A L I E N

ABSTRACTION LAYER FOR IMPLEMENTATION OF EXTENSIONS IN PROGRAMMABLE NETWORKS

Collaborative project co-funded by the European Commission within the Seventh Framework Programme

Grant agreement no: 317880
Project acronym: ALIEN
Project full title: "Abstraction Layer for Implementation of Extensions in programmable Networks"
Project start date: 01/10/12
Project duration: 24 months

Deliverable D5.3
Experimental-driven research

Version 8.1

Due date:	30/10/2014
Submission date:	12/11/2014
Editor:	Marc Bruyere (Dell Force10), Maider Huarte (EHU)
Internal reviewers:	Maider Huarte, Eduardo Jacob (EHU)
Author list:	Marc Bruyere (Dell Force10), Damian Parniewicz, Łukasz Ogrodowczyk, Bartosz Belter, Artur Binczewski (PSNC), Roberto Doriguzzi Corin, Michele Santuari (CREATE-NET), Tasos Vlachogiannis (UNIVBRIS), Eduardo Jacob, Victor Fuentes, Maider Huarte (EHU), Mariusz Żal, Marek Michalski, Janusz Kleban, Remigiusz Rajewski (PUT).

Dissemination Level

☒	**PU:**	Public
☐	**PP:**	Restricted to other programme participants (including the Commission Services)
☐	**RE:**	Restricted to a group specified by the consortium (including the Commission Services)
☐	**CO**	Confidential, only for members of the consortium (including the Commission Services)

© Authors and their corresponding institutions License: CC-BY-NC http://creativecommons.org/licenses/by-nc/4.0/legalcode
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page.

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Abstract

This deliverable describes how project partners have extended their ALIEN HAL developments and reflects the experimental-driven research carried out. The outcome of this work, which enlarges the initial scope of the deliverable, shows the tremendous potential of the ALIEN HAL approach.

Five developments are described in this deliverable: Datapath programming, the design of a new Quality of Service (QoS) extension for OpenFlow that takes into account the specificities of a DOCSIS access network, the integration of the Time Based Aggregated Manager (TBAM) and its application to Multicast and IPv6 traffic, a performance validation of some of the Ethernet based ALIEN devices and a multivendor evaluation of the HAL in the optical domain.

Finally, some conclusions are presented highlighting possible post ALIEN project extensions.

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Table of Contents

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Figure Summary

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Executive Summary

This last WP5 deliverable will present various experiments with an experimental-driven research approach. All partners have been investigating and experimentally validating the potential of ALIEN as a starting point for implementing innovative and ground-breaking ideas for new networking architectures and service paradigms.

This deliverable is focusing on extending the original developments done in the project, where every partner led the development of support for a different kind of equipment in a different context.

This deliverable is also targeted at research network engineers and technical teams working in the SDN domain alike the previous deliverable [D5.2]. The document contains low-level technical details of the experiments (which are included, for the sake of the clarity, in the appendixes), complemented with conclusions and lessons learned from this process. A list of key points is as follows:

- The API Path Abstraction Datapath of the first experiment has demonstrated how to successfully manage protocols knowledge inside an OpenFlow switch.

- The design and implementation of OpenFlow extensions to make available the native Quality of Service (QoS) characteristics of a DOCSIS based access networks are presented and demonstrated. This gives the possibility to deploy services in which the QoS is dynamically managed.

- The Time Based Aggregate Manager (TBAM) has demonstrated how the original OF1.0 based OFELIA Control Framework has been extended to support different OpenFlow versions and this is demonstrated with multicast and IPv6 experiments.

- The performance validation of some of the Ethernet based processing developments is presented by using the well-known OFLOPS test.

- Finally, cross-connecting a variety of vendor equipment shows a multivendor evaluation of the ALIEN HAL in the optical domain.

This experimentally-driven research shows how ALIEN approach can help in joining the two ends of academy-driven visionary research and industry-driven testing and experimentation approach.

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

1 Introduction

The Alien HAL allows hardware that was natively not OpenFlow compatible to be placed under OpenFlow control. In the previous deliverable, in the first place, the compatibility with a generic OpenFlow based application was demonstrated. Later a CCN based application, CONET, was deployed over the OFELIA experimental facility showing that ALIEN hardware can be integrated to the experimental facility and it can support custom applications.. This deliverable presents further developments around the HAL concept and reflects the experimental-driven research done in task T5.3, based on the work done in T5.2.

All partners have been working on extending ALIEN HAL first developments presented in D5.2, carrying out several different experiments and developments. The results of this work reflect a tremendous potential of the ALIEN HAL approach. Five developments are described in this deliverable: Datapath programming, the design of a new Quality of Service (QoS) extensions for OpenFlow that take into account the specificities of a DOCSIS access network, the integration of the Time Based Aggregated Manager (TBAM) and its application to Multicast and IPv6 traffic, performance validations of some of the Ethernet based ALIEN devices and a multivendor evaluation of the HAL in the optical domain.

Each experiment is presented providing an overview, the goals to achieve, a detailed description of the implementation (if needed), the environment description, the testing procedure overview, the results, and the validation. For simplicity, the experiment results are provided separately in the corresponding appendix (Appendixes I to V, respectively for each experiment), and an analysis of them is reported in the validation subsection, so that the appendixes can be referred for detailed information about the results.

This document is structured as follows; Section 2 describes the experimentation on Datapath protocol programming. In section 3, the QoS extensions to OF implementation are presented along with the related tests. Section 4 describes TBAM integration and demonstration with Multicast and IPv6 traffic experiments. Section 5 dedicated to two performance experiments, one conducted with NetFPGA technology and the other with EZappliance. In Section 6, description of the multi-vendor evaluation of HAL in optical domain is given. Finally, an overall summary and conclusions are presented in Section 7.

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

2 Datapath protocol programming

2.1 Experiment overview

Despite continuous developments in this area, Software Defined Networking (SDN) still seeks for a flexible way of defining network device behaviour. One of the currently discussed topics is making network switches independent from data plane protocol formats. That should allow for more efficient usage of TCAM memory, enabling faster network innovation with usage of newly designed protocols like VXLAN, TRILL, etc. and also should allow for adapting of the network infrastructure to Future Internet solutions which are proposing to replace current IP protocol stack and to use completely new data packet formats.

The main fact which triggered that discussion was that data plane protocols have quite simple structure because they are designed as fields of simple ordered set of bytes or bits, which can be easily and fast parsed in the hardware. Thanks to the simplicity of data plane headers they can be described with usage of quite simple schema. Protocol descriptions using that schema can be passed to the device datapath in order to teach the datapath how to parse network packets.

Another important fact is that programmable network devices, like network processors, contain memory where simple procedure codes can be stored. This can be used for the definition of forwarding functions which will be uploaded to the datapath. Forwarding functions tell datapath how to processes the network traffic in the device. They can also contain the logic performed with a full datapath processing speed. Forwarding functions are more general equivalent of OpenFlow instructions.

Programmable Abstraction of Datapath (PAD), initially described in [D2.2], utilizes the findings described above, by exposing API interfaces (see Figure 2-1) which could be used by SDN Control Plane to get hardware capabilities (i.e.: processor architecture, bytes endianness, length of the machine word, arithmetic functions supported, etc.), set logical entities in the datapath memory, define forwarding functions and teach data plane protocols.

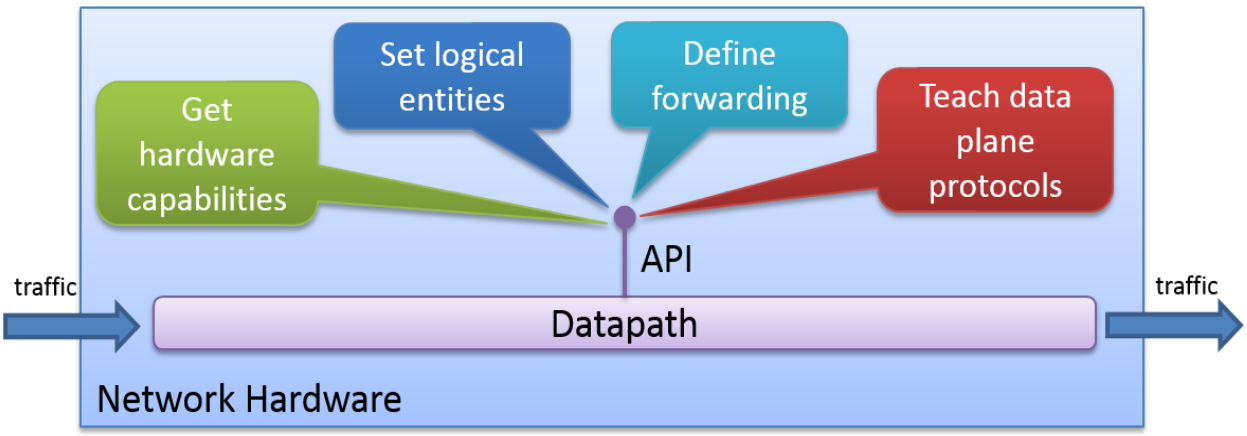


Figure 2-1 Programmable Abstraction of Datapath API overview

In the experiments with PAD, we would like to demonstrate usage of the last two mentioned functionalities of PAD API:

- Definition of data plane protocols
- Definition of forwarding actions

This API can be utilized by a new SDN Control Plane functional component (Data Plane Protocol Manager) which triggers protocol and actions interpretation for purpose of OpenFlow controller and all switches available in the network. The validation of PAD API starts with usage of software switches which makes it feasible in the timeframe of the ALIEN project (proper implementation for network hardware, e.g.: EZchip network processor will be much more complex and time-consuming).

The general idea of PAD API usage and main elements are presented in Figure 2-2.

2.2 Experiment goals

The testing performed in datapath protocol programming demonstration were performed in order to show that PAD API can be used successfully to manage protocols knowledge inside OpenFlow software switch and OpenFlow controller. Another goal of these tests is to investigate possibility of the protocol "learning" in the OpenFlow software switch in process of real evaluation of the code performing network frame parsing, classification and modification. Additionally, during the development of datapath protocol programming software, we were investigating already existing code related to well know IP protocols in xDPd [xDPD] and ROFL [ROFL] in order to investigate if we are capable to express structures and mechanisms of these protocols in the P4 protocol schema used by PAD API.

2.3 Detailed description of PAD implementation

For the PAD API demonstration, we decided to use OpenFlow GNU-Linux software switch from xDPd framework and ROFL-based OpenFlow controller as the base of our SDN network. We have developed a code generator [PAD-GITHUB], which generates a bunch of source code files for both ROFL and xDPd/GNU-Linux. The generated code files can be uploaded, in

the form of the source code or already compiled binaries, to hardware where OpenFlow controller and software switches are running.

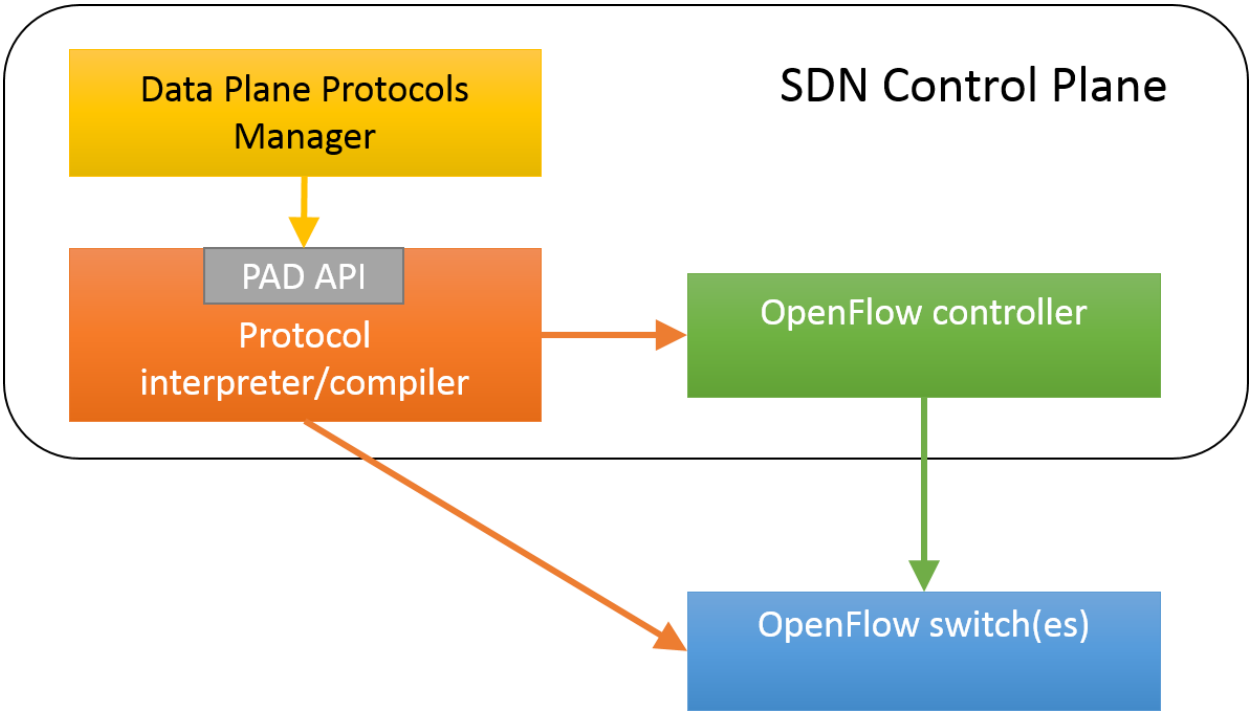


Figure 2-2 Protocol programmability in SDN network using PAD API

The code is generated basing on protocol headers, parsers and actions expressed in P4 language [P4]. Protocol and actions descriptions are passed from simple PAD API client to the code generator with usage of PAD API methods. Then P4 schema grammar is interpreted using Python pyparsing library [PYPARSING].

The PAD code generator for ROFL and xDPd generate files for (see Figure 2-3):

- OpenFlow experimental matches and actions classes
- Data plane protocol header classes and field manipulation functions
- Data plane header parsing functions
- Functions defining frame actions
- Translation of OpenFlow match and action extensions to internal xDPd AFA match and action structures

Currently, PAD code generator generates code for branch 'master-0.3' of both xDPd and ROFL projects. That branch supports OpenFlow version 1.0 and 1.2 versions only. OpenFlow version 1.0 is completely out of scope of the experiment, because it doesn't support OpenFlow experimental matches and actions which are very important for our PAD implementation because they allow interacting with newly defined protocols and actions without any modification to OpenFlow protocol itself. Thus, in our demonstration, we are using OpenFlow version 1.2.

When a new protocols and actions are declared by PAD client, a new source code files are generated, both OF controller and software switches must be recompiled and finally restarted.

