Pushing Software Defined Networking to the access

Richard G. Clegg
Dept of Elec. Eng.
University College London
Email: richard@richardclegg.org

Jason Spencer
Dept of Elec. Eng.
University College London
Email: contact@jasonspencer.org

Raul Landa
Dept of Elec. Eng.
University College London
Email: raul.landa@ieee.org

Manoj Thakur
Dept of Elec. Eng.
University College London
Email: manoj.thakur@ucl.ac.uk

John Mitchell
Dept of Elec. Eng.
University College London
Email: j.mitchell@ucl.ac.uk

Miguel Rio
Dept of Elec. Eng.
University College London
Email: miguel.rio@ucl.ac.uk

***Abstract*—As Software Defined Networking (SDN) and in particular OpenFlow (OF) availability increases, the desire to extend its use in other scenarios appears. It would be appealing to include substantial parts of the network under OF control but until recently this implied replacing much of the hardware with OF enabled versions. There are some cases, such as access networks in which the benefits could be considerable but deal with a great amount of legacy equipment that is difficult to replace. In this case an alternative method of enabling OF on these devices would be useful. In this paper we describe an architecture and software which could enable OF on many access technologies with minimal changes. The software has been written and tested on a Gigabit Ethernet Passive Optical Network (GEPON). The approach is engineered to be easily ported to any access technology with minimal requirements made on that hardware.**

I. Introduction

Software Defined Networking (SDN) is becoming established as a vital part of the Internet ecosystem. It is now used in a huge range of different technologies, for example inter-data center communication for service brokerage over large scale distributed and heterogeneous cloud environments [1]; performance evaluation of virtual network functions migration across cloud-based edge networks [2]; dynamic traffic engineering and adaptive network design to efficiently map logical/virtual topologies on physical network infrastructures [3], [4].

As video delivery increasingly dominates the traffic composition of the Internet (according to CISCO, video in all its forms should reach 90% of the traffic by 2017 [5]) it is putting enormous pressure on the access network. If one takes into account new developments like 4K/8K, virtual reality and tablet computing, the only realistic long term solution to deliver this traffic is with fibre to the home/premises (FTTP) which will most likely be implemented using passive optical network technologies. This will represent a massive investment from companies and/or governments and is important that this infrastructure is upgradable by software at very little cost. Software defined networks will be crucial to achieve this.

OpenFlow [6] is the best-known SDN technology. It allows OF enabled devices to communicate with an OpenFlow Controller (OFC) that is capable of, in software, creating rules to modify how the switch data-plane works. It is relevant because OpenFlow is an established protocol that is beginning to be widely deployed in development and production networks. OpenFlow is an emerging network technology that allows experimenters to change the behaviour of the network as part of the experiment. Using the OpenFlow protocol gives a remote controller the power to modify the behaviour of network devices. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardised interface to add and remove flow entries. OpenFlow has certainly moved beyond the status of research software, for example, Google use it for WAN and data-centre control [7].

However, for access networks, a particular problem remains. For logistical reasons, hardware for access networks often remains in place for a long time. Much of the hardware is not in a rack in a data centre but in street-side cabinets and sometimes user premises. Replacement would require not simply buying new equipment but often getting access to on-street locations and perhaps customer premises. Therefore if the access is to get the benefits which SDN and OpenFlow can provide a new approach is needed which can get the technology working at line rate using existing hardware with minimal hardware replacement.

This paper describes a method which can be used with a subset of access technology (point to multi-point devices) in order to make it OpenFlow enabled. This method takes advantage of features of most access technologies which make them uniquely suited for this approach:

1) Access technologies typically have lower cost "tail-end" (or "user-end") devices (also sometimes known as consumer premises equipment) and a smarter "head-end" device on the "network" side.
2) Access technologies are usually in operation configured with a further switch or router (often physically situated within an exchange or data centre) on the head-end side of the switch.

In the most usual deployment all traffic from the tail end devices (even if they are communicating with each other) goes via the head-end switch. This gives an opportunity at the head end to modify the traffic to be OpenFlow enabled.

