive the packets after the port change command was successfully executed. During the experiment, Host A constantly generates packets destined for Host B every 10 μs. Initially, the GPON virtual switch forwarded a flow with packets coming from port 1 to port 3. The OpenFlow controller issued a flow modification command to redirect the flow to port 4. Packet losses occurred during the process of reconfiguration in the virtual switch, but after the new flow entry was successfully installed, the PC running the controller could receive packets from the second NIC card. By comparing the gap between the time that the controller sent out the flow modification command and the time that the controller received the first incoming packet from the second NIC card, we could determine the response time required to process the flow modification.

In the second experiment, illustrated in Fig. 8(b), we examined the response time required to change the output port from an OFS external port to an ONU port. The configuration of the external switch was the same as it was in the first experiment. In the beginning of the second experiment, the flow generated from Host A was sent to Host B through port 3. Then, the controller modified the flow entry to redirect the output of the flow to port 2. Because port 2 is an ONU port, the flow went through the GPON in both the upstream and downstream directions. Again, the switching time was the time gap between when the controller sent the flow modification command to the virtual switch and when the controller received the first packet from Host A through its second NIC card.

The experimental results are shown in Table I. For each setting, we performed the experiment 30 times to obtain the maximum, minimum, and mean average switching time. For the flow in the upstream direction, i.e., the case shown in Fig. 8(a), the average switching time was 0.64 ms. For the flow using both upstream and downstream directions in the second experiment (see Fig. 8(b)), the average switching time was 0.82 ms. We also measured the response time using three real switches: an ofSoftswitch [37], an EdgeCore switch, and the HP 5900 switch. On average, to complete a flow modification command, the HP 5900 switch requires 211 ms, the EdgeCore AS4600 switch requires 1.71 ms, and the ofSoftswitch, the fastest of the three, requires only 0.194 ms. These experimental results show that the switching time for modifying a flow entry in the virtual switch is shorter than two commercially available OpenFlow switches.

B. Meter Function

In this set of experiments, we used a metering function to perform dynamic bandwidth control for each port in the virtual

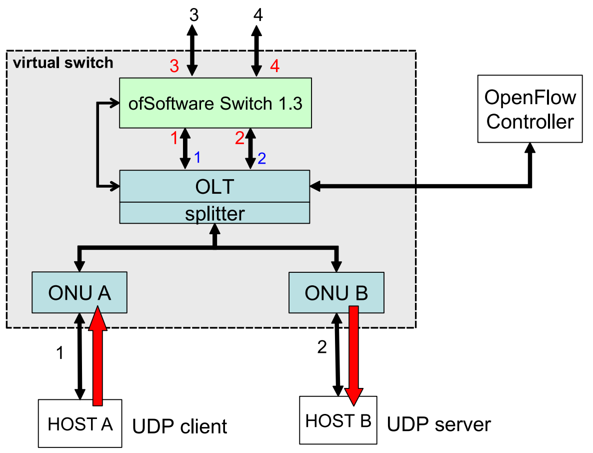


Fig. 9. Experiments for dynamic bandwidth control.

switch. For the upstream direction in the GPON system, the metering function of the virtual switch was achieved by using the GPON DBA function. For the downstream direction, the bandwidth control was implemented by the add-on OpenFlow switch. The settings for the experiments are shown in Fig. 9. We tested three combinations in the virtual switch. In the first experiment, the flow entered the system at port 1 and left the system from port 3. In other words, the flow came from ONU A and moved toward the third port of the add-on OpenFlow switch. In this set of experiments, the traffic source generated a 75 Mb/s UDP flow lasting 150 s. The rate of the meter was initially set to 25 Mb/s for the first 50 s, increased to 50 Mb/s for the second 50-s period, and after 100 s, the meter was raised to 100 Mb/s. The experimental results are shown in Fig. 10. During the experiments, all packets are logged for analysis. The horizontal axis is time in seconds, and the vertical axis is throughput in bits per second. The throughput is obtained using moving average under a 1 s window at 0.1 s resolution. From Fig. 10(a) the bandwidth was accurately limited to the desired 25 and 50 Mb/s for the first and second 50 s periods; however, because the meter was set to 100 Mb/s in the third period, a value larger than the flow rate from the traffic source, the meter did not impose any flow constraint—all 75 Mb/s were allowed through the switch.