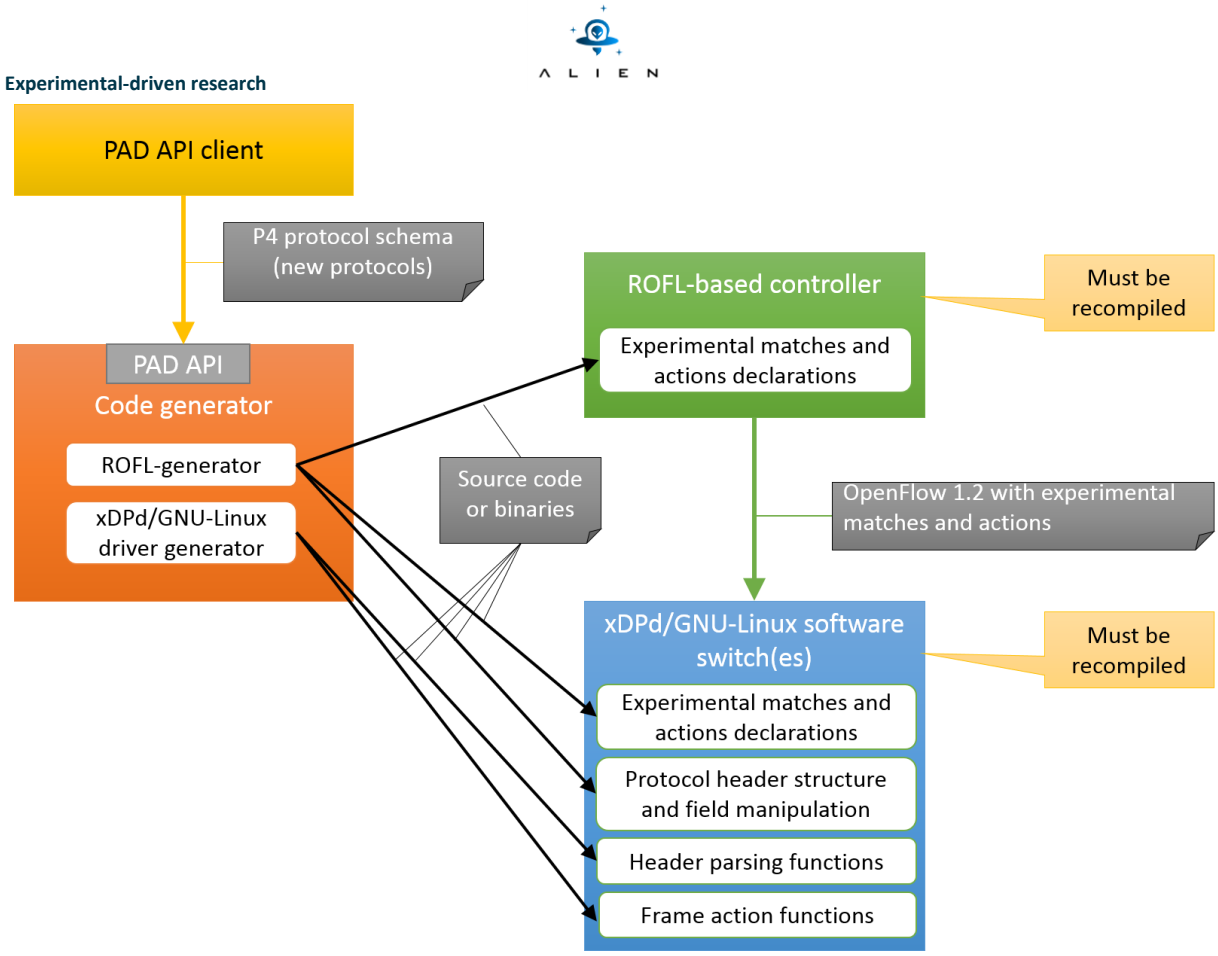


Figure 2-3 Details of protocol programmability implementation using PAD API, xDPd software switches and ROFL-based controller

2.4 Experiment environment

Datapath protocol programming testings are performed using resources from two OFELIA islands (Figure 2-4):

- PSNC island:
  - 1 EZappliance device exposed as OpenFlow switch
  - 1 server machine for deploying Virtual Machines (i.e. xDPd software switch, ROFL-based controller, PAD client and code generator)
- UNIVBRIS island:
  - NEC OpenFlow switches
  - 1 server machine for deploying Virtual Machines (i.e. xDPd software switch and POX controller)

OFELIA environment doesn't support deploying of the OpenFlow software switches by experimenters themselves. We decided to deploy two xDPd/GNU-Linux software switches within Virtual Machines running on edges of the OFELIA network both in PSNC and UNIVBRIS islands. These two software switches were interconnected by OFELIA network infrastructure (EZappliance and NEC switches) which were "statically" configured during all datapath protocol programming tests by POX controller (deployed in UNIVBRIS island). The POX controller established Layer 2 connectivity between xDPd software switches with usage of VLAN 700. VLAN 700 was ended on network interfaces of both VM servers, were VLAN tags were

added and removed. Thus, xDPd/GNU-Linux software switches were not aware of VLAN tagging in the network. We used also TAP interfaces, which are virtual Layer-2 network interfaces created by user-space programs as additional network interfaces for xDPd/GNU-Linux software switches. All software components of SDN control of datapath protocol programming were deployed in PSNC Island.

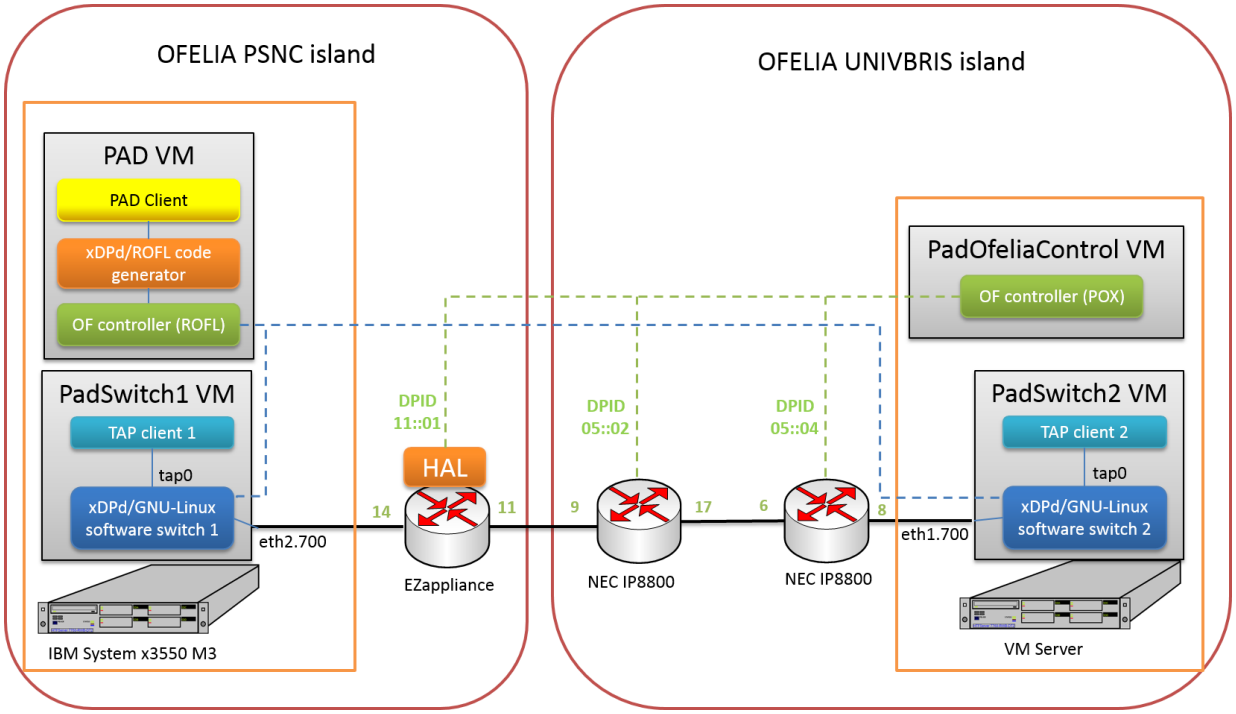


Figure 2-4 OFELIA resources used in datapath protocol programming testing

Figure 2-5 presents the logical topology which was established for purpose of datapath protocol programming tests. ROFL-based controller was controlling only software switches. Both ROFL-controller and xDPd software switches were modified by PAD client and code generator to include new data plane protocol descriptions. TAP clients were deployed as Python programs using pytun [PYTUN] library which allows for creation of tun interfaces as well as generating frames for this virtual interface (frames are seen for all user-space application as coming from the network) and receiving frames from applications (application try send network frames through that interface to the network).

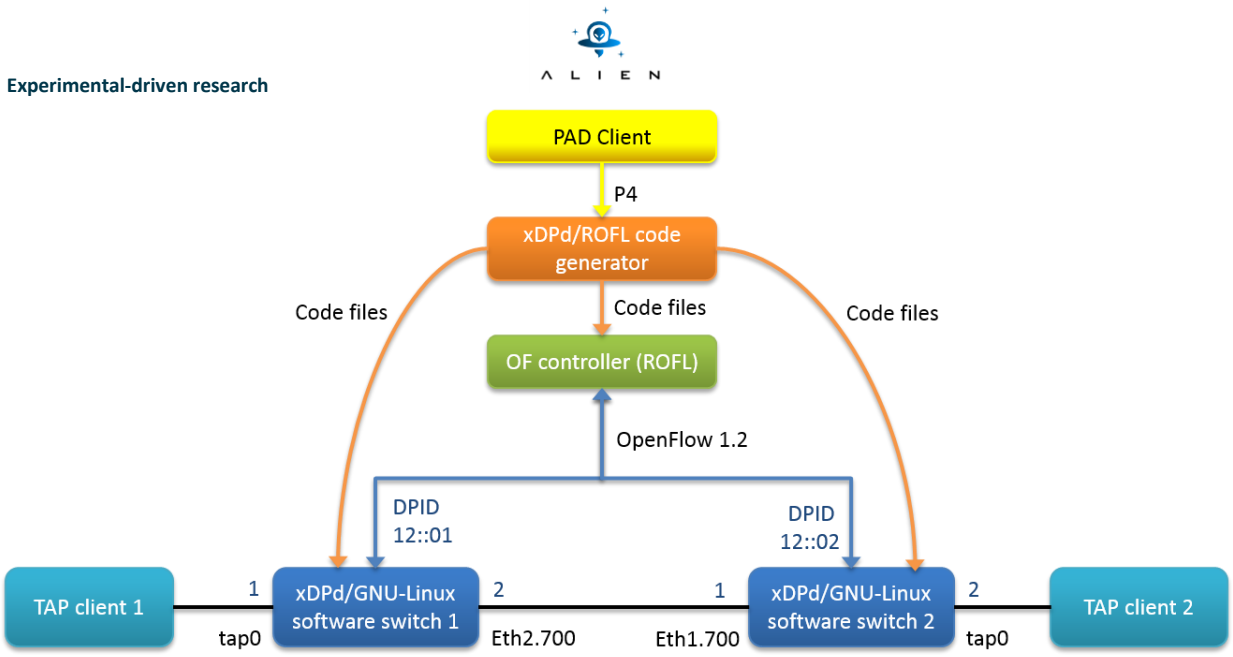


Figure 2-5 Logical view of the environment for PAD API tests

2.5 Testing procedure overview

Datapath protocol programming testing is composed of four phases:

- Step1: Loading a whole protocol description
- Step 2: Setting network configuration and processing network frames
- Step 3: Updating part of the protocol description
- Step 4: Setting network configuration after protocol update and processing network frames

In the first step, the PAD client is passing a complete P4 language description to the SDN network, containing the structure of protocol headers, protocols parsers and actions.

Figure 2-6 presents the structure and order of headers expressed in a description passed by PAD client to code generator.

Ethernet header			ICTP header	
dst_addr	src_addr	ethertype	nid	csn
6 octets	6 octets	2 octets	4 octets	4 octets

Figure 2-6 Structure of Ethernet and ICTP headers defined by P4 schema

When the protocol description is translated into code files and compiled with ROFL and xDPd, then ROFL-based controller as well all xDPd/GNU-Linux software switches are restarted and SDN network is ready to work.

In the second step, ROFL-based controller populates both software switches with flow entries containing the generated actions:

- for xDPd switch 1: push-ictp, set-field-nid, set-field-csn
- for xDPd switch 2: pop-ictp

In this way, the first software switch adds ICTP header directly after Ethernet header to all frames received on interface 1 (generated by TAP client 1) and set values of ICTP header fields. The modified frames are transmitted by interface 2 to the second software switch. On the second software switch ICTP headers are removed and frames are transmitted by interface 2 to the TAP client 2. The whole frames processing is presented in Figure 2-7.

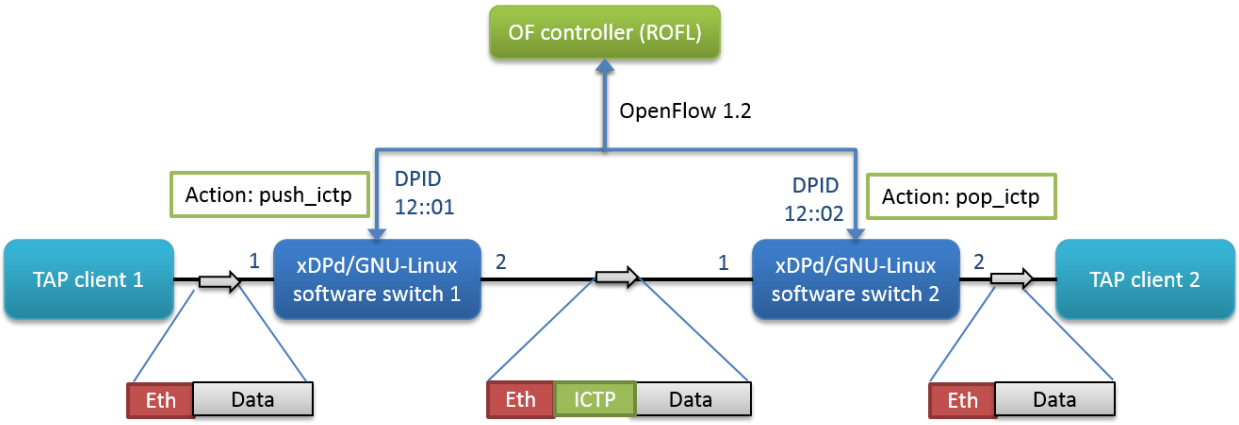
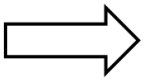


Figure 2-7 OpenFlow controller sets actions related to newly defined ICTP protocol

In the third step, the PAD client is removing ICTP header description and adding a new description of this header which is, right now, composed of more fields. Besides two 32-bits fields, the ICTP header includes one 1-byte field, one 2-bytes field and 1-byte padding expressed in the schema as __skip__ (__skip__ field is a special field, added by us to P4 syntax, which doesn't generate OpenFlow extensions, matches and actions). The change of ICTP header is presented bellow:

```
header ictp {
    fields {
        nid : 32;
        csn : 32;
    }
}
```



```
header ictp {
    fields {
        version : 8;
        slice : 16;
        __skip__ : 8;
        nid : 32;
        csn : 32;
    }
}
```

In the fourth step, we are performing the same actions as in the second phase:

- send the same flow entries to both software switches,
- generate the same Ethernet frames by TAP client 1,
- sniff packets on interfaces.

More details of experiment scenario steps are provided in Appendix I.