Using the architecture from section III the whole collection of head-end device, tail-end devices and head-end switch can be made to appear as if it were a single OpenFlow switch (albeit one distributed in space). This is achieved by an intermediate controller which intercepts and modifies OF messages. It is worth noting that an architecture with such an intermediate controller is not, in itself, novel (indeed it is core to FlowVisor [8]).

This work was done within the context of the EU project ALIEN that has as its aim to bring OpenFlow to new classes of devices not currently OpenFlow enabled. In some cases this involves a native implementation of OpenFlow on programmable hardware. In other cases (such as that described here) this involves an implementation on proprietary hardware which cannot be reprogrammed. All the implementations within ALIEN follow a common Hardware Abstraction Layer (HAL) which is described in [9].

The structure of this paper is as follows: section II describes in more detail the access networks that would be considered for this approach. Section III describes, in broad terms, the architecture used to get the best implementation of OpenFlow possible with such an approach. The aim is to create a general approach which can be quickly ported to access technologies with similar capabilities but different architectures. Section IV describes the finer design details and implementation problems which arise in these systems. Section V provides results from the implementations, showing that the hardware passes OpenFlow unit tests. Section VI describes the requirements to port the system to a new device. Finally section VII gives conclusions and further work.

II. Hardware requirements

The system used in our deployment (described more fully in section V) is the GEPON shown in figure 1. The details of the exact model and function of the devices is given in section V. The head-end device is the Optical Line Terminal (OLT) and the tail-end devices are Optical Network Units (ONUs). The system is Point-to-MultiPoint, that is to say all traffic from the OLT goes to every ONU and conversely each ONU can communicate directly only with the OLT.

The OLT is the most intelligent component in the system and the ONU are cheaper devices intended for installation in or near user premises. The OLT is responsible for negotiating the time division multiplexing between its connected ONUs and ensuring that newly connected ONU can join the system. To communicate with each other, traffic from ONU to ONU must travel via the OLT. The connections between the OLT and ONU are known as Link Layer IDs (LLIDs).

In its most usual deployment the GEPON has a switch or router sitting outside the OLT. In figure 2 this has been replaced with an OpenFlow enabled switch. Finally, the GEPON has a management port which enables commands to be sent for configuring ONUs, adding queuing priority, provisioning VLANs and so on.

The features described here are typical of access technologies and similar features are present in systems such as DOCSIS, DSLAM/xDSL and even some wireless technologies using IEEE 802.11. The architecture described here, then, has applicability much beyond the single system on which it is implemented. To emphasise the generality of the approach this paper will usually use the phrases “head-end device” and “tail-end device” to describe the parts of the access network instead of the GEPON specific terms OLT and ONU.

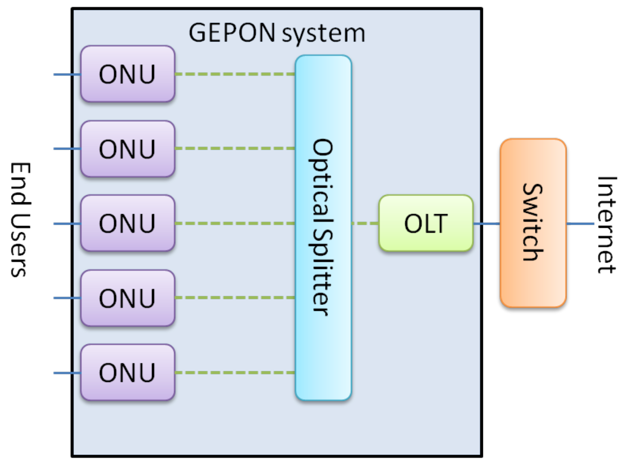

Fig. 1. The architecture which allows the GEPON to use OpenFlow

III. Architecture

This section describes the architecture used to enable OpenFlow in access technologies. This requires a head-end box sitting in front of the head-end device, however, as has been mentioned, this is the usual deployment of such a device.

Figure 2 shows the architecture for the OpenFlow enabled GEPON. The key changes are as follows:

- The switch outside the OLT has become OpenFlow enabled.
- An extra helper box labelled HAL (Hardware Abstraction Layer) has been added (this can be the same physical box as the OF switch.
- A connection is made from the OLT management port to the HAL.

