



ALIEN

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 D3.3: Final Prototypes of Hardware Specific Parts

Due date: 31/06/2014
Submission date: 25/07/2014
Editor: Damian Parniewicz (PSNC)
Internal reviewer: Bartosz Belter (PSNC)
Author list: Damian Parniewicz, Łukasz Ogrodowczyk (PSNC), Victor Fuentes, Eduardo Jacob (EHU/IPV), Marek Michalski (PUT), Richard Clegg (UCL), Umar Toseef, Kostas Pentikousis (EICT), Marc Sune, Hagen Woesner, Tobias Jungel (BISDN), Tasos Vlachogiannis (UNIVBRIS)

Dissemination level

-
- | | |
|-------------------------------------|---|
| <input checked="" type="checkbox"/> | PU: Public |
| <input type="checkbox"/> | PP: Restricted to other programme participants (including the Commission Services) |
| <input type="checkbox"/> | RE: Restricted to a group specified by the consortium (including the Commission Services) |
| <input type="checkbox"/> | CO: Confidential, only for members of the consortium (including the Commission Services) |
-

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

DRAFT

Table of Contents

Abstract	7
Excecutive Summary	8
1 Introduction	9
2 HSP Prototypes	10
2.1 HSP for EZappliance	10
2.1.1 Overview	10
2.1.2 Supported functionalities	11
2.1.3 Requirements, installation and configuration	13
2.1.4 Licenses and links	17
2.2 HSP for NetFPGA	17
2.2.1 Overview	17
2.2.2 Supported functionalities	18
2.2.3 Requirements, installation and configuration	18
2.2.4 Licenses and links	21
2.3 HSP for Cavium Octeon	21
2.3.1 Overview	21
2.3.2 Supported functionalities	22
2.3.3 Requirements, installation and configuration	24
2.3.4 Licenses and links	24
2.4 HSP for DOCSIS	24
2.4.1 Overview	24
2.4.2 Supported functionalities	24
2.4.3 Requirements, installation and configuration	26
2.4.4 Licenses and links	28
2.5 HSP for GEPON	28
2.5.1 Overview	28
2.5.2 Supported functionalities	29
2.5.3 Requirements, installation and configuration	30
2.5.4 Licenses and links	32
2.6 HSP for LO switch	33
2.6.1 Overview	33
2.6.2 Supported functionalities	33
2.6.3 Requirements, installation and configuration	36
2.6.4 Licenses and links	37
3 Lessons Learned	38
3.1 EZappliance HSP	38
3.2 NetFPGA HSP	38
3.3 DOCSIS HSP	39
3.4 GEPON HSP	39
3.5 LO switch HSP	40
4 xDPd/ROFL Improvements for ALIEN HSPs	41
5 Conclusions	42

References	44
Acronyms	45

DRAFT

List of Figures

Figure 1.1	Links to HSPs software packages	9
Figure 2.1	EZappliance HSP main components	11
Figure 2.2	Recommended environments for EZappliance HSP components	14
Figure 2.3	NetFPGA HSP general components	18
Figure 2.4	Implementation of HAL on the OCTEON Platform.	22
Figure 2.5	DOCSIS ALIEN-HAL based architecture	25
Figure 2.6	DOCSIS virtual switch model	25
Figure 2.7	ALHINP port mapping	28
Figure 2.8	Aggregation switch port mapping	28
Figure 2.9	OUI port mapping	29
Figure 2.10	GEAPON HSP general components	29
Figure 2.11	ADVA ROADM ROFL based datapath	33
Figure 2.12	Wireshark capture of initial communication and circuit port cross-connection	34
Figure 2.13	OpenFlow agent console startup and messages exchanged	35
Figure 5.1	Usage of xDPd and ROFL	42
Figure 5.2	HSP implementation approaches (b) and (c) in the context of HAL architecture (a)	43
Figure 5.3	Development of HSP prototypes - summary	43

DRAFT

List of Tables

Table 2.1	Software repository links	17
Table 2.2	Software repository links	21
Table 2.3	License Table.	24
Table 2.4	Software repository links	28
Table 2.5	Software repository links	32
Table 2.6	Software repository links	37

DRAFT

Abstract

This document provides the release notes for the Hardware Specific Parts (HSPs) enabling an OpenFlow 1.x control over a set of heterogeneous network platforms (i.e.: EZAppliance, NetFPGA, DOCSIS, GEAPON, ATCA with Cavium Octeon card and Dell Force10 Split Data Plane switch). The HSPs have been developed in a form of the hardware drivers for Hardware Abstraction Layer [9] software frameworks (see [6] and [4]). The HSP implementation has been made accordingly to the initial software design of hardware drivers provided in the project deliverable [8] as well as the high-level functional Hardware Abstraction Layer (HAL) presented in the project deliverable [9].

This document stands as a part of D3.3/MS11 that includes the following components:

- Hardware Specific Part prototypes for all hardware platform available in the consortium, delivered as a set of source codes and/or binaries.
- Release Notes providing HSPs description, lists of supported functionality, installation and configuration guidelines (this document).

DRAFT

Executive Summary

The ALIEN deliverable D3.3 is officially finishing Task 3.3 ("Hardware specific implementation and validation") in WP3 and thus finishing the WP3 activity as a whole.

The main goal of the HSPs development process was to provide an input for the HAL design performed in Task 2.2 ("Design and functional definition of Hardware Abstraction Layer") and to validate the HAL framework implementation provided by Task 2.3 ("Implementation of the common part of the OpenFlow datapath element").