2.6 Experiment results

All experiments results can be found in Appendix I.

2.7 Conclusions from validation of datapath protocol programming

The datapath protocol programming experiment showed that new simple data plane protocols could be expressed with usage of P4 schema. The protocols descriptions using that schema can be uploaded to SDN network elements (i.e.: OpenFlow controller and switches) via PAD API methods. Later on, when new requirements will appear, the data plane protocols can be redefined and replaced easily in the network elements. The datapath protocol programming was validated on the basic level by observing, using tcpdump tool, how generated frames were matched and modified by OpenFlow rules containing new defined header fields and new actions. The protocol programming requires less than 5 minutes for the compilation and installation of the code. Thus, it can be done during any maintenance window when the SDN network (or particular devices) can be adapted to new requirements.

Extending xDPd/GNU-Linux software switch with a new protocols by the compilation of generated code files (i.e.: an approach taken in PAD implementation) doesn't have any negative impact on switch frame forwarding performance but this aspect was not actually tested in the experiment.

The used testing scenario for the datapath protocol programming experiment was very simple. The description was provided for the most basic version of Ethernet IEEE 802.3 header and only one ICTP header following Ethernet header. Successful validation of this simple scenario means that the most crucial and development challenging task was achieved and open possibilities for further improvements of PAD implementation. The introduction of more levels of network headers will require additional development work in xDPd/GNU-Linux software switch but seems to be realistic to be accomplished. In the ICTP protocol headers, we have used 1-byte, 2-bytes and 4-bytes fields. Other fields types (i.e.: 1-7 bit(s) fields) are currently not supported. The forwarding function generator is currently very limited and forwarding function declaration can contain only add_header and remove_header primitive methods. Possible improvements of forwarding function generation will require much more efforts. An additional constrain of the current PAD implementation is support of OpenFlow 1.2 only. In order to support OpenFlow 1.3 and further version, we must migrate code generator for newest versions of ROFL and xDPd (version 0.5) in which a lot of changes were made related to experimental matches and action implementation, header parsing and performing actions on a frame.

The analysis of current IP protocols in xDPd and ROFL, performed during the development of protocol generation for those projects, showed that some existing protocols could require additional modification of P4 schema. VLAN tags matching in OpenFlow switches is quite unusual comparing to other protocols (see match field combination required for VLANs in Table 10 of [OF-SPEC-1.2], page 38) which can be problematic in PAD implementation. Another interesting finding is that UDP headers have checksum fields calculated basing on certain fields of lower layer (IP) and UDP header itself. The one possible solution for PAD is that checksum calculation must be provided as one of the forwarding functions and that in that function we will get values of all required fields and apply the correct checksum calculation algorithm available in the form of the primitive action method. Taking into account those findings, we are aware that really full validation of P4 language and PAD implementation can be done when all or almost all protocols which are supported by xDPd and ROFL could be

generated by PAD code generator. For this reasons we can assume that the experiment described in this section is just initial validation of datapath protocol programming idea.

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

3 Implementation of QoS Extensions for OpenFlow

This section provides information about the Quality of Service (QoS) extensions implemented and experimented for ALIEN DOCSIS.

3.1 Experiment overview

ALIEN DOCSIS integration into an OpenFlow scenario was successfully demonstrated in previous demonstrations and experiments [FIA2014, TNC2014, D5.2]. In both conferences functionalities of the proxy were progressively increasing up to the extent that it was able to interact with a standard OpenFlow controller running a learning switch application. However, the QoS aspect was not fully manageable from the proxy, and the provisioning of it was quite static, on behalf of favouring that any OpenFlow standard controller could interact with the proxy without any kind of modification or extension.

The new scenario introduced here represents a new feature implemented into the ALHINP proxy. This new feature allows the controller taking advantage of DOCSIS QoS handling capabilities with a few extensions done to the OpenFlow protocol, thanks to the functionalities provided by ROFL libraries.

The motivation for the development and experimentation of these new extensions is related to the fact that currently, broadly used access networks are based on shared media, which clearly implies dealing with resource optimization issues. One of the most useful ways to optimize the shared media is to apply QoS criteria, that is, to govern the media sharing based in the QoS needs of the users. From an OpenFlow point of view, this means that the Controller has to define how the flows are associated to the QoS levels.

Thus, the experiment presented in this section involved the development and testing of the new OF extensions and features, listed below:

- Queue concept extension: originally, a queue is only associated with an outport. This extension associates it also with an inport, thanks to a new ofp12_queue_prop_src_port property.
- Queue creation extension: which provides the mechanism to create those new queues extended with an associated inport.
- Flowmod SET_QUEUE action implementation: which involves the use of VLAN tags to support QoS mappings to service-flows.

3.2 Experiment goals

The goal of the experiment is to show how the DOCSIS QoS support can be exposed using an OpenFlow interface provided by the ALHINP proxy to the controller, and how this controller can take advantage of DOCSIS network QoS managing capabilities to apply this features to the flows installed by the OpenFlow controller.

Each queue used in the traditional DOCSIS scenario has some properties associated with it, such as the Minimum Reserved Traffic Rate or Maximum Sustained Traffic Rate and even a Traffic Priority; the solution provided here enables handling and configuring those parameters using OF protocol.

3.3 Detailed description of QoS extensions

3.3.1 DOCSIS QoS support

In a DOCSIS access network the concept of service-flow is related to a logical connection between the CMTS (DOCSIS head device) and a cablemodem (tail-end device). Each service-flow can be created with certain properties such as the Minimum Reserved Traffic Rate or Maximum Sustained Traffic Rate. These service-flows are loaded into the cablemodem during its boot-up process, which involves different configuration phases:

- First, the cablemodem configures its RF access.
- After that, it tries to get an IP address using DHCP. Apart from an IP, this protocol enables the cablemodem to get information about a Provisioning System which will be used in the last phase of this configuration procedure.
- Once the IP has been assigned, the cablemodem proceeds synchronizing its date using NTP.
- Finally, the cablemodem contacts the Provisioning System to get the DOCSIS configuration file.

This configuration file obtained from the Provisioning System has several information is encoded, such as the service flows to be created, classifiers and other configuration parameters. Thus, once a cablemodem gets this file, it starts to configure such service flows and classifiers, and after that, it gets the online status.

1) Service-Flows

The configuration of the service-flows is separated in to parts, regarding the properties for upstream and downstream.

In the configuration lines shown below, parameters marked in bold are configured for each service-flow. Service Flow Reference is automatically assigned by the proxy, whereas Minimum Reserved Traffic Rate or Maximum Sustained Traffic Rate is taken from OpenFlow or the default configuration file.

- Upstream Service-Flow Properties

This part of the configuration file defines the behaviour of an upstream service-flow.

```
Upstream Service Flow Encodings
```

```
Service Flow Reference:1
 Quality of Service Parameter Set: provisioned admitted active
 Traffic Priority:5
Upstream Maximum Sustained Traffic Rate:2000000
 Maximum Traffic Burst:8192
Minimum Reserved Traffic Rate:500000
 Maximum Concatenated Burst:8192
 Service Flow Scheduling Type:Best Effort
```

- Downstream Service Flow Properties

The configuration for the downstream service-flow is analogous to the configuration applied to the upstream service-flow.

```
Downstream Service Flow Encodings
Service Flow Reference:2
 Quality of Service Parameter Set: provisioned admitted active
 Traffic Priority:5
Downstream Maximum Sustained Traffic Rate:2000000
 Maximum Traffic Burst:8192
Minimum Reserved Traffic Rate:3000
```

2) Classifiers: Filtering features

Apart from capabilities to manage the bandwidth of the service-flows, the DOCSIS equipment is also able to filter and classify incoming traffic and mapping it into the corresponding service-flow. To perform this filtering action, some classifiers are set in the configuration file. These classifiers can establish an access policy to each service-flow for the flows regarding L2, L3, and even L4 fields. VLAN matching fields can be also supported by the DOCSIS equipment.

Each classifier is linked to a certain service-flow, and its configuration is given in two separated parts also, as it can be noticed below. Either for upstream or downstream, the Classifier Reference number is automatically assigned by the proxy, as the service-flow is the reference that links the classifier with the corresponding service-flow. The VLAN_VID is generated taking into account the identifier assigned to the queue.

- Upstream Classifier Properties

```
Upstream Packet Classification Encoding
Classifier Reference:1
Service Flow Reference:3
 Classifier Activation State:on
 IEEE 802.1P/Q Packet Classification Encodings
   IEEE 802.1P User_Priority:low 0 high 7
IEEE 802.1Q VLAN_ID:0220
```

- Downstream Classifier Properties

```
Downstream Packet Classification Encoding
Classifier Reference:4
Service Flow Reference:6
 Classifier Activation State:on
 IEEE 802.1P/Q Packet Classification Encodings
   IEEE 802.1P User_Priority:low 0 high 7
IEEE 802.1Q VLAN_ID:0220
```

3.3.2 OpenFlow QoS support

OpenFlow gives limited support for features related with QoS. Actually, OpenFlow just can map a flow into a certain outport queue, which is related to a processing priority.

Thus, the concept of an OpenFlow queue has to be reoriented as to be related to the service-flow that is going to be mapped into. This implies that a flow mod with a SET_QUEUE action inside its action set is going to be mapped into a certain DOCSIS service-flow. So, a relation must exist between queue ID and a marker that will be used by the DOCSIS equipment to carry the traffic over a specific service-flow.

3.3.3 OpenFlow QoS extensions for DOCSIS

Now all the elements of the access network have new functionalities to support these extensions. The functionalities are described in Figure 3-1.

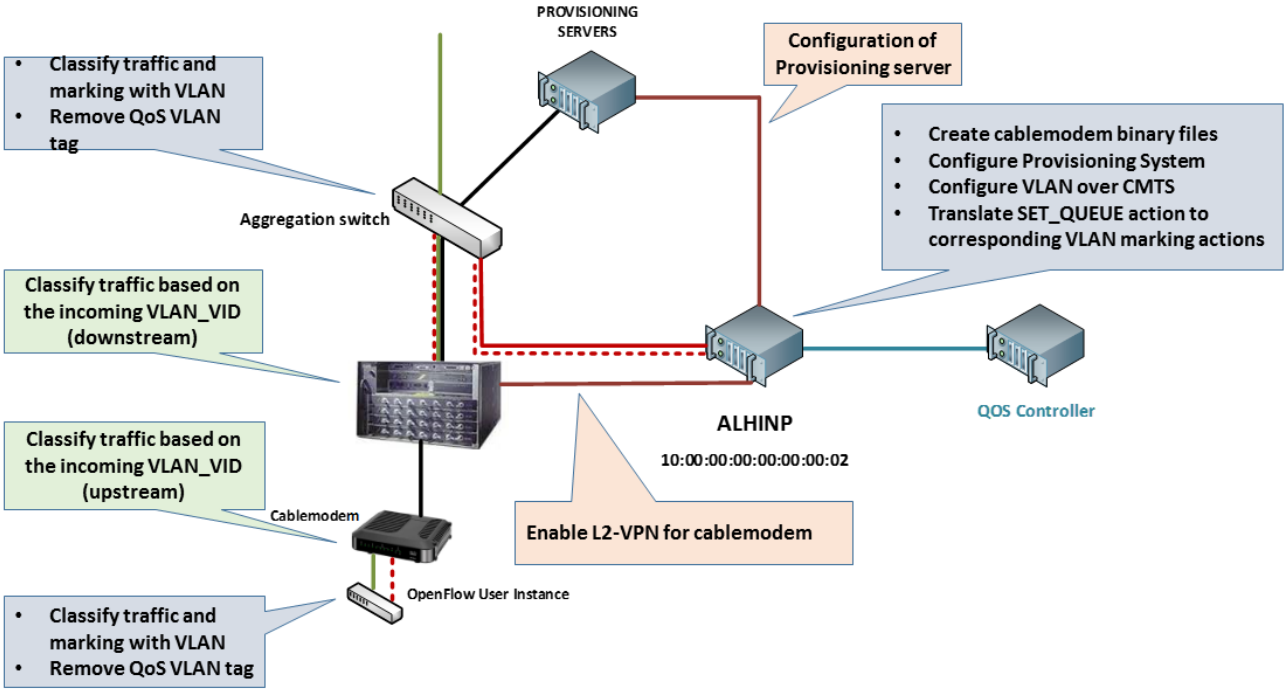


Figure 3-1 Functionalities performed by different elements of the ALIEN DOCSIS

A link between the OpenFlow Queue ID concept and a way to mark traffic understandable by DOCSIS classifiers is necessary. A DOCSIS classifier, as mentioned above, could match the majority of matching fields provided by OpenFlow (1.0); however, mapping the entire matching of a flow mod into a DOCSIS classifier would imply a very restrictive way of matching, as any change or creation of any of the classifiers would require to reboot the cablemodem and would push to it a new configuration binary file.

Extensions done to OpenFlow to support the new QoS features can be grouped into three main aspects of the specification, explained in the next subsections.

1) OpenFlow Queue new properties

As the concept of queue has been reoriented from the original usage provided by OpenFlow, new queue properties have been defined for the DOCSIS scenario. With these properties, queues have meaning between two ports, that is, not only with the egress port but with the inport also. The queue properties defined are:

- OFPQT_MIN_RATE: represents the maximum rate configured into the DOCSIS service-flow (in kbps)

```
/* Min-Rate queue property description. */
 struct ofp_queue_prop_min_rate {
 struct ofp_queue_prop_header prop_header; /* prop: OFPQT_MIN, len: 16. */
 uint16_t rate; /* Maximum allowed rate in kbps. */
uint8_t pad[6]; /* 64-bit alignment */
};
OFP_ASSERT(sizeof(struct ofp_queue_prop_min_rate) == 16);
```

- OFPQT_MAX_RATE: represents the minimum sustained rate configured into the DOCSIS service-flow (in kbps)

```
/* Man-Rate queue property description. */
 struct ofp_queue_prop_max_rate {
 struct ofp_queue_prop_header prop_header; /* prop: OFPQT_MAX, len: 16. */
 uint16_t rate; /*  Minimum sustained rate in kbps. */
uint8_t pad[6]; /* 64-bit alignment */
};
OFP_ASSERT(sizeof(struct ofp_queue_prop_max_rate) == 16);
```

- OFPQT_SRC_PORT: represents the incoming port from where this QoS can be provided. The queue is related to t he egress port, so this parameter means from where this queue is available.

```
struct ofp12_queue_prop_src_port{
    struct ofp12_queue_prop_header prop_header; /* prop: OFPQT_MIN, len: 16. */
    uint32_t src_port;        /* Qos can be applied from certain inputs */
uint8_t pad[4];          /* 64-bit alignment */
};
OFP_ASSERT(sizeof(struct ofp12_queue_prop_src_port) == 16);
```

```
struct ofp10_queue_prop_src_port{
    struct ofp10_queue_prop_header prop_header; /* prop: OFPQT_MIN, len: 16. */
    uint16_t src_port;        /* Qos can be applied from certain inputs */
uint8_t pad[6];          /* 64-bit alignment */
};
OFP_ASSERT(sizeof(struct ofp12_queue_prop_src_port) == 16);
```

2) New OpenFlow message: SET_NEW_QUEUE

When a controller wants to apply a new QoS schema to a port which is already configured, a new SET_NEW_QUEUES experimental message has to be used. In this message the new QoS schema is set by the controller. When the ALHINP proxy receives this message, internally a new set of service-flows and classifiers is pushed to the Provisioning System, which begins a procedure that ends up with the correct configuration of the new set of service-flows in the cablemodem.