The required changes to the system then are the addition of at least one physical machine which replaces an existing switch. The solid blue lines (far left and right) represent standard Ethernet frames. The dashed green lines (in the centre of the GEPON) are the optical section of the device. The dotted red lines (top right) represent OpenFlow control messages and the black piped line (between OLT and HAL) is the proprietary control path to the management interface of the head-end device.

Two tricks are key to the system. Firstly the OLT can provision VLAN end points and associate these with an LLID and hence an ONU. So, the OLT can set up a VLAN tag which associates with each ONU. When it receives traffic with that tag it removes the tag and sends it to the appropriate ONU. When it receives traffic from an ONU it adds an appropriate tag.

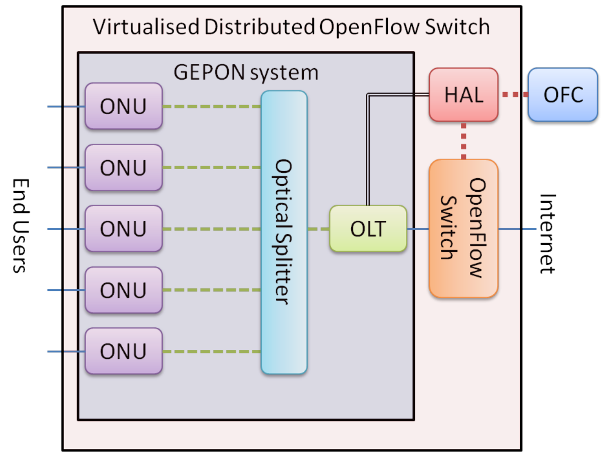

Fig. 2. The architecture which allows the GEPON to use OpenFlow

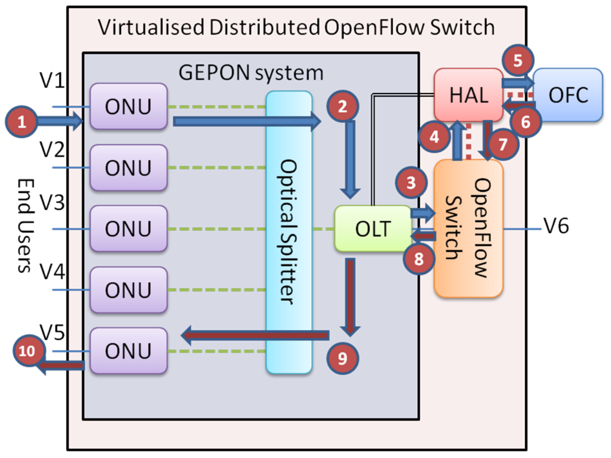

Fig. 3. An example traversal of the virtualised GEPON switch

Secondly, the HAL acts an intermediate OFC. The external OFC connects to the HAL and instead of seeing an OLT, ONU etc, sees only a single distributed switch with many ports: one port for the OLT and one port for each ONU. For OpenFlow interactions being sent to the GEPON switch from the OFC the HAL modifies the command to change which port it refers to and, if necessary, to add a VLAN tag. For OpenFlow interactions being sent to the OFC from the GEPON switch the HAL removes the VLAN tag (if present) and maps the interaction to the appropriate port.

The HAL also connects to the management port of the GEPON to set up appropriate VLAN tags and to listen for messages from the ONU which could indicate ports going down. A main role of the HAL can be thought of as being a map between VLAN/port pairs on the real physical system and virtual ports on the virtual distributed OpenFlow switch.

An example will help clarify. Take the system from figure 2. This will map to a distributed virtual OF switch with six ports, one corresponding to the user facing side of each ONU (call them $O_1, \ldots, O_5$). Let us call these virtual ports $V_1, \ldots, V_5$ and one corresponding to the outgoing network on the right hand side of the figure, call it V_6. For simplicity assume that $O_1, \ldots O_5$ are mapped to VLAN tags $T_1, \ldots, T_5$. Let us refer to the physical ports on the OpenFlow switch as P_1 for the port facing the OLT and P_2 for the port facing the exterior. The map, therefore, held by the HAL is:

$$\begin{aligned} P_1, T_1 &\leftrightarrow V_1 \\ P_1, T_2 &\leftrightarrow V_2 \\ P_1, T_3 &\leftrightarrow V_3 \\ P_1, T_4 &\leftrightarrow V_4 \\ P_1, T_5 &\leftrightarrow V_5 \\ P_2 &\leftrightarrow V_6 \end{aligned}$$

Figure 3 shows an example of a packet traversing such a system. The blue arrows show the outward journey and the red arrows the return journey of a hypothetical packet in the system. The external OFC connects to the HAL and knows nothing of the internal architecture. As far as the OFC can determine it is connected to an OpenFlow switch with six ports.