In the second part, we evaluated the metering function for downstream traffic. The tested flow started from port 3 of the add-on switch and was destined for the ONU B port. In this experiment, we used the metering function provided by the soft switch directly, thus freeing the OLT from having to limit the downstream bandwidth. We used the same settings used in the previous experiment to configure the metering rates. Fig. 10(b) shows the experimental results. The results indicate that the downstream bandwidth can be successfully controlled by the soft switch. The small peak in Fig. 10(b) comes from the ofSoftswitch [37]. A transient behavior occurs occasionally when the setting of the meter changes. Since it returns to normal case rapidly, the transient behavior does not cause much anomaly in the system.

In the final experiments to test the metering function, the flow came from ONU A and was destined for ONU B. The

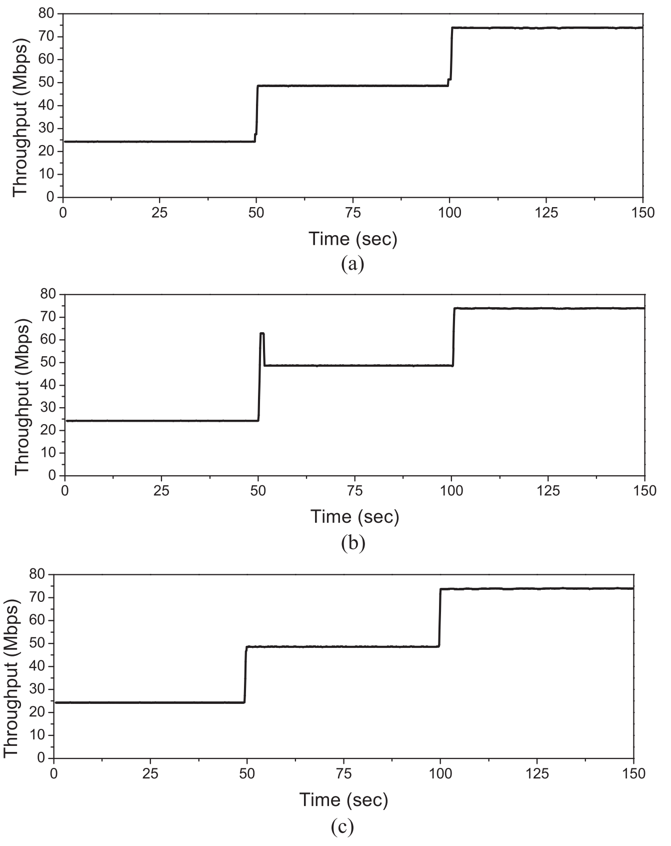


Fig. 10. Dynamic bandwidth controlled by the metering function. (a) Virtual port 1 to virtual port 3. (b) Virtual port 3 to virtual port 1. (c) Virtual port 1 to virtual port 2.

flow travelled in both upstream and downstream directions in the GPON. Again, we used the same metering settings as in the previous experiments. In this case, we needed to use only GPON DBA to perform the desired metering function. The results shown in Fig. 10(c) indicate that the desired bandwidth limitation can be realized in the virtual switch.

C. Port Status Change

In the third experiment, we evaluated the time required to detect a port status change. We were particularly interested in the time it takes for the virtual switch to detect a port down event. For an OpenFlow switch, when a port status change is detected, the switch generates an event and sends it to the OpenFlow controller automatically. To identify exactly how long the agent needs to detect a port status change and report the event to the controller, we sent a signal from the PC running the OpenFlow controller to the virtual switch. When the virtual switch received the signal, an active ONU port was turned off. When the agent detected the port status change, it generated a notification to the controller. The results are shown in Table II. To make performance comparisons, the table also includes the detection times measured from the 1 Gb/s ports in the HP 5900 and the EdgeCore AS4600. The results show that the virtual switch can react to a port status change within a very short time.