Each HSP prototype has been developed and validated by disjoint teams, owning a specific platform instance in their local testbeds:

- EZappliance HSP –PSNC
- DOCSIS HSP –UPV/EHU
- GEAPON HSP –UCL
- L0 switch –UNIVBRIS
- NetFPGA HSP –PUT
- Cavium Octeon HSP (covering ATCA platform and Force10 switch) –BISDN

The initial HSP validation has been performed with the usage of the OFtest testing tool which provides quite detailed analysis of implemented OpenFlow features, both at the control- and data- plane levels. The tests results have been included to this document as a subsection of Section 2. Additionally, the EZappliance and DOCSIS developments have been validated during live demonstrations supported by WP3 teams and presented during the FIA2014 [2] and TNC2014 [5] international conferences. The demonstration efforts have been summarized in a poster article submitted and accepted to EWSDN2014 and TNC2014 [10][11].

Further validation of the HSP prototypes will be carried out in WP5, after a successful integration with the OFELIA Control Framework elements (supported by WP4) and integration with OpenFlow controllers as identified in WP5 experiments.

1 Introduction

The concept of the HAL, a key enabler for an implementation of OpenFlow on alien devices, has been introduced in the ALIEN project and successfully disseminated to the Future Internet communities clustered around SDN. The HAL, since now materialized with well-tested proof-of-concept implementations, is available for several platforms, identified in the project as main targets of software development activities.

This deliverable provides the release notes of the Hardware Specific Parts (HSPs) enabling OpenFlow 1.x control over a set of heterogeneous network platforms (i.e.: EZappliance, NetFPGA, DOCSIS, GEAPON, ATCA with Cavium Octeon card and Dell Force10 Split Data Plane switch). The document summarizes development efforts towards hardware-specific parts of the HAL and provides a good overview of OF-based functionality currently supported by each platform.

All results of software development activities in ALIEN are accessible from a single web page http://www.fp7-alien.eu/?page_id=607, which summarizes practical outcomes of the project. Additionally, all ALIEN's HSP prototypes are publicly available at the ALIEN's github account, as presented in the table below. The only exception is the Octeon HSP prototype, which has been developed with a proprietary license due to the Non-disclosure Agreement signed by BISDN with an external company -- Cavium. Accessing the Octeon HSP source code and binaries are the subject of a bilateral agreement with BISDN upon a specific request of an interested third party. Similarly, the EZappliance HPS parts (Hardware Datapath and EZ Proxy) are covered by Non-disclosure Agreement signed by PSNC with EZchip.

Platform	Software package(s)	Location
Ezappliance	xDPd-for-Ezappliance	https://github.com/fp7-alien/xDPd-for-EZappliance
	EZ-Proxy and Hardware-Datapath (binary)	https://github.com/fp7-alien/xDPd-for-EZappliance/tree/master/src/xdpd/drivers/ezappliance/EZproxy-binary
NetFPGA	xDPd-for-NetFPGA	https://github.com/fp7-alien/xDPd-for-netfpga1g
Cavium Octeon	xDPd with Octeon driver	Provided only on direct request
DOCSIS	ALHIMP	https://github.com/fp7-alien/ALHIMP
GEAPON	xCPd	https://github.com/fp7-alien/xcpd
LO switch	ADVA-ROFL-DP	https://github.com/fp7-alien/adva-rofl-dp

Figure 1.1: Links to HSPs software packages

The remainder of this deliverable is organized as follows. Section 2 presents details of Hardware Specific Part (HSP) software packages developed for each ALIEN platform. Section 3 details lessons learnt and experiences gathered during the implementation phase related to porting xDPd/ROFL to ALIEN platforms. Section 4 presents major improvements and advances made to ROFL and xDPd within ALIEN, as a result of a deep usage of these packages in the project. Finally, Section 5 summarizes and concludes the deliverable.

2 HSP Prototypes

This section presents developed Hardware Specific Part (HSP) software packages for each ALIEN platform. The description of each HSP presents software components developed by the ALIEN teams and software location in the context of the hardware platform. In order to reflect that HSP implementation are done accordingly ALIEN HAL architecture [D2.2], the HSP overview pictures contain also position of HAL layers, HAL interfaces and most important functional blocks already presented in deliverable [D3.2].

Each HSP section contains information what exact functionality is provided by HSP complemented with OFtest tool [1] results which precisely validated both data plane and control plane functionalities of each HSP prototype.

The very important part of HSP description is a list of requirements and user manuals presenting installation and configuration procedures which can be used by anyone interested in testing or usage of ALIEN project developments.

The description of HSP is finished with a table containing information about licensing, form (source code or binary) and links to HSP software packages.

2.1 HSP for EZappliance

2.1.1 Overview

The HAL Hardware Specific Part for EZappliance platform is implemented as a set of three software packages (see Figure 2.1):

- xDPd for EZappliance
- EZ Proxy
- Hardware Datapath.

HSP for EZappliance was developed by Poznan Supercomputing and Networking Center (PSNC).

2.1.1.1 xDPd for EZappliance The xDPd project [6] is a software framework for instantiating platform specific components and is capable of tasks scheduling both for hardware agnostic and hardware driver work flows. Within xDPd framework, we have implemented the hardware driver for EZappliance device. The EZappliance driver uses IPC Corba interface to discover Data Plane interfaces and control Hardware Datapath located in EZchip NP-3 processor. The driver also creates a TCP connection to EZ Proxy which is used for OpenFlow packet-in and packet-out functionality. Additionally, The hardware driver for EZappliance includes also OpenFlow software pipeline from the ROFL library, which perform role of slow rule cache and complements functional limitations of the Hardware Datapath component. All flow entries, which are supported by Hardware Datapath, are passed to NP-3 processor (more sophisticated rule caching algorithms like [12] are currently not implemented). However, the ROFL software pipeline contains all flow entries and is applying OpenFlow matching on packets which were not matched in Hardware Pipeline. All EZappliance HSP software modules have been implemented in C++ language.