At (1) a packet enters the system. At (2) the packet becomes an optical frame at the ONU and goes through the Splitter to the OLT. At (3) the OLT tags the packet with tag T_1 and passes it to the OpenFlow switch which it enters through port P_1. The OpenFlow switch sees a packet on P_1 with tag T_1. The packet matches no FlowMods on the switch and triggers an OpenFlow PacketIn message which is sent to the HAL at (4). This accepts the packet, translates the pair P_1, T_1 to correspond to port V_1 and passes the packet to the OFC at (5). The OFC application chooses to respond with new FlowMod rule which forwards all packets from V_1 to V_5 and to forward the packet accordingly. At (6) the packet returns to the HAL to be output on port V_5. The HAL translates V_5 to a rule to tag the packet with T_5 and output it to P_1. At (7) the FlowMod rule is passed down to the OpenFlow switch. The rule which was to forward packets from V_1 to V_5 has been translated to a rule which forwards packets from P_1 tagged T_1 back to port P_1 tagged T_5. The OpenFlow switch duly forwards the original packet now tagged T_5 (and all subsequent packets from V_1) to the OLT in step (8). In step (9) the OLT sees the tag T_5 and removes the tag, forwarding the packet to ONU O_5 at (9). Finally at (10) the packet is output from port V_5 with no tag.

While the system is relatively simple to describe there are a number of deployment considerations, in particular around port statistics, tagging of VLANs (other than those used by the mapping) and ensuring continuity of communication. The implementation details are described in the next section. Of course the architecture is not tied to VLAN in particular. The system would work just as well with any alteration to the packets (which is here referred to as a tag) with the properties listed below.

- The OpenFlow switch can add the tag and match any packet against that tag in combination with the real underlying port.
- The head-end device can add the tag to any packet

entering the head-end device from a tail end device.

- The head-end device can route the packet according to the tag and remove the tag.

In addition, for a particular access technology to work with this system another requirement is that all traffic between tail-end devices must travel via the head-end OF enabled switch. Any access technology meeting these requirements could potentially be made OpenFlow aware using this approach.

IV. Design considerations

Section III described the general architecture used to convert the GEPON to be OpenFlow enabled. This section describes the actual technologies used in the deployment, problems faced and outstanding issues.

The deployment is critically focused on creation of the HAL for access devices (and particularly the GEPON). Important to both was the software in the Revised OpenFlow Library (ROFL)[1] and the eXtensible DataPath Daemon (xDPd)[2]. ROFL contains interfaces which abstract OpenFlow messages as C++ functions or, conversely, can generate OpenFlow messages in C++. It contains C++ abstractions representing most structures within an OpenFlow device and, as such, is much more general than an OpenFlow Controller platform. xDPd is a multi-platform, multi OF version, open-source datapath focusing on performance and extensibility. It can be used by coders wishing to create a native OF implementation which runs directly on their programmable hardware (this was not an option for the GEPON used here that has proprietary hardware and is not modifiable to such an extent).

A. GEPON deployment

The functions from ROFL were used as the basis for HAL on the GEPON which exists as the conversion software known as the eXtensible Control Path Daemon (xCPd). The xCPd software is available at https://github.com/richardclegg/xdpd.

The HAL box in the testbed was a standard PC containing a NetFPGA card programmed as an OpenFlow switch (in the final deployment this will also run the xDPd software but the intermediate version runs the Stanford OpenFlow 1.0 switch). This means that the datapath is pure hardware and hence has no slowdown. The xCPd runs on the PC as software and connects to the OpenFlow instance running on the NetFPGA and also offers external access to an external OFC.