The SET_NEW_QUEUE message is an asynchronous message sent by the controller to the proxy to set the new QoS schema to a port, and indeed to the cablemodem. OpenFlow and default queues are always defined by the configuration file of the proxy, and only the custom queues can be defined by the controller. The body of the message is as described below.

```
struct set_new queues{
struct ofp_header header;
uint32_t dst_port; /* port which queues are applied to */
uint8_t pad [4];
struct ofp_packet_queue queues[0]; /* List of queues to be created for the
   specified port */
};

OFP_ASSERT(sizeof(struct ofp_struct_ofp_set_new_queues) ==16));
```

When the message can be processed correctly, the proxy answers with a GET_QUEUE_CONFIG_REPLY with the new status of the queues available. If not, an ERROR message is generated, with OFPET_QUEUE_OP_FAILED reason.

3) OpenFlow SET_QUEUE action implementation

Including SET_QUEUE in the action set of a flow mod is the way that a controller can use QoS features implemented by the ALIEN DOCSIS to apply bandwidth usage properties to a certain flow.

Marking traffic with a specific VLAN tag gives more flexibility to the solution. Instead of pushing the entire matching field set to the classifier, a VLAN identifier is assigned for each service-flow classifier. Then, the traffic classification and filtering functionality will be delegated to ingress/egress OpenFlow elements (OpenFlow user Instance and Aggregation switch). Besides, as the QoS application mechanism is generic (pushing a VLAN tag) any type of Ethernet traffic could be suitable to be carried with guaranteed QoS requirements.

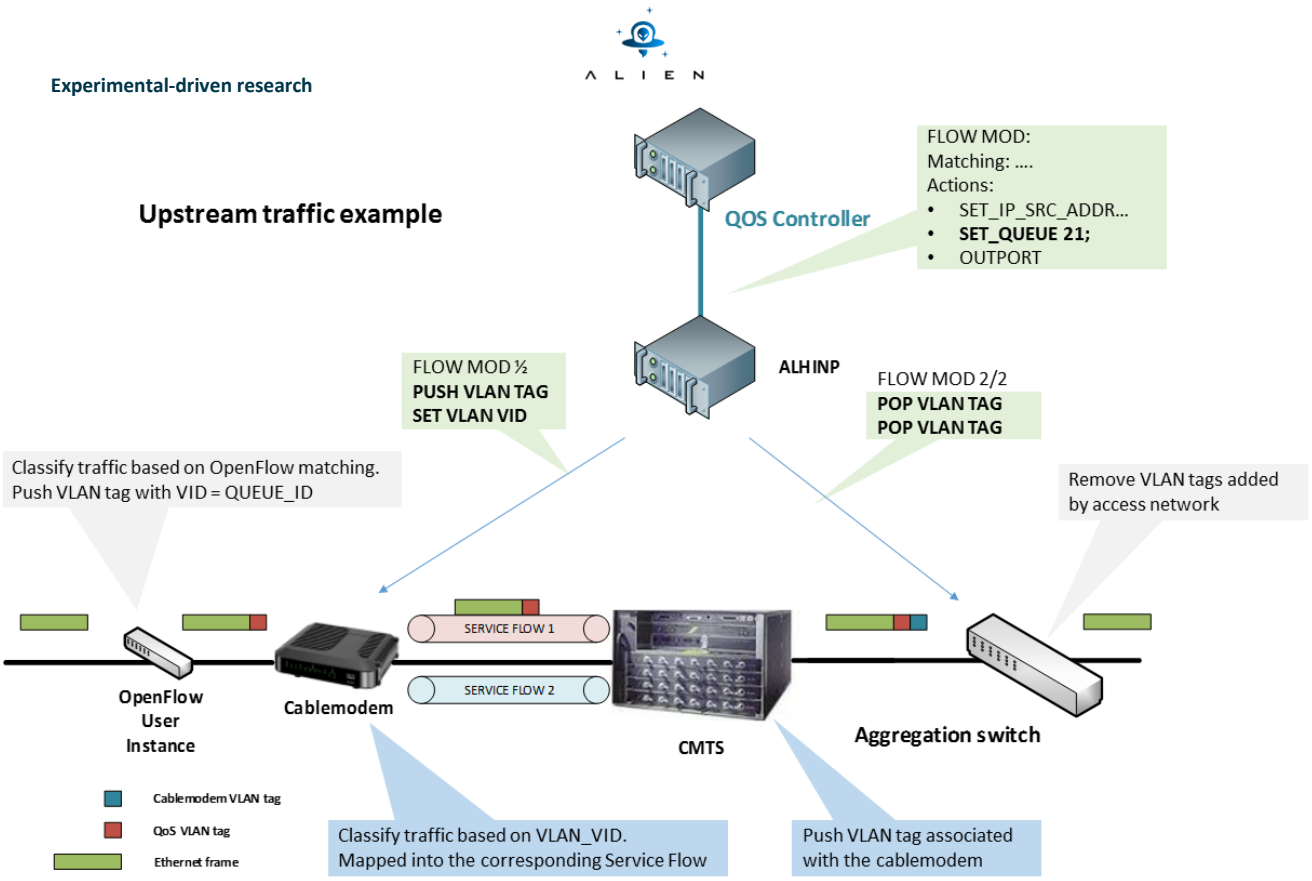


Figure 3-2 Flow mod processing

In the Figure 3.2: Flow mod processing is depicted the SET_QUEUE action processing by the ALHINP proxy. When a Flow mod with SET_QUEUE present in its action set is received, the proxy gathers the VLAN assigned to that identifier, and sends the corresponding partial flow mods to the OpenFlow devices to add or remove the VLAN tag assigned to that identifier. DOCSIS equipment applies the bandwidth requirements.

3.4 Experiment environment

To test the extensions developed with the ALIEN DOCSIS, a new OFELIA setup was created as depicted in Figure 3-3.

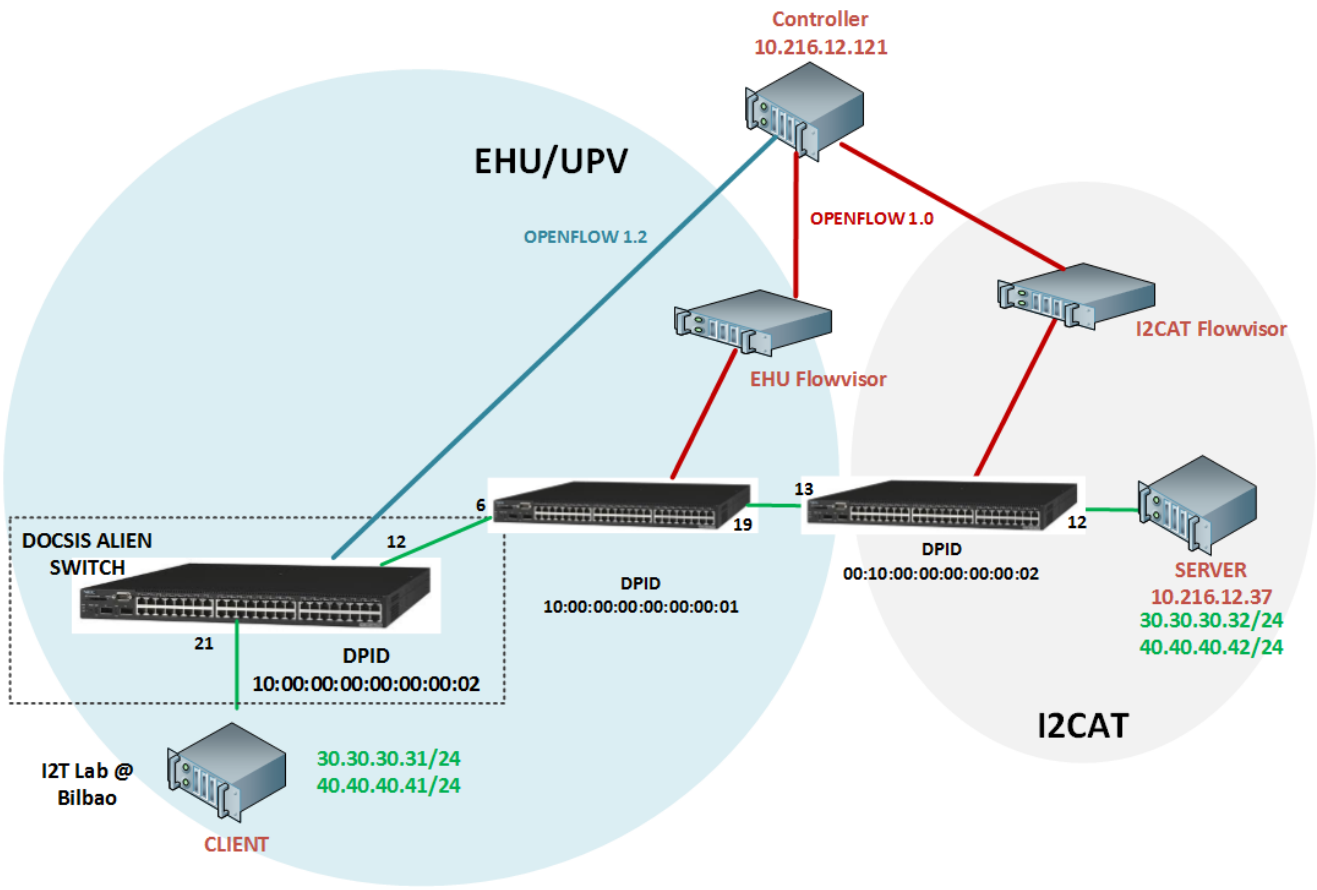


Figure 3-3 OFELIA setup for testing QoS extensions

A controller was deployed in I2CAT Island which is able to use either 1.0 or 1.2 version of the OpenFlow protocol. As the interaction with the ALIEN DOCSIS device is done using OpenFlow 1.2 version, this connection cannot be done through a Flowvisor. For this reason, it is done using a direct link via control plane.

The controller installs flow mods into switches that connect the client and the server. When interacting with ALIEN DOCSIS, it is also capable to handle QoS new developed features.

For the client and the server, an interface is used with different IP ranges. This way, different L3-based flows can be configured and easily tested.

3.5 Testing procedure overview

The testing carried out for the demonstration of the OpenFlow QoS extensions developed is described in the next workflow. It describes how the proxy works and the interaction between OpenFlow and DOCSIS to provide the abstraction of the QoS mechanisms.

OpenFlow QoS extension testing is composed of a few phases:

- Step 0: Loading default QoS profiles
- Step 1: Network boot up process
- Step 2: Controller, ask for queues
- Step 3:Default QoS usage
- Step 4: Enforcing new queues
- Step 5: Using new queues
- Step 6: Checking the bandwidth usage

Step 0: ALIEN DOCSIS default configuration related to QoS is defined in a new configuration file. In the first step, this configuration file is parsed by ALHINP (ALien HAL-based Integration Network Proxy). Proxy reads this file and generates a default profile of service flows to be loaded into cablemodem.

Step 1: When a new cablemodem is connected and a RF interface is configured and synchronized, a DHCP request is sent by the cablemodem (step 2). This request packet matches a specific flow rule in the aggregation switch and the action set for this matching is to send the packet to controller, (ALHINP proxy). Once the packet arrives the ALHINP, the cablemodem mac address is gathered. Then a VLAN is assigned to the cablemodem and this configuration is pushed to the CMTS. ALHINP checks if there is any specific QoS string for it. If not, default queue schema is compiled for it.

Step2: Once the cablemodem is configured and the service-flows and classifiers are provisioned, the OpenFlow User Instance, the helper located at the user side, will try to contact ALHINP. After this connection is established, the ports will be reported to the controller as available by using a PORT_STATUS message. However, in this message, information about queues available for this device cannot be reported. Therefore, the controller should ask for the queues available for the new ports sending a QUEUE_GET_CONFIG_REQUEST.

Step3: The controller installs flow mods to enable 2 different traffics, based on IP fields. The first flow is performed without SET_QUEUE action and the second one using SET_QUEUE and the corresponding identifier. As the only queue configured is the default, the bandwidth usage for both traffics will be the same.

Step 4: The creation of a new queue is requested, using an experimenter message for it. The properties of the new queue are 20Mb/3Mb from the port 12 (aggregation port) and port 21 (a client port) where the greater bandwidth is related to the network to user direction.

Step 5: A new profile is pushed to the provisioning and then applied to the cablemodem. Once the service flows are provisioned, a new identifier is assigned to this queue and reported to the controller via OFP_GET_QUEUE_CONFIG_REPLY message where the new properties and the identifier are detailed. From this moment, this QoS is enabled for being used by the controller.

Step 6: Now, the new queue is enabled and the controller overrides the rule for traffic from 40.40.40.40 by setting the new queue for it, this is, adding SET_QUEUE action with the new identifier.

Step 7: In order to test the application of new queues, an Iperf test is performed between IPs from the range of 30.30.30.0/24 and then for IPs from 40.40.40.0/24. According to the rules installed and queue numbering default queue is used for 30.30.30.30/24 whereas the high bandwidth one is used for 40.40.40.0/24 IPs. The bandwidth obtained is limited by DOCSIS access network.

More details of experiment scenario steps are provided in Appendix II.

3.6 Experiment results

All experiments results can be found in Appendix II.

3.7 Conclusions from validation of the DOCSIS QoS extensions for OpenFlow

Development and implementation of QoS OpenFlow extensions for DOCSIS has supposed a new improvement for the virtualization of the access network as an OpenFlow device. With the QoS extensions bandwidth managing features of the DOCSIS devices is now exposed to an OpenFlow controller. Furthermore, the capability of creation new DOCSIS service-flows directly and transparently from the controller

Implemented features are grouped into three main aspects. The firs point is the reuse of the concept of an OpenFlow queue. This has been oriented to bandwidth terms instead of processing priorities. With it, new properties have been also developed to expose other aspects that are needed. The second point is the SET_QUEUE implementation, as it is the mechanism that enables to apply QoS requirements to a specific flow from the controller. The third main point of the development is the OpenFlow SET_NEW_QUEUE message which enables to a controller creating a new queue with certain properties described in the message.

Experiment performed demonstrates how a controller could make use of the different types of QoS services available in the DOCSIS access network. Furthermore, it can also create new queues and use them through the provided extensions. This implies that the controller has this extensions implemented into its OpenFlow endpoint. Otherwise, it is only able to use the default queue settings. In order to test it, these extensions have been implemented and tested for the controller used in the validation.