2.1.1.2 EZ Proxy EZ Proxy is a wrapper around EZchip EZdriver library. EZ Proxy simplifies EZchip vendor C++ library API, hiding the complexity of NP-3 network processor controlling and exposes the required functionality by IPC Corba technology [3]. EZ Proxy offers direct access to search structures and statistic counters located in NP-3 memory, allows for TOP MicroPrograms deploying inside NP-3 and controlling MicroPrograms behavior. API allows also for bidirectional frame transmission between Control CPU and NP-3, and for initial configuration of network processor and its memories. EZ Proxy has been developed in C++ language.

2.1.1.3 Hardware Datapath Hardware Datapath is an OpenFlow pipeline implementation for NP-3 network processor. It is processing frames with a full speed of the network processor. Hardware Datapath is implemented as a set of TOP MicroPrograms using EZchip assembler language. In Hardware Datapath, the OpenFlow pipeline is implemented directly using one search structure dedicated for all flow tables and second search structure dedicated for mapping of output

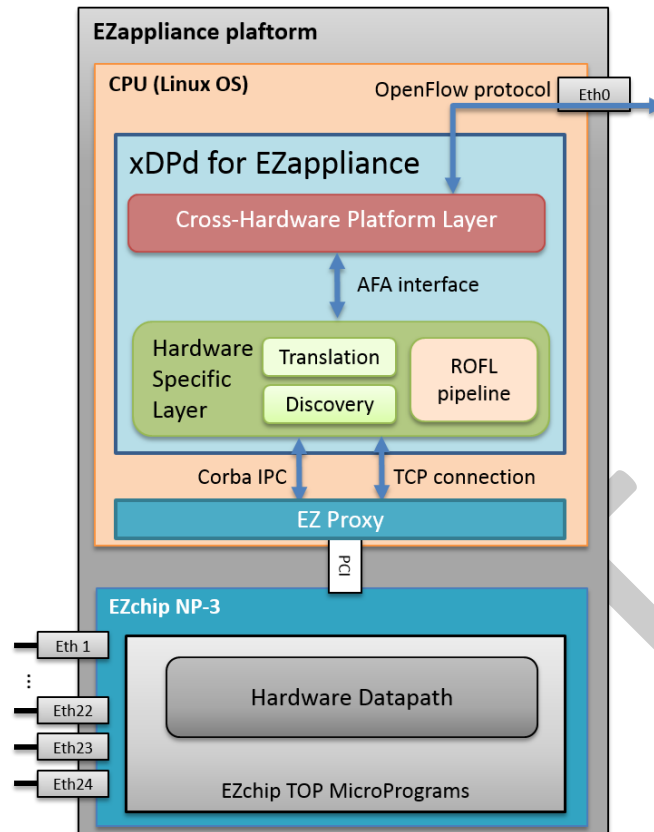


Figure 2.1: EZappliance HSP main components

ports into physical resources (queues in NP-3 Traffic Manager). The flow tables are stored in TCAM memory supporting wildcard matching which imposes limitations for amount of supported header fields. The packets which were not matched successfully in Hardware Datapath are sent to EZappliance hardware driver instance in xDPd framework.

2.1.2 Supported functionalities

The main influence to the list of EZappliance HSP supported functionalities has Hardware Pipeline component implemented for NP-3 network processor. Currently EZappliance supports:

- OpenFlow version 1.0 and 1.2 in terms of communication protocol (not data plane functionality)
- Part of OpenFlow 1.0 data plane functionalities:
 - One flow table
 - Ethernet, VLAN, IPv4, ICMP and ARP matches
 - Single port packet forwarding and drop actions
 - Packet-in and Packet-out

The more detailed information about supported functionalities are presented as Ofltest tool [1] results performed with usage of OpenFlow 1.0:

Basic protocol behaviour:

```
sh# alien@alien:~/oftest$ sudo ./oft basic -p 6633 -i 15@eth7 -i 16@eth6 -i 17@eth5
basic.EchoWithData ... ok
```

```

basic.PacketInBroadcastCheck ... ok
basic.DescStatsGet ... ok
basic.PacketOutMC ... ok
basic.PacketOut ... ok
basic.PortConfigModErr ... ok
basic.Echo ... ok
basic.PortConfigMod ... ok
basic.FlowMod ... ok
basic.TableStatsGet ... ok
basic.BadMessage ... ok
basic.FlowStatsGet ... ok
basic.PacketIn ... ok

```

Supported protocols fields matching:

```

sh# alien@alien:~/oftest$ sudo ./oft flow_matches -p 6633 -i 15@eth7 -i 16@eth6 -i 17@eth5
flow_matches.UdpDstPort ... ok
flow_matches.IpTos ... FAIL (*)
flow_matches.WildcardMatchPrio ... FAIL (*)
flow_matches.EthernetDstAddress ... ok
flow_matches.UdpSrcPort ... ok
flow_matches.ExactMatch ... FAIL (*)
flow_matches.ICMPCode ... FAIL (*)
flow_matches.MultipleHeaderFieldL2 ... ok
flow_matches.MultipleHeaderFieldL4 ... FAIL (*)
flow_matches.VlanPCP ... ok
flow_matches.EthernetSrcAddress ... ok
flow_matches.AllWildcardMatch ... FAIL (*)
flow_matches.ICMPType ... FAIL (*)
flow_matches.IngressPort ... FAIL (*)
flow_matches.TcpSrcPort ... ok
flow_matches.TcpDstPort ... ok
flow_matches.ArpOpcode ... FAIL (*)
flow_matches.ExactMatchPrio ... ok
flow_matches.ArpTargetIP ... ok
flow_matches.ArpSenderIP ... ok
flow_matches.VlanId ... ok
flow_matches.IpProtocol ... ok
flow_matches.EthernetType ... ok