B. Minimising hardware specific requirements

Some OpenFlow requirements can only be achieved via interaction with the hardware. An obvious example would be the initial provisioning of VLANs for system. However, there are more subtle examples where a "pure" implementation using only xCPd, xDPd and VLAN provisioning does not work. OpenFlow 1.0 messages can be split into three groups according to how they will work with xCPd and the described architecture. The first group is those that simply work using this architecture without needing translation. For example the EchoRequest and EchoReply messages which simply establish a connection. Those messages become an EchoRequest from the controller passed to xCPd, passed down to the OpenFlow switch and, conversely, an EchoReply passed back up from the switch to xCPd and onward to the controller. This group includes: Hello, EchoRequest, EchoReply, Vendor, FeaturesRequest, FeaturesReply, BarrierRequest, BarrierReply, GetConfigRequest, GetConfigReply, SetConfig, PortStatus and QueueGetConfigReply.

The second group is those messages which require translation into the VLAN format. For example, a FlowMod cannot typically be directly sent down from xCPd to the switch. A FlowMod consists of a match and an action. Both the match and the action require translation so that, for example, if the FlowMod match is against a virtual port provided by xCPd this must be translated to a match against a real port and (if that port is the port facing the OLT) a VLAN tag. This group includes: PacketIn, PacketOut, StatsRequest/StatsReply (for a FlowMod), Error and FlowRemoved.

The third group is those messages which cannot be "faked" using the VLAN technique: PortMod, StatsRequest/StatsReply (for a Port) and QueueGetConfigRequest. These messages work as expected on "real" ports – that is ports which correspond to a single outgoing port on the real underlying hardware rather than those mapped with VLANs. However, for virtual ports then these messages present a problem, for example, if a configuration change requests a port to be taken down, then taking down the underlying physical port would remove other virtual ports as a side effect. The best way to reliably implement the request to take that port down is via a request to the head-end device made through the management interface and xCPd is configured to connect to an external module which translates these messages into appropriate head-end management commands (these are usually proprietary and hence will need translating individually). A StatsRequest/StartReply targeting a virtual port cannot get accurate statistics by querying the underlying physical port as this would provide statistics aggregated over all virtual ports sharing that physical port. In practice this means that a query for the statistics related to a single tail end device would have to, in reality, query the underlying hardware. This can be achieved on the OLT via its management interface and it would be expected that similar requirements can be achieved for most access devices.

Apart from the messages from the previous section, the current xCPd design can achieve all OpenFlow actions with the exception of the optional Enqueue action. Problems arise, however, when users require VLANs across the hardware as VLANs are being used as a method to identify virtual ports. QinQ (802.1ad) allows VLAN tags to be "stacked" (a tagged packet can be tagged with an outer VLAN header) and no problem arises. Without QinQ if a packet arrives from the tail end with an unknown VLAN tag then there is no way to know from which virtual port this packet arrived unless it has been preconfigured in the head-end device. Packets arriving at the head-end with a VLAN tag present a problem

[1]https://www.codebasin.net/redmine/projects/rofl-core/

[2]https://www.codebasin.net/redmine/projects/xdpd/

if they have to exit via a tail end device. The tag must be replaced (to ensure the packet is routed to the correct tail end device) unless the VLAN has been provisioned at the head-end device. Obviously tagging with a VLAN which is currently in use by the xCPd tagging system introduces its own problem. These problems could be avoided if other (non VLAN) tagging mechanisms were used but such tagging might bring problems of its own. This porting requirements for xCPd to run on hardware other than the specific model of GEPON described here are discussed in more detail in section VI.

V. Testing results and porting requirements

The previously described architecture has been implemented and tested. The state of the art unit tests for OpenFlow are those in OFtest[3]. This tests various OpenFlow functionalities separately by connecting to the switch under test and generating events (PacketIn, PacketOut etc). OFtest acts as the OFC for the switch, makes a connection and runs its tests, giving output to say whether each test has passed or failed.

The GEPON design was illustrated in figure 1 where the GEPON is the set of devices within the larger dark blue box on that diagram. It consists of three types of device:

1) The Optical Line Terminal (OLT), in this case the Planet technologies EPL-1000[4];
2) The Splitter, in this case Planet technologies EPL-SPT-32; and
3) One or more Optical Network Units (ONUs): Planet technologies EPN-102[5].