TABLE II
TIME TO DETECT A PORT STATUS CHANGE (MS)

Flow Entry Modification	Average	Maximum	Minimum
Port Status Change by Controller	22.7	45.0	14.5
EdgeCore AS4600	170	395	35
HP 5900	680	959	426

V. CONCLUSION AND FUTURE WORK

In this paper, we presented an architectural design and system implementation for a GPON based OpenFlow-enabled SDN virtual switch. Our virtual switch can be controlled by a standard OpenFlow 1.3 controller with no modifications. The virtual switch has the same flexibility and controllability as a real OpenFlow switch, but it also inherits the benefits of a GPON network for connecting multiple widespread sites in a cost effective manner. We implemented a prototype virtual switch based on a commercial GPON system that used port based VLANs for service differentiation. Our virtual switch provides standard OpenFlow-defined functions that include packet forwarding, dynamic bandwidth control, and statistical data collection. We take full advantage of multiple flow tables in the add-on OpenFlow switch to simplify the VLAN tagging problem. Experimental results show that the virtual switch modify flow entries and detect port status changes very quickly. By combining the DBA and bandwidth control of the add-on OpenFlow switch, our virtual switch can control bandwidth usage accurately and dynamically.

This work has demonstrated the feasibility for implementing a GPON virtual switch on a common commercial GPON system. However, due to the limitations of the Ethernet chips inside current OLTs, our system required an add-on OpenFlow switch for the prototype described here. In the future, we will be able to replace the Ethernet switch chip with an SDN switch chip. This new design can further reduce the cost for building a GPON-based virtual switch.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful and constructive comments, which greatly contributed to improve the final version of this paper. They also want to thank TECOM for technical support and for kindly making their GPON products available during the development of the system, and Y.-H. Shiao and I-F. Hung, both of whom are with the Industrial Technology Research Institute, Hsinchu, Taiwan, for their in-depth discussions.

REFERENCES

[1] W. Xia, Y. Wen, C. Heng Foh, D. Niyato, and H. Xie, “A survey on software-defined networking,” *IEEE Commun. Surveys Tut.*, vol. 17, no. 1, pp. 27–51, Jan.–Mar. 2015.

[2] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015. [Online]. Available: http://arxiv.org/pdf/1406.0440.pdf

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling innovation

in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[4] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 3–14.

[5] K. Nagase, "Software defined network application in hospital," *InImpact, J. Innovation Impact*, vol. 6. no. 1, pp. 1–11, 2013.

[6] E. Escalona, J. I. Aznar Baranda, L. Miguel C. Murillo, O. Gonzalez de Dios, G. Cossu, E. Salvadori, and F. M. Facca, "Using SDN for cloud services provisioning: The XIFI Use-case," in *Proc. IEEE SDN Future Netw. Services*, 2013, pp. 1–6.

[7] I. Ku, Y. Lu, and M. Gerla, "Software-defined mobile cloud: Architecture, services and use cases," presented at the Int. Wireless Communications Mobile Computing Conf., Nicosia, Cyprus, 2014.

[8] M. Jarschel, T. Zinner, T. Hoßfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases: A compass for SDN," *IEEE Commun. Mag.*, vol. 52, no. 6, pp. 210–217, Jun. 2014.

[9] OpenFlow Switch Specification 1.3.2. (2013 Apr.). [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.2.pdf

[10] S. Gringeri, N. Bitar, and T. J. Xia, "Extending software defined network principles to include optical transport," *IEEE Commun. Mag.*, vol. 51, no. 3, pp. 32–40, Mar. 2013.

[11] M. Channegowda, R. Nejabati, and D. Simeonidou, "Software-defined optical networks technology and infrastructure: Enabling software-defined optical network operations," *J. Opt. Commun. Netw.*, vol. 5, no. 10, pp. A274–A282, Oct. 2013.