```

Supported flow entry actions:

```

sh# alien@alien:~/oftest$ sudo ./oft actions -p 6633 -i 15@eth7 -i 16@eth6 -i 17@eth5
actions.ModifyL4Dst ... FAIL (*)
actions.Announcement ... ok

```

```

actions.NoAction ... ok
actions.AddVlanTag ... FAIL (*)
actions.ModifyL2Src ... FAIL (*)
actions.ModifyTos ... FAIL (*)
actions.ForwardLocal ... ok
actions.ForwardAll ... FAIL (*)
actions.ModifyL4Src ... FAIL (*)
actions.ForwardTable ... FAIL (*)
actions.ForwardController ... FAIL (*)
actions.ModifyL2Dst ... FAIL (*)
actions.ForwardInport ... FAIL (*)
actions.ModifyL3Dst ... FAIL (*)
actions.ForwardFlood ... FAIL (*)
actions.VlanPrio2 ... FAIL (*)
actions.VlanPrio1 ... FAIL (*)
actions.ModifyL3Src ... FAIL (*)
actions.ModifyVlanTag ... FAIL (*)

```

(*) A feature having lower implementation priority because not required by any demonstrations; pending to be implemented;

2.1.3 Requirements, installation and configuration

The general overview of software packages and main environmental requirements are show in Figure 2.2. More info about each software package can be found in the following subsections.

2.1.3.1 xDPd for EZappliance The xDPd for EZappliance software must be run on modern standard CPU system. It is recommended to deploy it on x86 processor with Linux operating system (i.e. Ubuntu). It is not recommended to deploy xDPd for EZappliance over Freescale PowerPC MPC8543 present in EZappliance device, where it performs the role of Control CPU, because of its limited capabilities. MPC8543 is not capable of performing OpenFlow software pipeline processing.

Library requirements

The xDPd for EZappliance software requires all standard libraries foreseen by [6] and [4] software projects and ROFL library in version devel-0.4:

<https://github.com/bisdn/rofl-core/tree/devel-0.4>

Additionally, it is required to install omniORB implementation [omniORB] of the CORBA protocol as a set of the following software packages for Debian systems:

```

omniorb >= 4.1.x
omniidl >= 4.1.x
python-omniorb >= 3.x
omniidl-python >= 3.x

```

Installation

Note: Check always <https://github.com/fp7-alien/xDPd-for-EZappliance> for the newest installation and configuration guidelines.

First clone project from github.

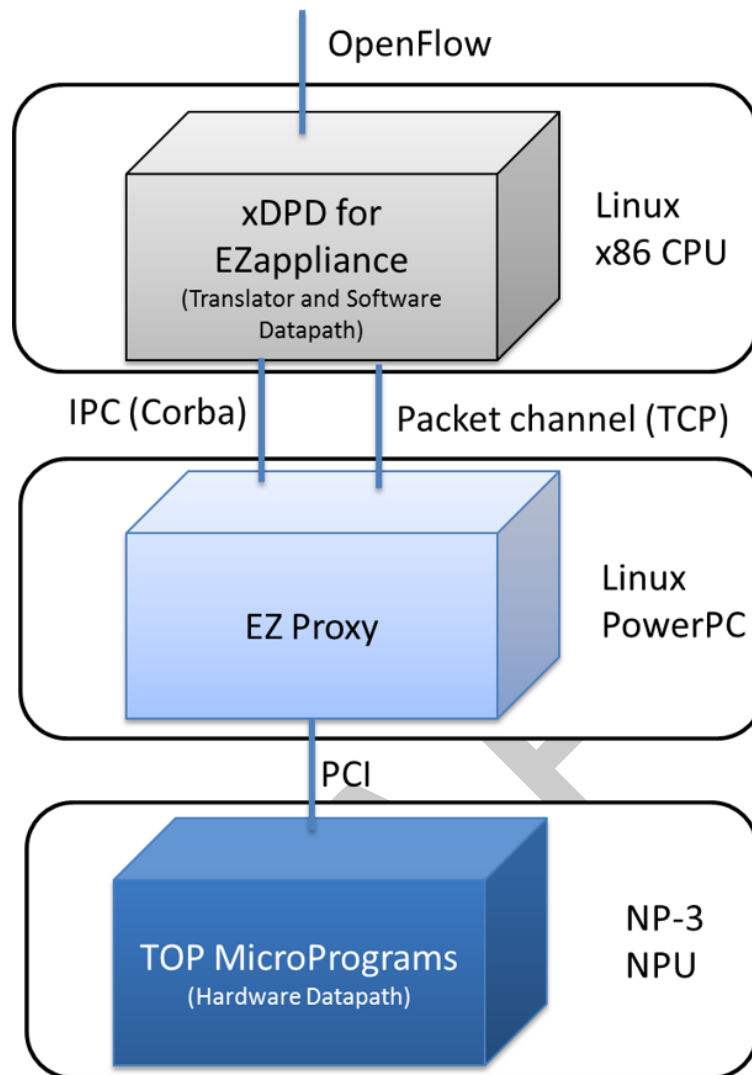


Figure 2.2: Recommended environments for EZAppliance HSP components

```
sh# git clone git://github.com/fp7-alien/xDPd-for-EZAppliance
```