The head-end switch in the test system was a standard PC with a NetFPGA card that can be flashed to act as an OpenFlow 1.0 switch. This PC also ran the xCPd software. Behind the tail-end ONUs in the test system was another PC with a four port NIC which allowed up to four test ONUs to connect to it. In the test configuration the internet-facing port of the head-end PC was connected back to this PC to allow OFtest to put traffic into all ports on the distributed switch.

The OLT takes ordinary IEEE 802.3u Ethernet at its external interface and converts it to/from optical signals that go to and from the splitter. The splitter is a passive device which takes that signal from the OLT and optically power divides it to all the ONUs in the system. All optical signals from the ONUs, similarly, are sent onwards to the OLT. Finally the ONUs have external IEEE 802.3u Ethernet interfaces and convert the electrical signal to optical to send to the splitter. They use time division multiplexing to avoid clashes and to determine which traffic should be received by which ONU. Wavelength division multiplexing is used to separate upstream (to the head-end) and downstream traffic.

A. Tests on GEPON

The main testing for the GEPON required wiring the system so that OF test could access all the "ports" of the switch, that is to say the outgoing port from the OpenFlow switch next to the OLT and a number of ONUs. This was achieved by connecting each of the ONUs under test into a network interface on a single PC and connecting the outgoing port from the OpenFlow switch into another port on the same switch. In these tests we used three ONUs so the entire GEPON plus switch system presented as a four port OpenFlow switch. Table I shows the results of the tests.

Test	**Result**
Echo	Passes using xCPd
EchoWithData	Passes using xCPd
PacketIn	Passes on systems with QinQ. On systems without, one part of this test fails when VLAN tags are used
PacketInBroadcastCheck	Passes using xCPd
PacketOut	Passes using xCPd port mapping
PacketOutMC	Passes using xCPd port mapping
FlowStatsGet	Passes using xCPd port mapping
TableStatsGet	Passes using xCPd
DescStatsGet	Passes using xCPd
FlowMod	Passes using xCPd port mapping
PortConfigMod	Requires hardware specific code to pass correctly
PortConfigModErr	Requires hardware specific code to pass correctly
BadMessage	Passes using xCPd
TableModConfig	Passes using xCPd

TABLE I
Basic OFtest unit tests

As can be seen, the majority of basic tests in OFT are passed by xCPd with no need to write code specific for the hardware. The PacketIn test fails because one part of that test uses a VLAN tagged packet. This fails on hardware that does not implement QinQ if the tagged packet traverses the head-end of the hardware as the new VLAN strips the original packet header (a system with QinQ would add a new VLAN header).

VI. Porting requirements

A. Porting requirements

The aim of this work is not simply to provide a way to get OpenFlow working on one specific model of GEPON but, instead, to provide the simplest possible method for a large class of devices to become OpenFlow enabled. For complete OpenFlow 1.0 functionality, any access device which supports VLANs (or any equivalent tag matching the requirements given at the end of section III) could be made to implement OF1.0. However, certain code would need to be provided that is hardware specific (communication along the black line to the management port of the OLT in figure 2). This requires the following code:

1) Code to query the head-end device for port statistics specific to the tail-end devices. In the absence of this then port statistics will be given for the underlying physical port and hence, for example, the number of packets, bytes and errors will be a total over all the tail-end devices.
2) Code which can modify the links to the end user devices as required by the OpenFlow PortMod command.

[3]http://www.projectfloodlight.org/oftest/
[4]http://www.planet.com.tw/en/product/product.php?id=25817
[5]http://www.planet.com.tw/en/product/product.php?id=22826

3) In OpenFlow 1.0 queue configuration "takes place outside the OpenFlow protocol, either through a command line tool or through an external dedicated configuration protocol". OpenFlow does, however, provide the ability to query queue configurations and optionally provides an action to enqueue packets to a given queue. Queries for queue configurations could be achieved in two ways:
   a) Ensure that a user configuring the queue on the underlying hardware also gives xCPd the same information.
   b) Write code to query the head-end device about its queue configurations (this is optional in the OF 1.0 specification).
4) Optional: Code which automatically configures tags on the head-end device on start up.