4 TBAM integration and demonstration with Multicast and IPv6 traffic experiments

4.1 Experiment overview

One of the outcomes of the ALIEN project is the so-called ALIEN Control Framework (ACF). The main motivation behind the ACF is to extend the current OFELIA Control Framework (OCF) [OFELIA] in order to allow the OFELIA users to request and add ALIEN resources (basically HAL-enabled switches) to their experiments from a single platform. In fact, the current OCF leverages on FlowVisor [FlowVisor] to allow the sharing of network resources among different experiments. On the other hand, the ALIEN hardware can implement OpenFlow version 1.2 or 1.3 and some particular extensions while FlowVisor only supports version 1.0 of the protocol. In order to allow the experimentation with any version of the protocol (even with extensions outside the standard), the ACF has been designed to avoid the inspection of the OpenFlow protocol by replacing FlowVisor with a TCP proxy that forwards the control messages to the user's controller without any flowspace slicing operation.

With this approach only one experiment at time is allowed and the management of the time-slots booking process is managed by a new aggregate manager called Time-Based Aggregate Manager (TBAM) and configured by the users via the Expedient [Expedient]. Although this approach does not allow multiple simultaneous experiments, on the other hand, the user has exclusive access to the devices, therefore he/she is allowed to reconfigure or re-program them with new features or extension of the control and management protocol.

In this Chapter we demonstrate how the integration of an ALIEN island (a testbed of ALIEN devices) within the OFELIA facility is achieved and we validate this integration by reporting the results of two experiments performed on ALIEN and OFELIA devices. The first shows how it is possible to configure and run an inter-island experiment such as an IPv4 multicast video streaming service. The second, an IPv6 multicast video streaming service, aims to show how experiments leveraging on versions > 1.0 of the OpenFlow protocol can be performed on the ALIEN devices without interfering with the legacy part of the OFELIA facility.

Multicast is a bandwidth-conserving technology that reduces traffic by simultaneously sending streams of information only to the clients that are interested to receive it. The Multicast technology involves multicast addresses and groups. The latter are used by the clients to subscribe to a particular data stream that is made available by a Multicast server. Clients that are interested in receiving data of a particular group must join the group by signalling. A single destination address is used by the Multicast server to reach all members of a group.

4.2 Experiment goals

The purpose of these experiments is to validate the ACF's components functions and to show how the ACF allows the configuration and the execution of inter-island experiments involving ALIEN devices. In particular the outcome of these experiments is twofold:

- Showing how an "island" of ALIEN devices can be integrated within the OFELIA facility. The integration is achieved with the same procedure needed for the connection of a standard OFELIA island to the facility: (i) the connection of the ALIEN island to the OFELIA facility through both data and management planes, (ii) the installation of all ACF's components on a dedicated server and (iii) the connection of the ACF's Expedient to the OpenFlow aggregate managers (Opt-In) running on the OFELIA islands (CREATE-NET, i2CAT, TUB, ETHZ, iMinds etc.). In particular, during an IPv4 multicast experiment spanning the CREATE-NET, iMinds and i2CAT OFELIA islands and the CREATE-NET ALIEN island will be configured and executed.
- Demonstrating that with the ACF and the ALIEN devices is possible to run experiments controlled through versions of the OpenFlow protocol beyond v1.0. In particular, an IPv6 multicast experiment will be performed and controlled with a controller using the version v1.3 of the OpenFlow protocol. Since the legacy OFELIA facility only provides support for v1.0 of the protocol, this experiment will be confined within the ALIEN Island.

To summarize, these experiments will allow the validation of the two main goals of the ACF: (i) the management of experiments on the ALIEN devices and (ii) the compatibility of the ACF software with the OFELIA OCF so that an island of ALIEN devices can be integrated with the OFELIA facility.

4.3 Experiment environment

The ALIEN Island, physically located at the CREATE-NET's premises, has been connected to the OFELIA facility through both data and management planes. Referring to the Figure 4-1 below, the data plane connection between the ALIEN Island and the OFELIA facility goes through the OpenFlow GateWay (OFGW). As explained in Deliverable [D4.2], this operation is necessary to tag the traffic exiting the ALIEN island with the OFELIA VLAN ID assigned to the experiment in order to allow the FlowVisor instances running on the OFELIA islands to perform the flowspace slicing (which is based on the VLAN ID).

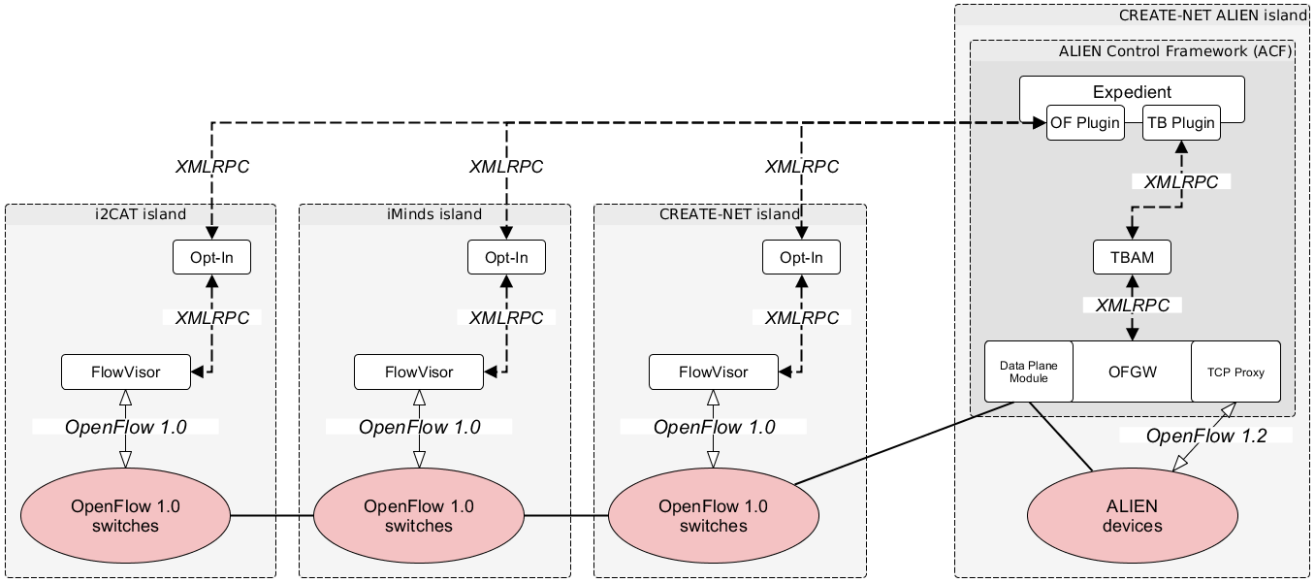


Figure 4-1 Logical architecture of the Control Framework

The ACF has been installed and configured to connect to the OFELIA central LDAP server located at the EICT premises. The connection to the LDAP allows the OFELIA users to use the same credentials to access the ALIEN resources through the ACF. The OpenFlow Plugin for Expedient (OF Plugin in Figure 4-1) has been connected via a XMLRPC connection to the aggregate managers of the standard OFELIA islands (the Opt-In). With this connection the resources of the OFELIA islands are visible and accessible on the ACF's Expedient GUI and can be used for experimentation.

Finally, the ALIEN switches (commodity PCs running the HAL) have been configured to use the version v1.2 of the OpenFlow protocol in order to support the IPv6 traffic. For this reason, the implementation of the experiments requires the employment of two different OpenFlow controllers: the first supporting and running the OpenFlow v1.0 to control the OFELIA devices, the second running and supporting the v1.2 of the OpenFlow protocol to control the ALIEN devices. For our experiments we use [FloodLight] and [Ryu] respectively (see Figure 4-2), with multicast video distribution applications running on top of them.

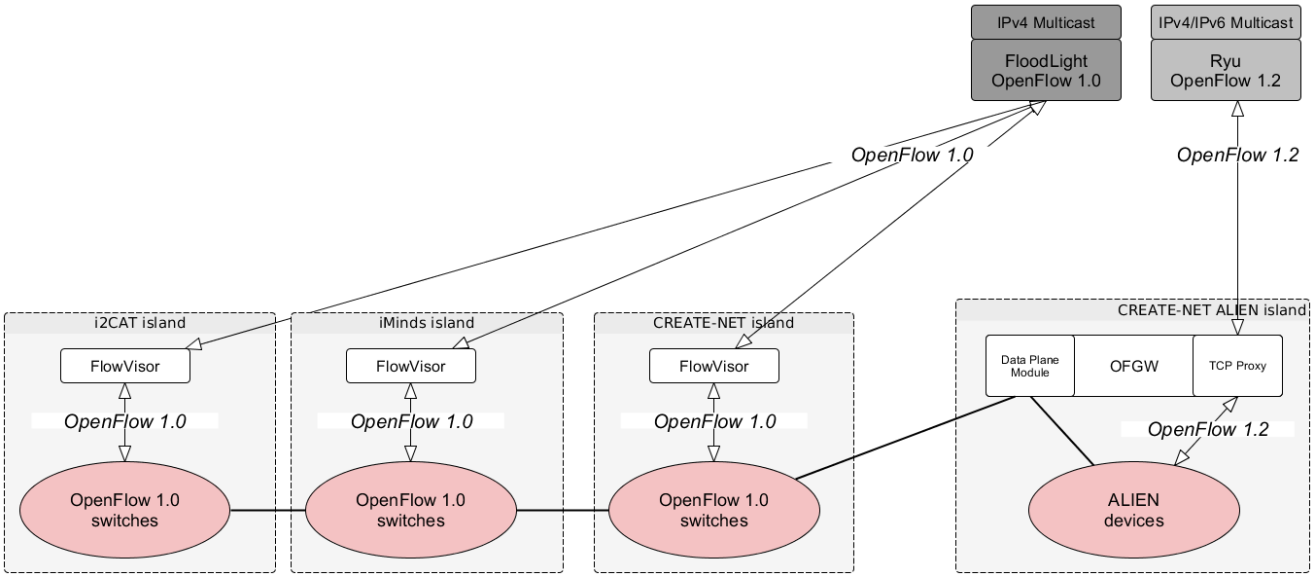


Figure 4-2 FloodLight and Ryu frameworks were used to build the multicast applications for the two experiments

4.4 Testing procedure overview

- ALIEN island (4 ALIEN switches) located at the CREATE-NET premises and connected to the CREATE-NET's OFELIA island
- ACF installed to manage the ALIEN switches and connected to the OCF instances running on 5 OFELIA islands
- Slice composed of switches of 3 OFELIA islands (CREATE-NET, i2CAT and iMinds) plus the ALIEN island
- Two different controllers: FloodLight for the OFELIA switches using OpenFlow v1.0 and Ryu for the ALIEN switches using OpenFlow v1.2
- IPv4 and IPv6 multicast applications running on top of FloodLight and Ryu
- VLC [VLC] software running on the virtual machines and acting as multicast server and client

More details of experiment scenario steps are provided in Appendix III.

4.5 Experiment results

All experiments results can be found in Appendix III.

4.6 Conclusions from validation of the TBAM integration and demonstration with Multicast and IPv6 traffic experiments

The implementation of multicast video streaming experiments was motivated by the need of validating the ALIEN Control Framework components that have been developed for the reasons explained in the previous sections and that basically

are: (i) the orchestration of experiments running on ALIEN devices and (ii) for the integration of these devices within the OFELIA facility.

The two experiments, IPv4 and IPv6 multicast video streaming, have proven that the ACF can be used to orchestrate the experiments on ALIEN devices and that the ACF can be connected to the OCF instances of the standard OFELIA islands in order to configure multi-island experiments.

Moreover, the IPv4 experiment, demonstrated that it is possible to run multi-island experiments with different controllers using different versions of the OpenFlow protocol (even versions outside the standard as envisioned for ALIEN devices).

Finally, with the IPv6 multicast experiment, we demonstrated that the users can leverage on the ALIEN devices to test different versions of the protocol without interfering or disrupting the experiments running on the legacy OFELIA devices. In fact, only the traffic tagged with a specific VLAN ID (configured by the user through the TB Plugin for Expedient as shown in Figure 1-7 of Appendix III) is forwarded to the OFELIA islands. The other traffic is dropped by the OFGW when attempting to cross the border between ALIEN and OFELIA, therefore, experiments using OpenFlow versions > 1.0 or using custom version of the protocol (even outside the standard), can be confined within the ALIEN islands.

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

5 Performance Experiments

5.1 Experiments overview

The ALIEN platforms performance tests are based on OFLOPS application [OFLOPS], [OFLOPS1]. The OFLOPS is a tool that allows developing use-case tests for both hardware and software OpenFlow implementations. Using this tool, developers can define and simulate specific usage scenario and understand the impact of switch implementation on performance. The OFLOPS platform is implemented as multi-thread application without any concurrent access control. It results in reduced latency. The OFLOPS consists of the following five threads, each one serving specific type of events (Figure 5-1):

- 1: Packet generator - controls data plane traffic generators,
- 2: Packet capture/analyzer - captures and pushes data plane traffic to modules,
- 3: Message generator - translate OpenFlow packets to control events or generates OpenFlow messages,
- 4: SNMP channel - perform asynchronous SNMP polling (because selected HAL implementation does not support SNMP protocol, in prepared tests this module is switched off),
- 5: Action scheduler - manages time events scheduled by measurement modules.

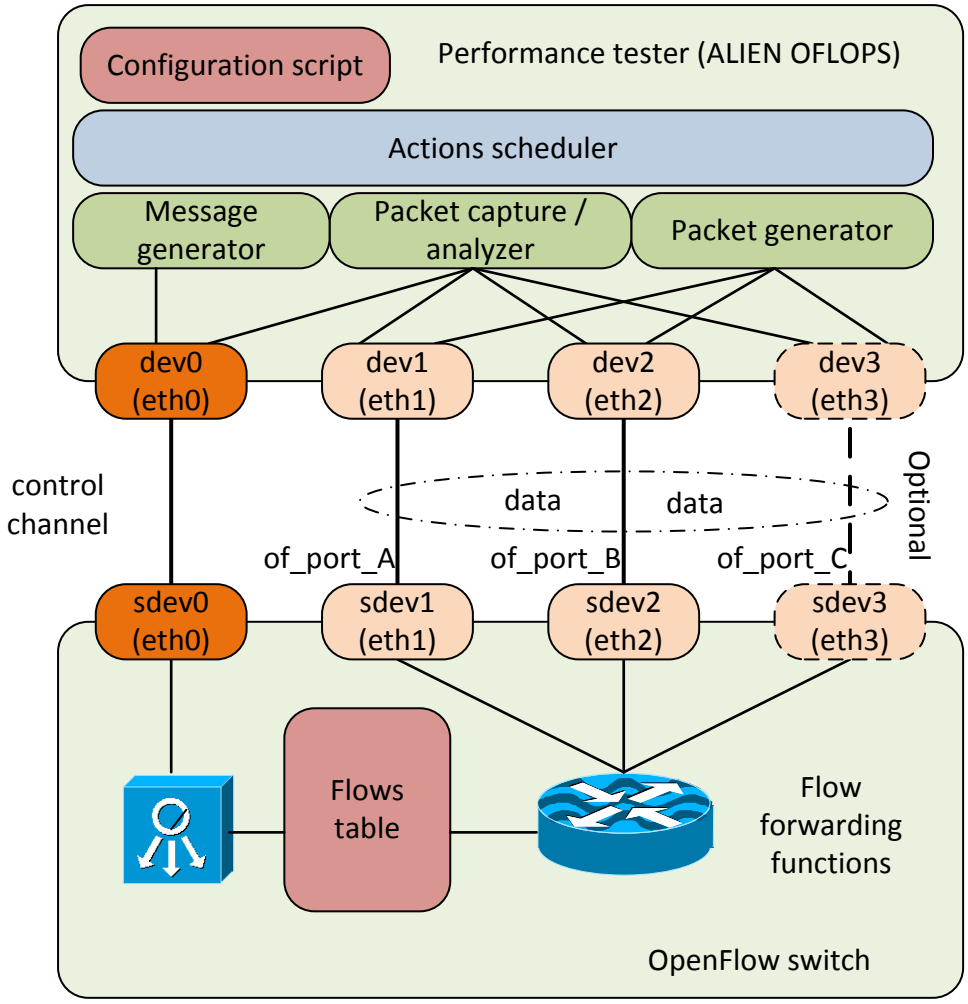


Figure 5-1 Hardware configuration for experiments

The OFLOPS application offers many ready to use experiment scenarios. Some of these scenarios are adapted to performance evaluation of the OpenFlow implementation on ALIEN platforms. These scenarios and modified application will be referred to later in this document as ALIEN OFLOPS.