Then you can build the project.

```
sh# cd ./xDPd-for-EZAppliance
sh# ./autogen.sh
```

At the end, it is specified that xDPd must be built and compiled with EZAppliance platform driver.

```
sh# cd build
sh# ./configure --with-hw-support=ezappliance
sh# make
```

Configuration

Copy the example configuration file and edit it:

```
sh# cd build/src/xdpd
sh# cp src/xdpd/management/plugins/config/example.cfg .
sh# vi src/xdpd/management/plugins/config/example.cfg
```

In the config file you must set required OpenFlow protocol version (1.0 or 1.2), number of tables equal 1, list of ports (from 'eth0' to 'eth23') and IP address of EZ Proxy module as the *driver-extra-params* attribute:

```
config:{
  openflow:{
    logical-switches:{
      #Name of the switch dp0
      dp0:{
        #Most complex configuration
        dpid = "0x1100000000000001"; #Must be hexadecimal
        version = 1.0;
        description="This is an PSNC-EZappliance switch";
        #Controller
        controller-connections:{
          main:{
            remote-hostname="10.134.0.15";
            remote-port=6633;
          };
        };
        reconnect-time=1; #seconds
        #Tables and MA
        num-of-tables=1;
        #Physical ports attached to this logical switch (mandatory)
        #The order and position in the array dictates the number of
        ports = ("eth0", "eth1", "eth2", "eth3", "eth4","eth5","eth6",
                "eth7","eth8","eth9","eth10");
      };
    };
  };
  system:{
    driver-extra-params="10.134.0.4"; #EZ Proxy IP address
  };
};
```

How to run it

Before starting xdpd, you need to guarantee that proper CORBA ior files, created by EZ Proxy, are available for xdpd in */tmp/ior* folder.

```
/tmp/ior/EZapi_struct.ior
/tmp/ior/EZapi_monitor.ior
```

Then just run xdpd.

```
sh# cd build/src/xdpd
sh# ./xdpd -c example.cfg
```

2.1.3.2 EZ Proxy and Hardware Datapath EZ Proxy is hosted on EZappliance Power PC embedded Linux where EZchip libraries are installed and direct access to EZppliance PCI interface is provided.

EZ Proxy compilation

For the compilation process the proprietary EZChip development environment is necessary. Final compiled program during run time on Control CPU loads dedicated MicroPrograms into TOP cores, sets up the whole environment (create channel, run driver) and runs EZ Proxy API. After successful compilation the new created binary can be found here: */home/alien/HAL/EZproxy/run*

How to mount EZproxy to the EZappliance

EZ Proxy is accessible from EZappliance Control CPU through the samba software and smb/cifs protocol. To mount the VM with EZproxy to the Control CPU system:

Login as a root to EZappliance (IP address of the Control CPU board) and check that */etc/profile* file contains:

```
sh# export LD_LIBRARY_PATH=/mnt/ezsamba/HAL/EZproxy/cross-compile/omniorb-cross-sx/lib:
                               /mnt/ezsamba/HAL/EZproxy/cross-compile/lib
sh# export PATH=$PATH:/mnt/ezsamba/HAL/EZproxy/cross-compile/omniorb-cross-sx/bin
sh# export LIBS=$LIBS:/mnt/ezsamba/HAL/EZproxy/cross-compile/lib
```

Then run the script:

```
sh# cd /home/user1
sh# ./ezsamba_alien.sh
```

How to run microcode for TOPs and EZproxy

ALIEN EZ Proxy executable binary should be run from Control CPU system. EZ Proxy uses proprietary shared object libraries from EZChip company. That's why Control CPU must be connected through SAMBA with additional Virtual Machine with EZChip libraries. EZ Proxy is accessible from EZappliance Control CPU through directory: */mnt/ezsamba*.

EZ Proxy can be run in **normal** or in **debug** mode. Debug mode is needed for debugging TOP processors using MDE (EZChip MicroCode Development Environment). It can be configured in the file located at

/mnt/ezsamba/HAL/EZproxy/run/AHE_MODE_INI

Normal mode configuration (AHE_MODE_INI file)

```
REAL
MCODE_HOST
CHAN_CRT_HOST
FRAMES_FROM_HOST_ALSO
MULTI_ENGINE
```

Debug mode configuration (AHE_MODE_INI file)

```
REAL
MCODE_MDE
CHAN_CRT_HOST
FRAMES_FROM_HOST_ALSO
SINGLE_ENGINE
```

All event are logged to EZproxy.log file in */mnt/ezsamba/HAL/EZproxy/run/*

How to run EZ Proxy:

Login as a root to EZappliance (IP address of the Control CPU board)

Project:	ALIEN (Grant Agr. No. 317880)
Deliverable Number:	D3.3
Date of Issue:	25/07/2014


```
sh# cd /mnt/ezsamba/HAL/EZproxy/run
sh# ./ALIEN
```

How to run EZ Proxy in debug mode (trace information mode):

Login as a root to EZappliance (IP address of the Control CPU board)

```
sh# cd /mnt/ezsamba/HAL/EZproxy/run
sh# ./ALIEN d
```

When binary file is executed it performs the following tasks:

- load microcode into NP-3 TOP cores,
- initialize the whole environment to work
 - create channel
 - create CORBA interfaces and related ior files:
 - * */mnt/ezsamba/HAL/EZproxy/iors/EZapi_struct.ior* – ior for manipulation of search entries in NP-3
 - * */mnt/ezsamba/HAL/EZproxy/iors/EZapi_monitor.ior* – ior for gathering information about data plane ports of EZappliance
 - * */mnt/ezsamba/HAL/EZproxy/iors/EZapi_tm.ior* – ior for NP-3 Traffic Managers as well as Token Bucket mechanism management
 - create TCP listening socket for network frames exchange between Hardware Datapath and xDPd driver for EZappliance.