It should be noted though that in some cases certain equipment may respond relatively slowly (in terms of the timescales considered for network deployment) and this could cause buffering problems if the hardware response to these configuration commands is too slow.

If the group porting to the new hardware do not perform the steps then it will have certain implications. Failure to do 1) will mean that xCPd will fall back to querying the underlying physical port for statistics (or with a configurable option, fail to return any statistics) and statistics returned will be the sum of the statistics on the virtual ports associated with that port. (In figure 3 then V_6 will get accurate statistics and $V_1, \ldots, V_5$ will return the same answer which is the sum of their statistics.) Failure to do 2) will again mean that the PortMod will fall back to the underlying physical port (or with a configurable option, fail to return any statistics), so, for example, if that port is shut all physical ports are shut. Failure to do 3) will mean that OpenFlow will not be able to operate with queues (which is an optional feature in OF1.0). Failure to do 4) will simply mean that the user needs to make a one off adjustment to the device through its management interface.

Overall, then, the system should easily port to any access device with the characteristics of the GEPON system in figure 1. Devices that do not support QinQ (VLAN stacking) can never fully make use of VLANs. Without writing hardware specific code the majority of compulsory OpenFlow 1.0 features are supported. With hardware specific code then all OF1.0 features can be supported. The deployment requirements are an OpenFlow switch sitting outside the access network and the xCPd software. It is not important where this runs and it could be co-located with the OpenFlow Controller for the switch or run on the same box as the switch.

VII. Conclusions and further work

This paper has described work to enable OpenFlow for access hardware meeting certain requirements. The architecture presented uses tagging (in specific VLAN tags) to allow an access network consisting of several devices to masquerade as a single distributed OpenFlow enabled switch. This involves introducing an OpenFlow switch outside the head-end of the access network and running intermediate control software (known as the eXtensible Control Path daemon, xCPd) that speaks OpenFlow northbound and southbound and sits between the OpenFlow Controller and the switch modifying OpenFlow messages. The architecture has been implemented as software for OpenFlow 1.0 using the Revised Open Flow Library (ROFL). This software has been tested and passes OpenFlow unit tests from the test suite OFtest. For the majority of OpenFlow functionality the software would work unmodified on a large amount of access hardware given appropriate (very simple) configuration. For the remainder of OpenFlow functionality, extremely simple drivers can be written to interface directly with the hardware via its management interface. This means that implementing this system for new hardware which meets the requirements should be possible in an extremely short time span enabling OpenFlow functionality to be pushed to a large variety of access devices.

Development of the xCPd design is ongoing. In particular a porting guide is being produced to allow developers wishing to bring xCPd to new hardware to produce the required code for their hardware. Current work is expanding the code to work with OpenFlow 1.2 and to include instructions for groups porting to their own hardware.

Acknowledgements

This research has received funding from the Seventh Framework Programme of the European Commission, through the ALIEN (317880) project.

References

[1] M. Mechtri, I. Houidi, W. Louati, and D. Zeghlache, "Sdn for inter cloud networking," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, 2013, pp. 1–7.

[2] F. Callegati and W. Cerroni, "Live migration of virtualized edge networks: Analytical modeling and performance evaluation," in *IEEE SDN for Future Networks and Services*, 2013, pp. 1–6.

[3] R. Trivisonno, I. Vaishnavi, R. Guerzoni, Z. Despotovic, A. Hecker, S. Beker, and D. Soldani, "Virtual links mapping in future sdn-enabled networks," in *IEEE SDN for Future Networks and Services*, 2013, pp. 1–5.

[4] J. Mueller, A. Wierz, and T. Magedanz, "Scalable on-demand network management module for software defined telecommunication networks," in *IEEE SDN for Future Networks and Services*, 2013, pp. 1–6.

[5] Cisco systems, http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf, 2013.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[7] "Openflow@google," Presentation at Open Networking Summit http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf, 2012.

[8] R. Sherwood, G. Gibb, K. K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," OpenFlow Switch Consortium, Tech. Rep., Tech. Rep., 2009.

[9] ALIEN Consortium, "Hardware abstraction layer (hal) whitepaper," Available from ALIEN website: http://www.fp7-alien.eu/files/deliverables/ALIEN-HAL-whitepaper.pdf, 2013.