[12] J. Zhang, Y. Ji, J. Zhang, R. Gu, Y. Zhao, S. Liu, K. Xu, M. Song, H. Li, and X. Wang, "Baseband unit cloud interconnection enabled by flexible grid optical networks with software defined elasticity," *IEEE Commun. Mag.*, vol. 53, no. 9, pp. 90–98, Sep. 2015.

[13] J. Oliveira, J. Oliveira, E. Magalhães, J. Januário, M. Siqueira, R. Scaraficci, M. Salvador, L. Mariote, N. Guerrero, L. Carvalho, F. van't Hooft, G. Santos, and M. Garrich, "Toward terabit autonomic optical networks based on a software defined adaptive/cognitive approach," *J. Opt. Commun. Netw.*, vol. 7, no. 3, pp. A421–A431, Mar. 2015.

[14] H. Chen, J. Zhang, Y. Zhao, J. Deng, W. Wang, R. He, X. Yu, Y. Ji, H. Zheng, Y. Lin, and H. Yang, "Experimental demonstration of datacenter resources integrated provisioning over multi-domain software defined optical networks," *J. Lightw. Technol.*, vol. 33, no. 8, pp. 1515–1521, Apr. 2015.

[15] X. Chen, M. Tornatore, S. Zhu, F. Ji, W. Zhou, C. Chen, D. Hu, L. Jiang, and Z. Zhu, "Flexible availability-aware differentiated protection in software-defined elastic optical networks," *J. Lightw. Technol.*, vol. 33, no. 18, pp. 3872–3882, Sep. 2015.

[16] H. Harai, H. Furukawa, K. Fujikawa, T. Miyazawa, and N. Wada, "Optical packet and circuit integrated networks and software defined networking extension," *J. Lightw. Technol.*, vol. 32, no. 16, pp. 2751–2759, Aug. 2014.

[17] S. Zhang, M. Tornatore, G. Shen, J. Zhang, and B. Mukherjee, "Evolving traffic grooming in multi-layer flexible-grid optical networks with software-defined elasticity," *J. Lightw. Technol.*, vol. 32, no. 16, pp. 2905–2914, Aug. 2014.

[18] Z. Zhu, C. Chen, X. Chen, S. Ma, L. Liu, X. Feng, and S. J. B. Yoo, "Demonstration of cooperative resource allocation in an openflow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, no. 8, pp. 1508–1514, Apr. 2015.

[19] D. Zhang, X. Song, C. Chen, H. Guo, J. Wu, and Y. Xia, "Software defined synergistic IP+optical resilient transport networks," *J. Opt. Commun. Netw.*, vol. 7, no. 2, pp. A209–A217, Feb. 2015.

[20] S. Yan, Y. Yan, B. Rahimzadeh Rofoee, Y. Shu, E. Hugues-Salas, G. Zervas, and D. Simeonidou, "Real-time Ethernet to software-defined sliceable superchannel transponder," *J. Lightw. Technol.*, vol. 33, no. 8, pp. 1571–1577, Apr. 2015.

[21] S. Peng, B. Guo, C. Jackson, R. Nejabati, F. Agraz, S. Spadaro, G. Bernini, N. Ciulli, and D. Simeonidou, "Multi-tenant software-defined hybrid optical switched data centre," *J. Lightw. Technol.*, vol. 33, no. 15, pp. 3224–3233, Aug. 2015.

[22] Optical transport use cases, ONF TR-509, Aug. 2014.

[23] Optical transport protocol extensions, ONF TS-022, version 1.0, Mar. 2015.

[24] Requirements analysis for transport OpenFlow/SDN, ONF TR-508, v1.0, Aug. 2014.

[25] Gigabit-capable passive optical networks (GPON), ITU-T, Rec. G.984, 2003.

[26] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Commun. Surveys Tut.*, vol. 18, no. 1, pp. 655–685, Jan. 2016.