2.1.4 Licenses and links

EZappliance HSP package	Licence	Link
xDPd for EZappliance	Mozilla Public Licence 2.0	(source code) https://github.com/fp7-alien/xDPd-for-EZappliance/tree/master/src/xdpd/drivers/ezappliance
EZ Proxy + Hardware Datapath	Proprietary	(binary file) https://github.com/fp7-alien/xDPd-for-EZappliance/tree/master/src/xdpd/drivers/ezappliance/EZproxy-binary

Table 2.1: Software repository links

2.2 HSP for NetFPGA

2.2.1 Overview

The general overview of HAL Hardware Specific Part for NetFPGA cards with OpenFlow and xDPd is shown in the Figure 2.3. In this case, the hardware and software parts have to be in the same machine because they communicate via PCI bus (see Figure 2.3). It is possible to install and properly maintain more than one NetFPGA cards in the same PC (even different versions, i.e. with interfaces with speed of 1Gbps and 10Gbps), but some operations (mentioned in details later in this section) have to be realized carefully and with awareness of the consequences. To run whole system both, software and hardware need to be installed properly. The prototype of OpenFlow switch based on NetFPGA and xDPd has been developed within ALIEN project.

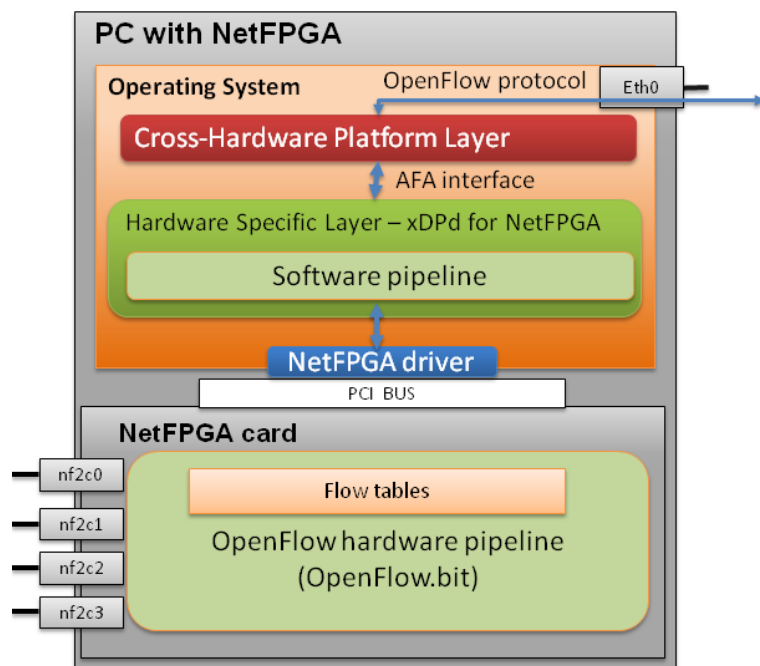


Figure 2.3: NetFPGA HSP general components

2.2.2 Supported functionalities

The final release of Openflow switch based on netfpga cards will support full functionality of OpenFlow 1.0 and some elements of higher versions using xDPd. Most of them will be realized in hardware part, only few will be masked by software pipeline realized by ROFL. In this moment prototype can realize: Typical for OpenFlow PacketIn and PacketOut, packet forwarding and dropping actions, field matching (Ethernet MAC addresses, IP addresses) Gathering statistics and flow table listing. In details, example of ofttest report is shown below:

```
./oft basic -p 6633 -i 1@eth0 -i 2@eth1 -i 3@eth2 -i 4@eth3
basic.EchoWithData ... ok
basic.PacketInBroadcastCheck ... ok
basic.DescStatsGet ... ok
basic.PacketOutMC ... ok
basic.PacketOut ... ok
basic.PortConfigModErr ... ok
basic.Echo ... ok
basic.PortConfigMod ... ok
basic.FlowMod ... ok
basic.TableStatsGet ... ok
basic.BadMessage ... ok
basic.FlowStatsGet ... ok
basic.PacketIn ... ok
```

2.2.3 Requirements, installation and configuration

According to our experience and official information, dedicated and fully supported operating system for NetFPGA 1G card is Fedora 14. It is possible to install netfpga driver on other systems, but it requires some tricks and does not guarantee fully correct operation.

Installation of NetFPGA cards

Here we will present only commands with minimal explanation of their meaning (only if this is needed, in most of the cases commands are self-defined). You will need root privileges to perform driver installation and grub.conf modification by `user_account_setup.pl` script. Please note that some additional packages may be required as a part of this process.

```
Install clean Fedora 14
```

```
yum update -y
yum install kernel-devel, kernel-headers
reboot
```

```
wget http://netfpga.org/yum/e15/RPMS/noarch/netfpga-repo-1-1_CentOS5.noarch.rpm
rpm -ivh netfpga-repo-1-1_CentOS5.noarch.rpm
(or)
rpm -Uvh http://netfpga.org/yum/e15/RPMS/noarch/netfpga-repo-1-1_CentOS5.noarch.rpm
```

```
yum install netfpga-base
```

```
cd ~
sudo /usr/local/netfpga/lib/scripts/user_account_setup/user_account_setup.pl
cd ~/netfpga
sudo make
sudo make install
reboot
```

After reboot you should see something as an output of command: (loaded driver `nf2.ko`)

```
lsmod | grep nf
```

Command `ifconfig` should show `nf2c0`, `nf2c1`, `nf2c2` and `nf2c3` interfaces for first NetFPGA card. If you have multiple cards in one machine, you will see next 4 interfaces with names `nf2c{4xN+0}` `nf2c{4xN+1}` `nf2c{4xN+2}` `nf2c{4xN+3}` for each NetFPGA card.