In order to avoid any additional delay caused by other network devices, the host with installed ALIEN OFLOPS application is connected directly with tested platform. The minimal number of required interfaces is equal to two, one for control channel and one for data path. Unfortunately, in this configuration only two test scenarios can be realized. In order to realize full set of prepared test scenarios two interfaces in data plane are required. The OFLOPS application allows using also the third data plane interface, but for our purpose, this interface is superfluous.

The test scenarios are implemented as runtime linked modules written in C++ code. The parameters of each experiment are configured using the following configuration script:

The OFLOPS configuration file:

```
oflops: {
```

```
    control: {
   control_dev = "eth0";
   control_port = 6633;
   snmp_addr = "10.1.0.2";
   cpu_mib="1.3.6.1.2.1.25.3.3.1.2.768";
   in_mib="1.3.6.1.2.1.2.2.1.11.2";
   out_mib="1.3.6.1.2.1.2.2.1.17.2";
   snmp_community = "public";
};

data = ({
     dev="eth1";
     port_num=8;
     in_snmp_mib="1.3.6.1.2.1.2.2.1.11.3";
     out_snmp_mib="1.3.6.1.2.1.2.2.1.17.3";
},{
     dev="eth2";
     port_num=7;
     in_snmp_mib="1.3.6.1.2.1.2.2.1.11.4";
     out_snmp_mib="1.3.6.1.2.1.2.2.1.17.4";
},{
     dev="eth3";
     port_num=6;
     in_snmp_mib="1.3.6.1.2.1.2.2.1.11.5";
     out_snmp_mib="1.3.6.1.2.1.2.2.1.17.5";
});

traffic_generator = 1;
dump_control_channel=0;

module: ({
   path="/home/mariusz/alien/std1/oflops/example_modules/openflow_add_flow/.lib
   s/libopenflow_add_flow.so";
   param="flows=1    data_rate=500    probe_rate=500    pkt_size=500    table=0
   probe_time=120 echo_rate=4";
});
};
```

Experiment parameters:

- dev - the name of the network device used by the channel,
- port num -the ID of the port of the OpenFlow switch to which the network device is connected,
- SNMP OIDs - not required for prepared experiments,
- flows - the number of flows in flow_add and flow_mod tests [/],
- data_rate - the rate of the measurement probe [Mbps],
- probe_rate - the rate of the constant probe [Mbps],

- pkt-size - the length of UDP packets of the measurement probe [B],
- table - define whether the inserted flow will be wildcard (1) or exact match (0),
- probe _time - the experiment duration time [s],
- echo_rate - interval time between OFPT_ECHO_REQ.

5.2 Experiments goals

The prepared performance tests are designed in order to show that ALIEN platforms with implemented HAL have the performance comparable with existing commercial and opensource OpenFlow switch implementation. We want to show, that ALIEN platforms are capable to act as native OpenFlow implementations. As a reference point two implementations are considered: OpenVSwitch and OpenFlow NetFPGA implementation. Since particular ALIEN platforms are quite different, it is impossible prepare common testing environment. Moreover, for selected platforms, such as L0switch, performance tests have no reference points or time required by actions realized in HAL are at least one order of magnitude smaller than time of actions realized in Alien hardware.

5.3 Experiments environment

Performance evaluation of HAL implementation in NetFPGA and EZappliance platforms ware realized in PUT OFELIA Island. Since NetFPGA and EZappliance implementations are quite different, two OFELIA configurations were used. Both configurations are presented in Figure 5 2 and Figure 5 3. In order to avoid any additional delay, connections in the data plane ware set up directly, without any additional devices. Connections in the control plane required additional switch, but in this case, the delay introduced by Ethernet switch was very low and constant, so it can be omitted. Results of the performance test for the NetFPGA and EZappliance platforms were compared with the results obtained for the OpenVSwitch (OVS) OpenFlow switch implementation installed as a virtual machine (see Figure 5 4).

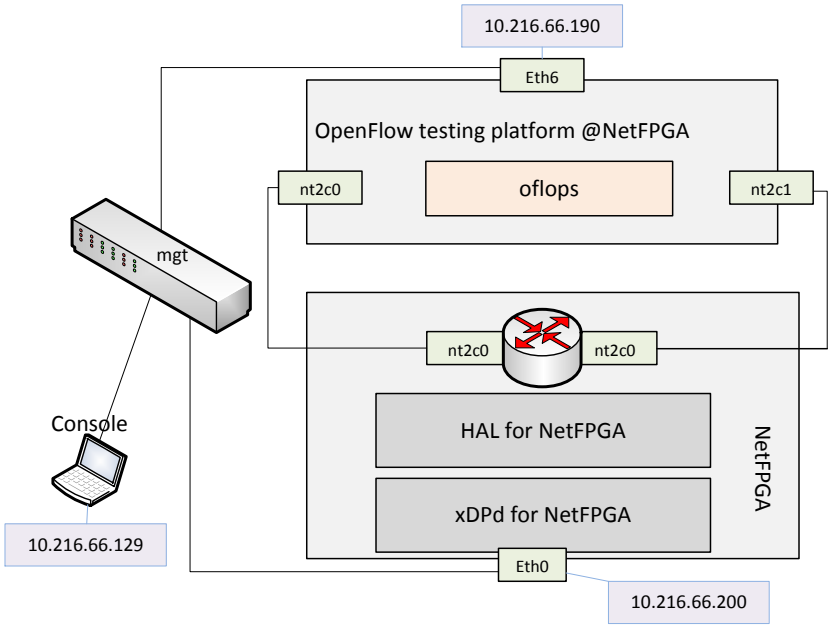


Figure 5-2 OFELIA setup for NetFPGA platform performance testing

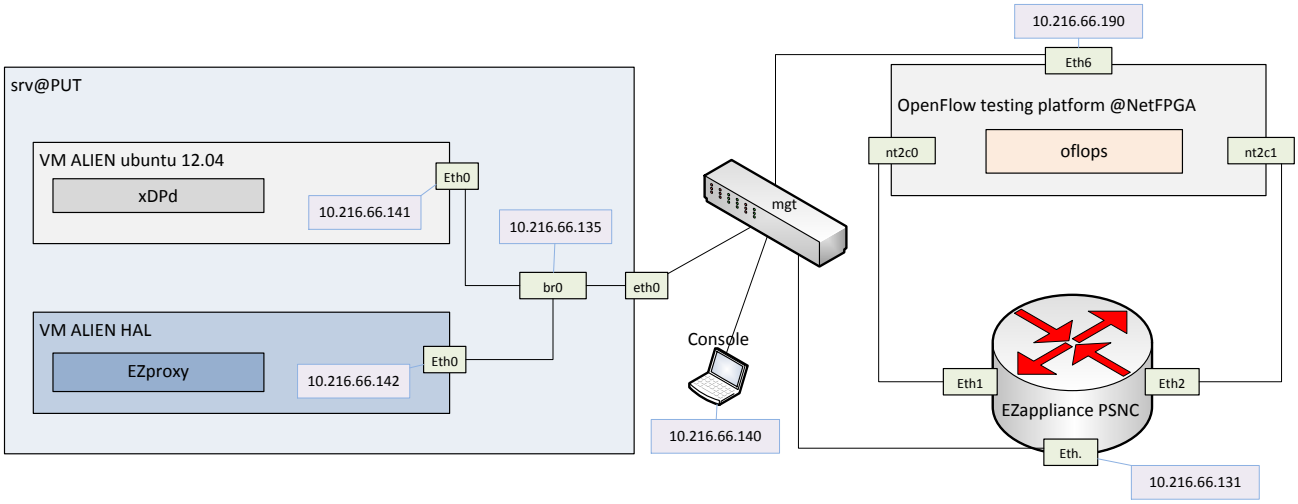


Figure 5-3 OFELIA setup for EZappliance platform performance testing

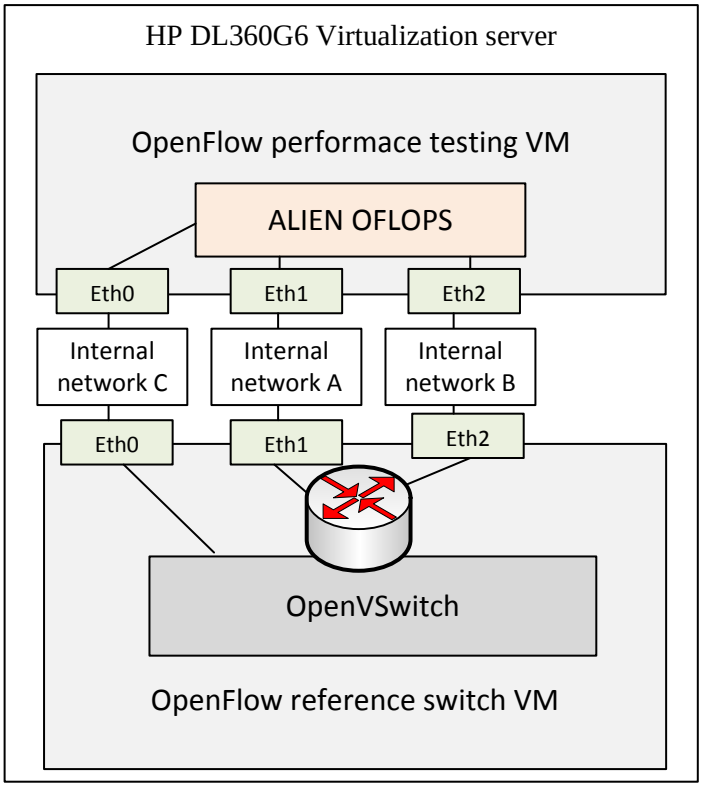


Figure 5-4 Implementation of OpenFlow reference switch performance testing environment

5.4 Testing procedure overview

During the OpenFlow performance testing, the following set of tests is executed for NetFPGA and EZappliance HAL prototypes:

- PacketIn test measuring delay of Packet_in action,
- PacketOut test measuring delay of Packet_out action,
- Path_delay test measuring a single packet delay both in control and data plane,
- Flow_add test measuring time required to add the given number of flows,
- Flow_mod test measuring time required to modify the given number of table flow entries.

More details of experiment scenario steps are provided in Appendix IV.

5.5 Experiment results

All experiments results can be found in Appendix IV.

5.6 Conclusions from performance experiments

5.6.1 NetFPGA

The realized tests allowed us to investigate time and performance characteristics of OpenFlow implementation on NetFPGA cards. They were realized by software packet generator and analyzer, so they have some disadvantageous and bottlenecks can appear (because tester has lower "quality" than tested element). It was possible to use hardware tester based on OFLOPS which can be run on NetFPGA chip or professional network tester, but we decided to use resources which are available for each project partner, which allows realizing the same or very similar procedure for each platform.

The main assumption was to test separately performance of particular functionality. In real life there is no clearly analogical situations, there is no stream with only Packet_In, Packet_Out (except attacks or another abnormal situations). The most realistic situation is the case when all flows from stream are served in hardware part (in something like steady-state of switch and network). In real life, all tested situation appear together. It is possible to prepare tests representing such a situations but it requires a lot of scientific effort to gather statistics of traffic and prepare its model. Such a work is very interesting but out of scope of this deliverable.

Another area, where tests can be little improved, it is their automation and granularity.

5.6.2 EZappliance

Oflops as a counterpart of the oftest platform allowed us to test HAL-enabled EZappliance platform. Next to functional tests of OpenFlow implementation over NP-3 network processor showed in deliverable D3.3 this chapter presents performance tests of OpenFlow-capable alien hardware platform based on NPU. Most of the results are compared with OVS implementation. Due to prototype implementation of Hardware Abstraction Layer using separate virtual machines for xDPd and EZproxy deployment as well as Corba technology used to interconnect software modules the presented results are satisfactory but worse (especially for control channel) than in case of OCV implementation. For the performance tests xDPd was compiled with the maximum level of optimization and logging was disabled on EZproxy.

During the tests, we observed how different test conditions affect performance of data and control channels. We measured delays of these channels as well as total system throughput. Charts showed the thresholds of full throughput performance and parameters at which it becomes less than one (not all frames sent are received). For the fixed probe rate (in Mbps) the smaller size of the packet means more packets generated during the test (usually 30, 60 and 120 seconds).

Results of packet_in test together with packet_out test shows responsiveness of the system for a new (unknown) flows. As it is expected, for the huge amount of generated events, an event handling time becomes higher. For 1000 packet_in events generated during one minute, it takes over 10 seconds. For some test conditions it was not possible to make a reliable packet_out tests (too big delay). The data path delay tests showed that data plane packets switching time for EZappliance device depends on frame length. The shortest measured frames (100 bytes) are switched in ~30 µs whereas the longest (1500 bytes) are processed in ~120 µs. These delays are really small what can be explained by high performance and high throughput of EZchip NP-3 NPU - 30Gbps in total. In case of control channel delay test, the measured mean delay is about 394 µs. This is a time required to xDPd and ROFL software to generate OpenFlow replay. This pure software processing make the results quite similar to result achieved for OVS. For add_flows we observed that flow installation in

EZappliance is much slower (more than two orders of magnitude) in comparison to OVS software switch. Such difference is caused by fact that EZappliance is storing flow entries in TCAM memory, which is very slow for writing operations whereas OVS is using fast DRAM memory. The single flow installation delay in EZappliance is increasing when more flows have to be installed for less than 70 flows and then this value fluctuating between 40-50 msec per flow. The more flows are requested to be installed the longer delay in installation of single flow. It is because of abovementioned speed of TCAM memory.

From the oflops framework point of view performance tests of HAL-enabled EZappliance platform was done without distortions. The control plane message exchange between testing tool and tested platform does not evidence unexpected reactions from EZappliance xDPd. During the whole experiment, time data path acts in accordance with expectation. Only small modification in oflops scripts was done for add_flow_max_number test. When the number of flow table entries grows, action installation time increases exponentially, so in case of experiments with the number of installed flows greater than 300, time restrictions of testing application (probe_time parameter) were disabled and only maximum number of installed flows was determined.