[27] R. Gu, Y. Ji, P. Wei, and S. Zhang, "Software defined flexible and efficient passive optical networks for intra-datacenter communications," *Opt. Switching Netw.*, vol. 14, pp. 289–302, 2014.

[28] Y. Luo, X. Zhou, F. Effenberger, X. Yan, G. Peng, Y. Qian, and Y. Ma, "Time- and wavelength-division multiplexed passive optical network (TWDM-PON) for next-generation PON stage 2 (NG-PON2)," *J. Lightw. Technol.*, vol. 31, no. 4, pp. 587–593, Feb. 2013.

[29] P. Parol and M. Pawlowski, "Towards networks of the future: SDN paradigm introduction to PON networking for business applications," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, 2013, pp. 829–836.

[30] H. Woesner and D. Fritzsche, "SDN and openflow for converged access/aggregation networks," presented at the Optical Fiber Communication Conf./Nat. Fiber Optic Engineers Conf., Anaheim, CA, USA, 2013.

[31] A. Amokrane, J. Hwang, J. Xiao, and N. Anerousis, "Software defined enterprise passive optical network," in *Proc. IFIP 10th Int. Conf. Netw. Service Manage.*, 2014, pp. 406–411.

[32] A. Amokrane, J. Xiao, J. Hwang, and N. Anerousis, "Dynamic capacity management and traffic steering in enterprise passive optical networks," in *Proc. IEEE/IFIP Int. Symp. Integr. Netw. Manage.*, 2015, pp. 406–413.

[33] AT&T and Open Networking Lab. CORD: Central Office Re-Architected as a Datacenter [Online]. Available: http://xosproject.org/wp-content/uploads/2015/04/Whitepaper-CORD.pdf

[34] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, "FlowVisor: A network virtualization layer," *OpenFlow Consortium, Tech. Rep.*, OPENFLOW-TR-2009-1, 2009.

[35] S. S. W. Lee, K.-Y. Li, K.-Y. Chan, Y. C. Chung, and G.-H. Lai, "Design of bandwidth guaranteed openflow virtual networks using robust optimization," in *Proc. IEEE GLOBECOM*, Dec. 2014, pp. 1916–1922.

[36] RYU Certification (2015, Oct.). [Online]. Available: http://osrg.github.io/ryu/certification.html

[37] ofSoftswitch13 (2016, Feb.). [Online]. Available: https://github.com/CPqD/ofsoftswitch13

[38] NOX (2014, Feb.). [Online]. Available: https://github.com/noxrepo/nox

Steven S. W. Lee received the Ph.D. degree in electrical engineering from National Chung Cheng University, Min-Hsiung, Taiwan, in 1999. From 1999 to 2008, he was with the Industrial Technology Research Institute, Hsinchu, Taiwan, where he was the Leader of the Intelligent Optical Networking Project and a Section Manager of the Optical Communications Department. He was also a Research Associate Professor of National Chiao Tung University, during 2004–2007. In 2008, he joined National Chung Cheng University, where he is currently an Associate Professor of the Department of Communications Engineering. He has published more than 70 papers in international journals and conferences and is a coauthor of 20 international patents in the areas of broadband communications. His research interests include optical and broadband networking, network optimization, green network, and software-defined networking.

Kuang-Yi Li received the B.S. degree in information management from Saint John's University, Taipei, Taiwan, in 2010. He is currently working toward the Ph.D. degree in communications engineering at National Chung Cheng University, Min-Hsiung, Taiwan. His research interests include optical and broadband networking, network survivability, green network, and software-defined networking.

Ming-Shu Wu received the B.S. degree in electronic engineering from the National Kaohsiung University of Applied Sciences, Kaohsiung, Taiwan, in 2012, and the M. S. degree in communications engineering from National Chung Cheng University, Min-Hsiung, Taiwan, in 2015. He is currently a Firmware Engineer at Okayo Electronics, Taichung, Taiwan. His research interests include optical access network and software-defined networking.