Software tools should be available:

```
nf_info
nf_download
```

`nf_info [-l nf2c{4N}]` allows to check actual state of netfpga card (with number N if you have more than one in one machine)

`nf_download project.bit [-l nf2c{4N}]` allows to download bit file to FPGA chip of NetFPGA (with number N)card. It starts running immediately after downloading.

Installation of OpenFlow.bit in NetFPGA cards

There are several acceptable bitfiles for OpenFlow. You can use demo or synthesize your own. To prepare own bitfile you will need IDE and source code, which requires several licenses (see section License) and experience in its configuration. To make whole process easier, we suggest to use public OpenFlow demo implementation for netFPGA cards. To download and install it use following commands:

```
sudo yum install netfpga-openflow_switch
/usr/local/netfpga/lib/scripts/user_account_setup/user_account_setup.pl
```

Example of command for downloading openflow project to hardware FPGA chip on NetFPGA card where at least 3 of them are in the same machine:

```
nf_download ~/netfpga/bitfiles/openflow.bit -i nf2c8
```

Installation of xDPd for NetFPGA cards

1. ROFL library: xDPd require ROFL-core library to be installed in path available for xDPd executive binary. All required information in this regard is accessible at ROFL website [4]. Here we put only commands which should download, compile and install library considering an optimistic case.

```
git clone https://github.com/bisdn/rofl-core
cd rofl-core
git checkout devel-0.3
sh autogen.sh
cd build
../configure
make
sudo make install
```

2. xDPd source and binary: You can download xDPd for netFPGA cards from several online locations. However, you must be careful as few locations host xDPd software for two different versions of NetFPGA cards (NetFPGA1G and NetFPGA10G) which are not compatible with each other.

```
git clone git://github.com/fp7-alien/xDPd-for-netfpga1g
cd xDPd-for-netfpga1g
sh autogen.sh
cd build
../configure -with-hw-support=netfpga1g
make
```

After this you will obtain executable binary file in build/src/xdpd. To run xDPd, it needs config file which provide information about current hardware machine configuration and parameters of controller. An example configuration is presented below:

```
config:{
  openflow:{
    logical-switches:{
      #Name of the switch dp0
      dp0:{
        #Most complex configuration
        dpid = "0x004E463243FF"; #Must be hexadecimal
        version = 1.0;
        description="NetFPGA1G-xDPd-OpenFlow-sw";
        #Controller
        controller-connections:{
          main:{
            remote-hostname="7.7.7.6";
            remote-port=6633;
          };
        };
        reconnect-time=1; #seconds
        #Tables and MA
```

```

        num-of-tables=1;
        #Physical ports attached to this logical switch (mandatory).
        #The order and position in the array dictates the number of
        ports = ("nf2c0", " nf2c1", " nf2c2", " nf2c3");
    };
};
};
system:{
};
};
};

```

Important notice: It is possible to maintain more than one NetFPGA card with xDPd and OpenFlow in the same machine, but each of them should have its own instance of xDPd for NetFPGA card, because it represents one hardware switch, but two netfpga cards are not one switch -- there is no connection in hardware between ports from different cards.

Editorial comment: Above sentences need revision to convey the clear meaning.

Configuration file for next NetFPGA card should have different interfaces:

```
ports = ("nf2c4", " nf2c5", " nf2c6", " nf2c7");
```

and datapath ID:

```
dpid = "0x024E463243FF";
```

Important notice: When you use more than one netfpga cards in different machines, be careful with their MAC addresses. By default, each netfpga interface with name nf2c{0xHH} has MAC address equal to 004E463243HH.

When you have everything installed and prepared config you can start xDPd:

```
cd build/src/xdpd
./xdpd -c config_xdpd_for_netfpga.cfg
```

2.2.4 Licenses and links

NetFPGA HSP package	Licence	Link
OpenFlow-NetFPGA-1.0.0 reference switch	Proprietary	(source code) https://github.com/NetFPGA/netfpga/wiki/OpenFlowNetFPGA100
xDPd for NetFPGA	Proprietary	(source code) https://github.com/fp7-alien/xdpd-for-netfpga1g

Table 2.2: Software repository links

2.3 HSP for Cavium Octeon

2.3.1 Overview

The HSP for the Octeon platform is realized as a driver to xDPd and therefore implementing the AFA interface. The OF endpoint is running in the Linux core environment of the Octeon and accessing the HSP through the AFA interface (see Figure 2.4). The pipeline shown in the Linux part is used for Packet out requests and state keeping of all FlowMods. A new FlowMod in the Linux core is stored in a shared memory, that is accessible from all Standalone cores. The pipeline running

in each core of the Octeon is able to apply the OpenFlow rules written into that memory.