In most tests of EZappliance we gathered worse results than with OVS implementation of OpenFlow switch. It is worth to mention that HAL-enabled EZappliance implementation wasn't designed to achieve good performance results. In fact HAL-enabled EZappliance is only the prototype with software components installed on separate VMs. For sure migration of the software components from separate VMs to Host CPU board should improve performance of OpenFlow-capable EZappliance platform based on NP-3 network processor. From the other hand EZappliance device is just the evaluation platform with poor performance of Host CPU board with PowerPC architecture. For that reason we decided to adopt the HAL architecture and Hardware Specific Parts for EZappliance platform as a set of three software packages: xDPd for EZappliance, EZ Proxy and Hardware Datapath on three separate VMs which also easier code maintenance and development process in the team.

6 Multi-vendor evaluation of HAL in the optical domain

6.1 Experiment overview

Software Defined Networking (SDN) is an emerging solution to realise less complexity and more flexibility in today's networks. In order to achieve that, SDN proposes the separation of control plane from the data plane. The control of the network device migrates to a logically centralized location and the network entity becomes essentially a forwarding device of the network. A well-defined southbound API is required to allow controlling the device. While other southbound protocols, like ForCES and OpFlex exist, OpenFlow has become the de-facto protocol supported by many SDN vendors.

In an OpenFlow-enabled packet switch when a packet arrives at the device, it will be examined and if a matching rule exists in the flow table, then it will be forwarded to the right port. Otherwise, the packet will be forwarded to the controller which will make the decision on forwarding this packet. This process requires matching the fields of the packet in order to perform a number of actions on it. Many network equipment vendors already support OpenFlow as a standard to allow network operations to control their devices.

In contrast to that, in the optical domain there is no notion of packets and the switch has no visibility in the payload. Instead of matching packets to assign to flows, optical switches are cross-connecting ports inside the switch to redirect traffic from one port to another. This already creates a separation between the control and the forwarding (data) plane of the switch. However, due to the diversity of functions and different implementation technologies used by optical switch vendors the network operator needs to configure each device separately using the vendor's proprietary interface. In Figure 6-1 below we are showing an example of different vendor's implementation for installing cross-connections. OpenFlow can be used to facilitate this process and allow unified control for all the nodes across the network irrespective of domain and vendor.

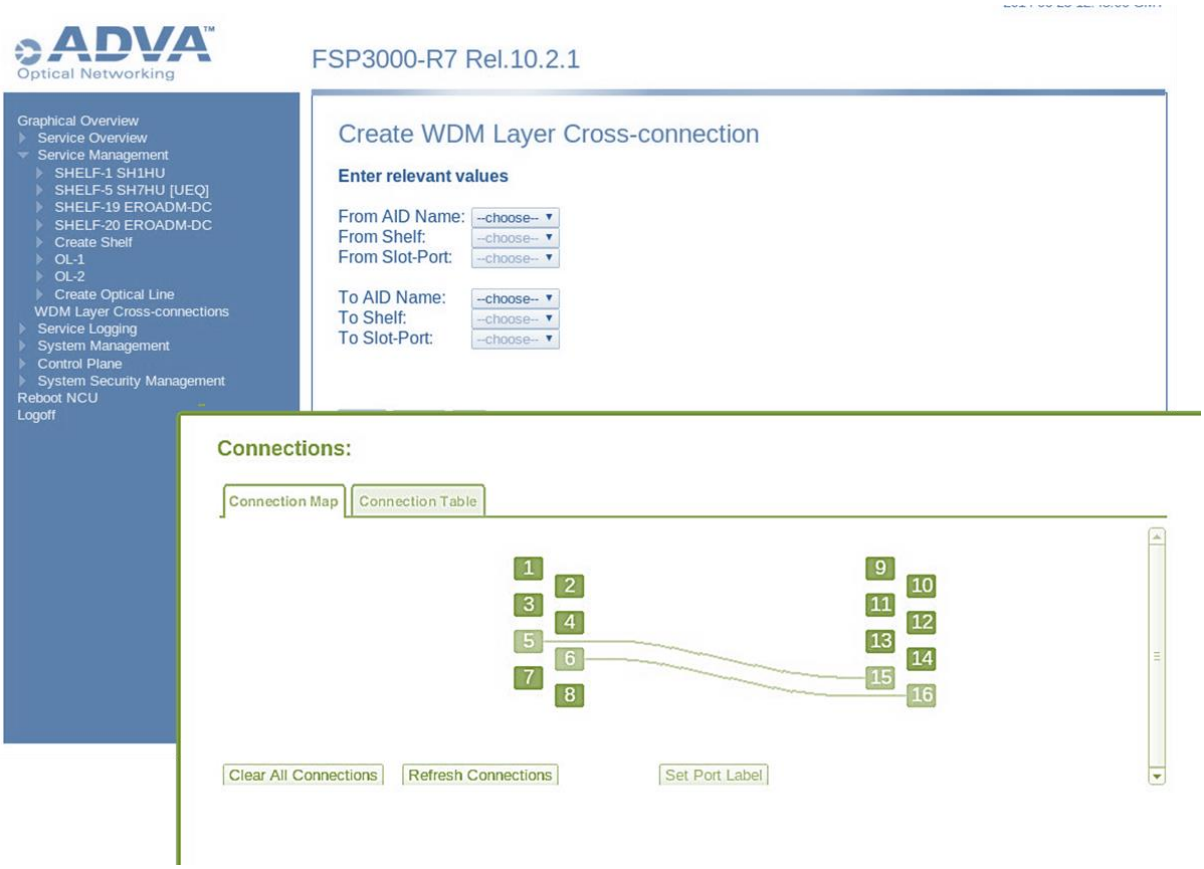


Figure 6-1 Cross connections web panel provided by different optical switch vendors

6.2 Experiment goals

In the experiment described in the following sections we are describing such a case where we are dealing with a network topology comprising of optical devices of different switching technologies also coming from different vendors. On the one hand, we have a ROADM device, demonstrated in D5.2 which is able to perform wavelength and space switching. On the other, we have a low latency fibre switches capable of space switching only. The SDN controller utilized is responsible for discovering and controlling these types of devices. This demonstration is highlighting basically two points. Firstly, the ability of OpenFlow to act as an enabler to control different types of optical devices, using different technologies and from different vendors. Secondly, HAL and the way it facilitates the process of writing OpenFlow datapaths for non OpenFlow devices, like fibre switches demonstrated below. For the purpose of this experiment we develop another OpenFlow agent based again in Alien HAL layer for fibre switches this. Most focus is given on this new development and also how OpenFlow abstraction smoothens the differences between optical switching technologies.

6.3 Experiment environment

In this section we are describing the building blocks that allow us to realise the experiment. The experiment requires that all devices involved can support OpenFlow v1.0. Furthermore, we need to ensure compatibility with the optical extensions

proposed in the Circuit switch addendum [OPEXT] to be able to control the optical nodes of the demo. These extensions have been developed and integrated to ROFL library as part of the developments for the ROADM switches [D3.3]. In the following block diagram we are exposing the core modules utilised and also the different layers of abstraction.

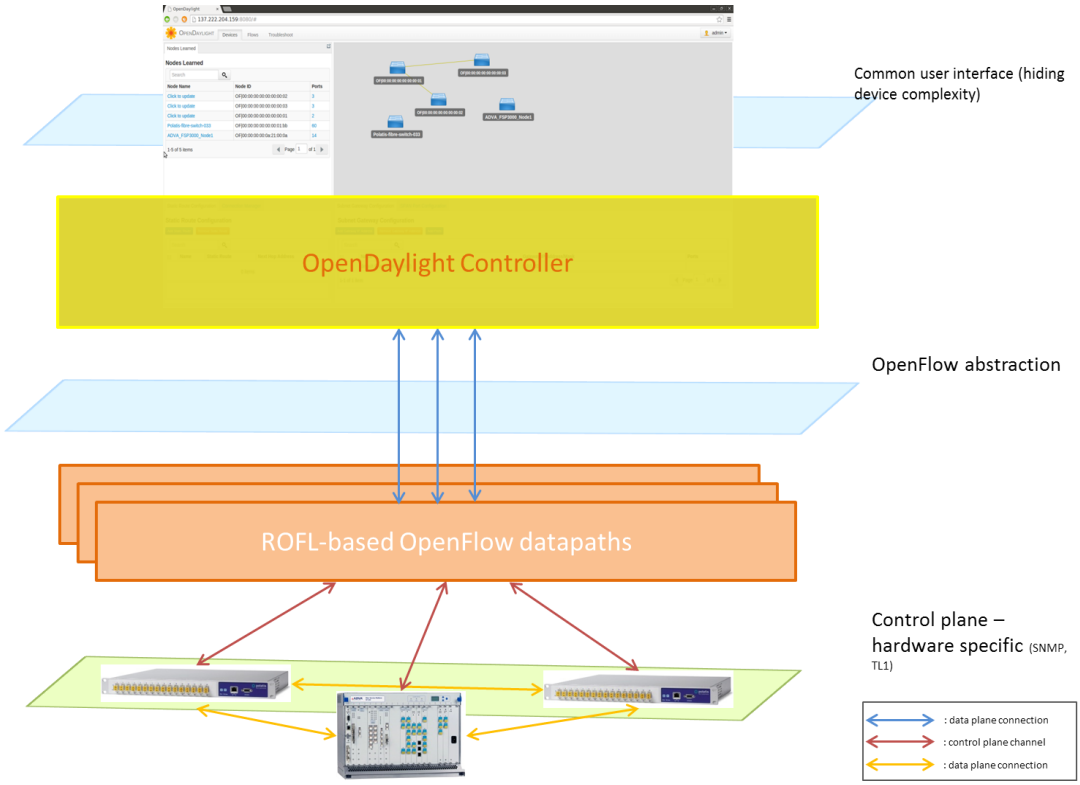


Figure 6-2 Hierarchy of building blocks demonstrated

6.3.1 Wavelength Selective switch OpenFlow datapath

The Layer0 switches are comprised of a wavelength selective switching (WSS) module and a space switching module. UNIVBRIS has developed an OpenFlow datapath for Layer0 switches and the code software has been tested with an extended version of NOX controller and ADVA FSP3000 devices hosted HPNG lab at University of Bristol. The software stack has been described in D3.3 and demonstrated in D5.2, thus we will not get into details about the actual implementation and functions provided by the developed OpenFlow agent. The diagram below shows a high level overview of the software.

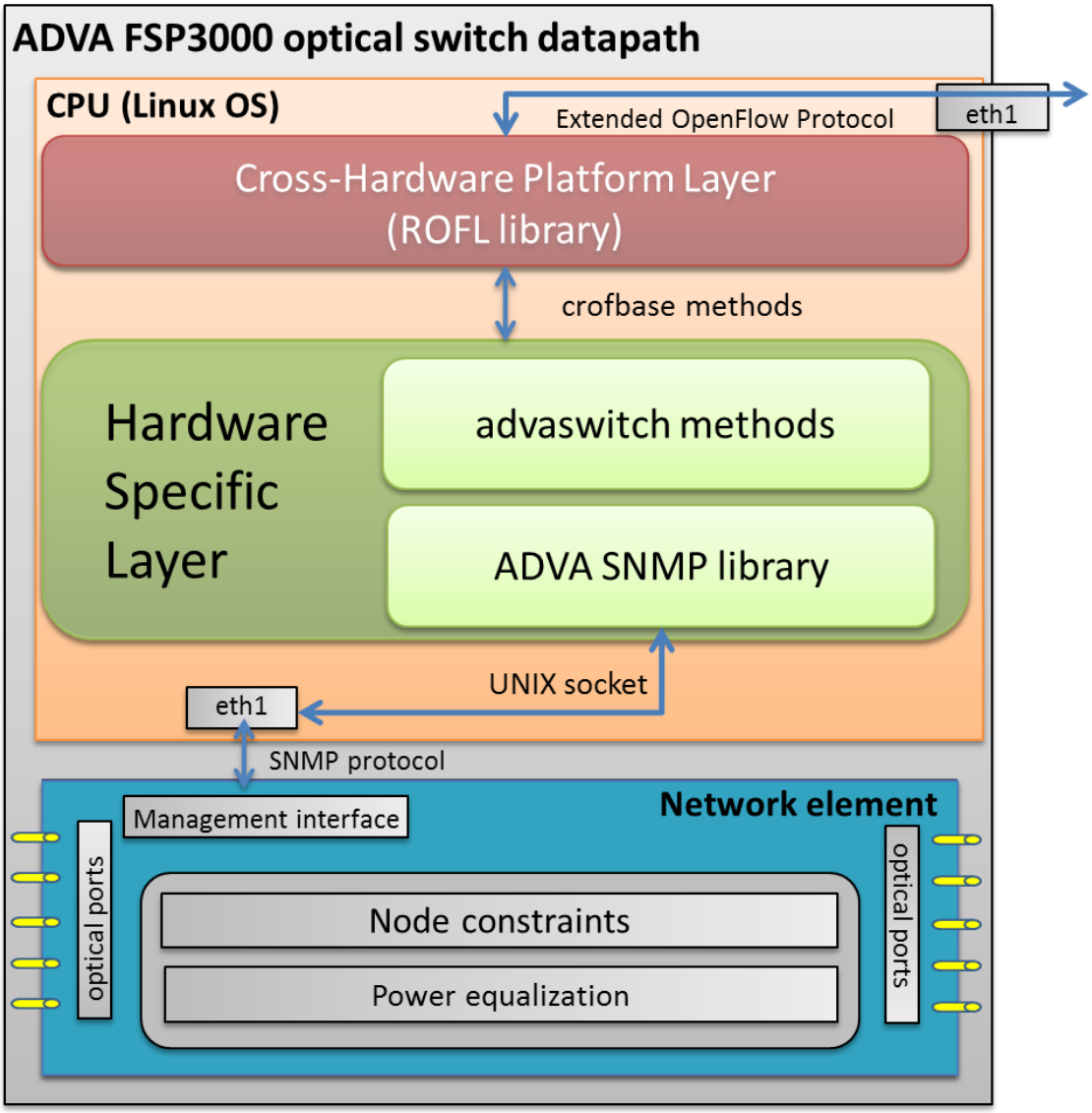


Figure 6-3 WSS optical switch datapath software building blocks

6.3.2 Fibre channel switch OpenFlow datapath

Similar to the Layer0 space and wavelength optical switch we have developed an OpenFlow datapath, compatible with fibre switches (space optical switch). UNIVBRIS has tested this software against a Polatis [POL] fibre switch for the purpose of this demonstration. The purpose of this development is to showcase the facilitation in the process of developing an OpenFlow datapath using a common hardware abstraction layer. The differences between the two agents can only be noted in the hardware specific layer, where the WSS switch is using the SNMP protocol while the fibre switch is using the TL1 protocol. Furthermore, for the WSS switch we had to develop a number of vendor-specific messages (OFPT_VENDOR) to allow control of the device, while for the fibre switch no additional messages where developed.

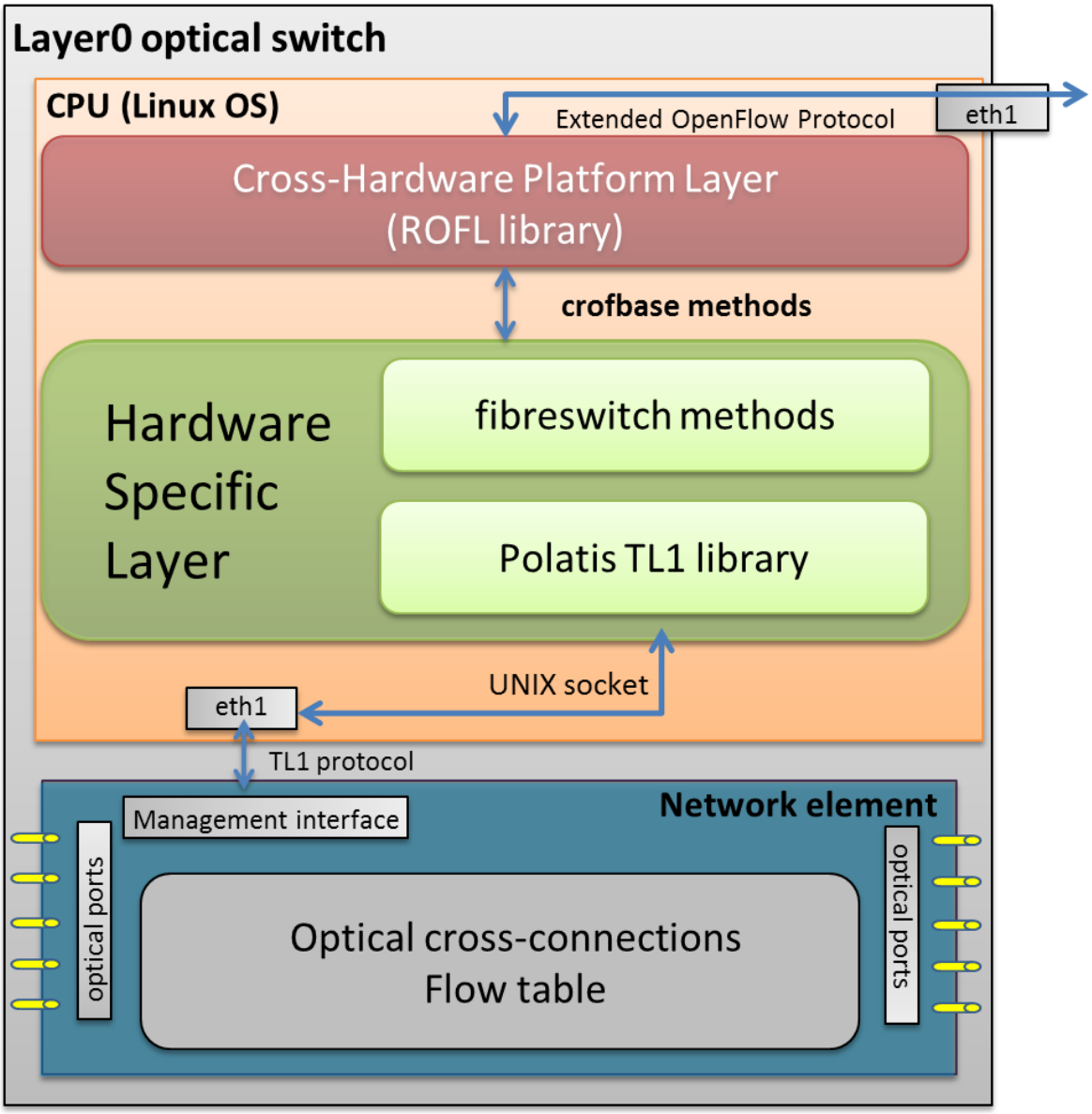


Figure 6-4 Fibre switch datapath software building blocks

The hardware specific part of the fibre switch has been developed around the ROFL library and we have reused the methods extended for the Layer0 WSS switch. Of course the implementation of these methods is different and specific to the hardware used to test the OpenFlow agent. However the Hardware Abstraction Layer (Alien HAL) is handling the connection with the controller and the maintenance of the secure channel. Messages that are common among all network nodes and are not specific to the device are used directly from crofbase class with no modifications as shown in Figure 6-4 above.

However there are also methods implementing OpenFlow messages that require interaction with the device and use of the Polatis TL1 interface. In order to parse these messages from the controller and push the commands to the switch we have used the polatisswclass. This latter has inherited the structure and methods from the generic crofbase class and additionally we have implemented the methods that are specific to the Polatis fibre switch. The most important of them are the handle_features_request(cofctl* ctl, cofmsg_features_request* msg) and the handle_cflow_mod(cofctl* ctl,

cofmsg_clow_mod* msg). The first is triggered when a new FEATURES_REQUEST message is received by the controller. After communicating with the switch, using the TL1 management interface the agent will reply by sending a FEATURES_REPLY message to the controller containing the features of the fibre switch. The second method is responsible for installing cross-connections to the switch and tearing down existing cross connections. The structure send to the switch from OpenDaylight controller is the following:

```
struct ofp_cflow_mod {
struct ofp_header header;/* Openflow header */
    uint16_t command;/* one of OFPFC_* commands */
    uint16_t hard_timeout;/* max time to connection tear down,
                                        if 0 then explicit tear-down required */
    uint8_t pad[4];/* Align to 64 bits */
struct ofp_connect_ocs connect;/* 8B followed by variable length arrays */
struct ofp_action_header actions[0];/* variable number of action */
};
```

The structure starts with the normal OpenFlow header like all other OpenFlow messages and then they follow the others structures required. The command field is taking values from the ofp_flow_mod_command_ocs enumeration and in our case it can be either OFPFC_OCS_ADDor OFPFC_OCS_DELETE depending on the case if we are parsing a message requesting an optical cross-connection or an optical path tear-down respectively. The hard_timeout can be used in cases where the duration of the flow is pre-determined or it can be zero if we don't want to specify it. Finally the ports that will be used are defined inside the ofp_connect_ocs struct. More information on these structures can be obtained from the specification provided by the circuit switch addendum [OPEXT]. The code of this implementation resides at [GIT FIBRESW] in a different branch called fibre-switch.

6.3.3 OpenDayLight Controller

OpenDaylight is an open source platform for network programmability hosted by Linux Foundation [ODL] that enables SDN through a combination of technologies including a fully pluggable controller, interfaces, protocol plug-ins and applications. The core part of the project is the modular, pluggable and flexible controller: it is implemented strictly in software and is contained within its own Java Virtual Machine (JVM) that can run on any platform that supports Java.

The core components of OpenDaylight are depicted in Figure 6-5. For the purpose of this demonstration we will use the controller module and the web user interface to show the discovered topology and push new flows to the network devices under controller. Also we have used the OpenFlow extensions developed as part of the Lightness FP7 project [LIGHT DEL].

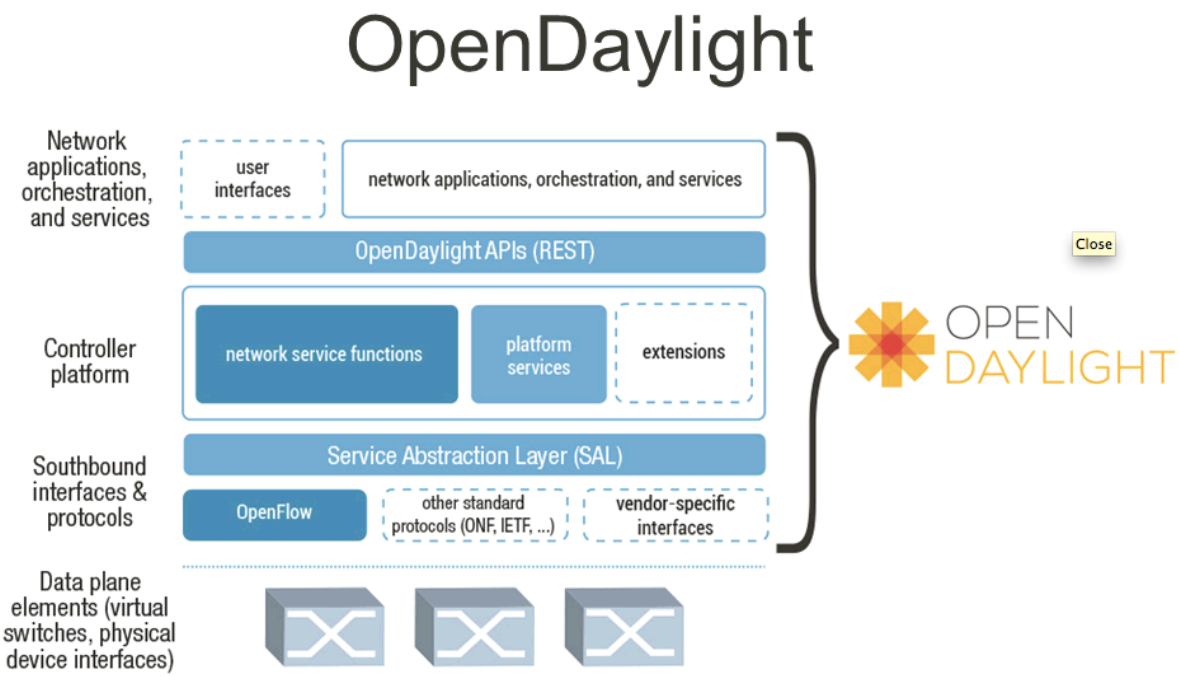


Figure 6-5 OpenDaylight controller building blocks

6.4 Testing procedure overview

More details of experiment scenario steps are provided in Appendix V.

6.5 Experiment results

All experiments results can be found in Appendix V.

6.6 Conclusions from validation of Multi-vendor evaluation of HAL in the optical domain

During the demonstration we just outlined we have tested successfully the following functions:

- OpenFlow 1.0 with circuit switch extensions (v0.3)
- Resource and peering features discovery of multi-technology (multi-vendor) optical domain
- Installing and tearing-down cross-connections using OpenFlow extensions for optical switches.

Our main objective is to showcase the beneficial behaviour of OpenFlow on controlling different types of optical devices, using different switching technologies compared to proprietary solutions provided by each vendor and the way HAL facilitates this process.

7 Summary and conclusions

This deliverable, which is rather heavy in technical content, concludes the experiments performed on OFELIA testbed facilities. The activities in this work package have evolved since the original planning as the results have been produced. In this final deliverable the partners have shown that ALIEN developments can be used as a starting point for several OpenFlow related developments.

The detailed results analysis is performed for each individual experiment and reported in the corresponding section describing that experiment. Based on these analyses it is confidently claimed that this work will intrigue interest of research community for further developments in this area. For example,

i) The possibility to have a programmable datapath will foster the development of switch based optimized solutions.
ii) The QoS extensions designed taking into account the special characteristics of a shared network in the access are already being ported to a GPON OLT based access networks. They have started receiving a great interest from the industry.
iii) The TBAM and its related work can be further extended at the Control Framework level to support new approaches.
iv) The possibility to include multivendor and multi-technology equipment under a common architecture has shown its potential for further research and development.

The work presented in this document is already filling future papers and conferences and undoubtedly see the possibilities for further developments in the future.

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

8 References

[CCNx]	Content-Centric Networking, https://www.ccnx.org
[CONET]	COonten NETwork project, https://alpha.fp7-ofelia.eu/cms/assets/Newsletters-and-Press-Releases/Newsletter-Issue-10.pdf
[D2.2]	Alien project, Deliverable D2.2 "Specification of Hardware Abstraction Layer"
[D4.2]	Alien project, Deliverable D4.2 "Definition of interfaces toward the OFELIA CF and functional specification of management software"
[D4.3]	Alien project, Deliverable D4.3 "Experimental-driven research"
[Expedient]	Expedient: A Pluggable Platform for GENI Control Frameworks, http://yuba.stanford.edu/~jnaous/expedient/
[FloodLight]	FloodLight OpenFLow controller, http://www.projectfloodlight.org/floodlight/
[FlowVisor]	FlowVisor wiki page, https://github.com/OPENNETWORKINGLAB/flowvisor/wiki
[GIT FIBRESW]	Github repository containing fibre switch code, https://github.com/fp7-alien/adva-rofl-dp
[LIGHT DEL]	Lightness FP7 Deliverable 4.6, http://www.ict-lightness.eu/wp-content/uploads/2014/09/lightness-D4.6_final.pdf
[ODL]	OpenDaylight, http://www.opendaylight.org/
[OF-SPEC-1.2]	ONF, OpenFlow switch specification version 1.2
[OFELIA]	M. Suñé, L. Bergesio, H.Woesner, T. Rothe, A. Köpsel, D. Colle, B. Puype, D. Simeonidou, R. Nejabati, M. Channegowda, et al. Design and implementation of the OFELIA FP7 facility: the European OpenFlow testbed. Computer Networks, 61:132–150, 2014.
[OFLOPS]	OFLOPS, http://archive.openflow.org/wk/index.php/Oflops
[OFLOPS 1]	Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, Andrew W. Moore, "OFLOPS: An Open Framework for OpenFlow Switch Evaluation", in Passive and Active Measurement, Lecture Notes in Computer Science Volume 7192, 2012, pp 85-95, Springer 2012
[OPEXT]	Extensions to the OpenFlow Protocol in support of Circuit Switching, Addendum to OpenFlow Protocol Specification (v1.0) – Circuit Switch Addendum v0.3, Saurav Das, June 2010 http://archive.openflow.org/wk/images/8/81/OpenFlow_Circuit_Switch_Specification_v0.3.pdf
[PAD-GITHUB]	Programmable Abstraction of Datapath, https://github.com/fp7-alien/pad
[PYPARSING]	Python parsing module, http://pyparsing.wikispaces.com/
[PYTUN]	Linux TUN/TAP wrapper for Python, https://pypi.python.org/pypi/python-pytun/0.2
[PYPARSING]	Python parsing module, http://pyparsing.wikispaces.com/
[POL]	Polatis fibre switch vendor, http://www.polatis.com/
[P4]	Programming Protocol-Independent Packet Processors, http://arxiv.org/abs/1312.1719
[ROFL]	Collection of libraries to build OpenFlow controllers and switches, http://www.roflibs.org/
[Ryu]	Ryu SDN framework, http://osrg.github.io/ryu/
[SDN]	Software-Defined Networking: A Comprehensive Survey, Diego Kreutz et al http://arxiv.org/pdf/1406.0440.pdf
[VLC]	http://www.videolan.org/index.html
[xDPd]	OpenFlow multi-platform switch, http://www.xdpd.org/

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

9 Acronyms

[AFA]	Abstract Forwarding API
[ALHINP]	ALien HAL-based Integration Network Proxy
[ICTP]	Information Centric Transport Protocol
[IP]	Internet Protocol
[IP]	Internet Protocol
[PAD]	Programmable Abstraction of Datapath
[ROFL]	Revised OpenFlow Library
[SDN]	Software Defined Network
[TCAM]	Ternary content-addressable memory
[TRILL]	Transparent Interconnection of Lots of Links
[VLAN]	Virtual Local Area Network
[VXLAN]	Virtual Extensible Local Area Network
[UDP]	User Datagram Protocol
[xDPd]	eXtensible DataPath daemon