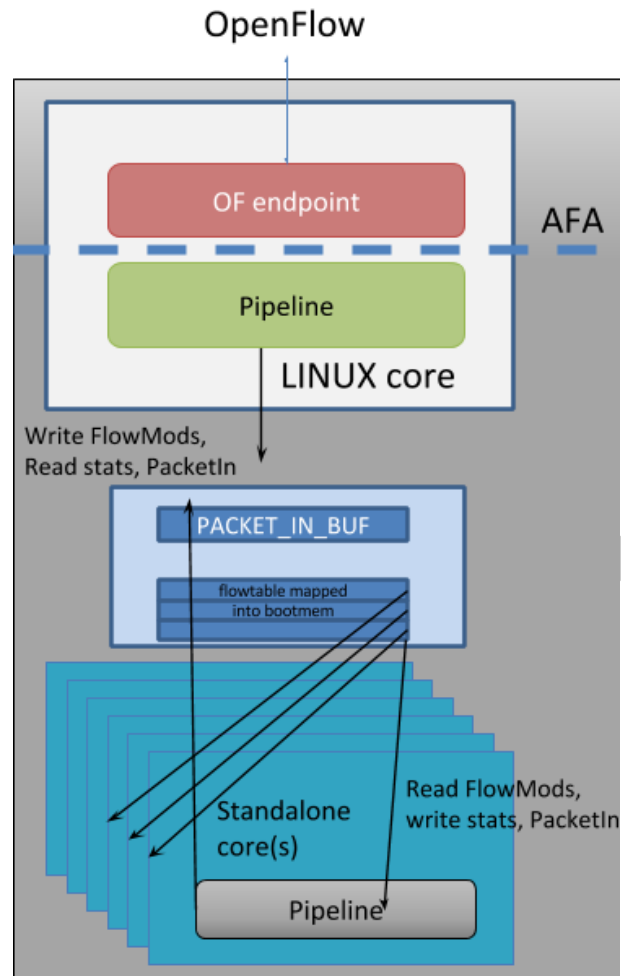


Figure 2.4: Implementation of HAL on the OCTEON Platform.

2.3.2 Supported functionalities

OFTest basic results:

```
# ./oft basic
basic.EchoWithData ... ok
basic.PacketInBroadcastCheck ... ok
basic.DescStatsGet ... ok
basic.PacketOutMC ... ok
basic.PacketOut ... ok
basic.PortConfigModErr ... ok
basic.Echo ... ok
basic.PortConfigMod ... ok
basic.FlowMod ... ok
basic.TableStatsGet ... ok
basic.BadMessage ... ok
basic.FlowStatsGet ... ok
basic.PacketIn ... ok
```

OFTest fields matching results:

```
# ./oft flow_matches
flow_matches.UdpDstPort ... ok
flow_matches.IpTos ... ok
flow_matches.WildcardMatchPrio ... ok
flow_matches.EthernetDstAddress ... ok
flow_matches.UdpSrcPort ... ok
flow_matches.ExactMatch ... ok
flow_matches.ICMPCode ... ok
flow_matches.MultipleHeaderFieldL2 ... ok
flow_matches.MultipleHeaderFieldL4 ... ok
flow_matches.VlanPCP ... ok
flow_matches.EthernetSrcAddress ... ok
flow_matches.AllWildcardMatch ... ok
flow_matches.ICMPType ... ok
flow_matches.IngressPort ... ok
flow_matches.TcpSrcPort ... ok
flow_matches.TcpDstPort ... ok
flow_matches.ArpOpcode ... ok
flow_matches.ExactMatchPrio ... ok
flow_matches.ArpTargetIP ... ok
flow_matches.ArpSenderIP ... ok
flow_matches.VlanId ... ok
flow_matches.IpProtocol ... ok
flow_matches.EthernetType ... ok
```

OFTest flow entry actions results:

```
# ./oft actions
actions.ModifyL4Dst ... ok
actions.Announcement ... ok
actions.NoAction ... ok
actions.AddVlanTag ... ok
actions.ModifyL2Src ... ok
actions.ModifyTos ... ok
actions.ForwardLocal ... ok
actions.ForwardAll ... ok
actions.ModifyL4Src ... ok
actions.ForwardTable ... FAIL (*)
actions.ForwardController ... ok
actions.ModifyL2Dst ... ok
actions.ForwardInport ... ok
actions.ModifyL3Dst ... ok
actions.ForwardFlood ... ok
actions.VlanPrio2 ... ok
actions.VlanPrio1 ... ok
actions.ModifyL3Src ... ok
actions.ModifyVlanTag ... ok
```

(*) By design ROFL-pipeline forbids the usage of OFF_TABLE as an output port (table loops).

2.3.3 Requirements, installation and configuration

The major requirement is of course the possession of an OCTEON network processor. There are different generations of OCTEON available, the HSP has been tested and developed for types OCTEON-1 (in the DELL SDP) and OCTEON-2 (in the Emerson ATCA). Some minimal adjustments need to be done in the source code in order to adapt to the correct version. Installation takes place typically through a serial interface (e.g., the USB port). Configuration of ports etc. needs to happen in the core(s) that run the CMM in the Linux implementation.

2.3.4 Licenses and links

OCTEON HSP package	Licence
xDPd	As the Cavium SDK that is the basis for this work comes with three different license models (including GPL), we decided to provide our code only to organisations that themselves have signed the NDA with Cavium Networks. Please contact B1SDN.

Table 2.3: License Table.

2.4 HSP for DOCSIS

2.4.1 Overview

The HAL Hardware Specific Part for the DOCSIS platform is implemented in several parts:

- ALHINP -- virtualization proxy
- Aggregation Hardware OpenFlow switch.
- OUI Hardware OpenFlow switch: OpenFlow switch instance running at the client side, next to the CableModem (CM)

The DOCSIS access network consists of a head end device (CMTS) and several tail end devices (CMs). The hardware OpenFlow Aggregation switch is placed after the CMTS, connecting the ALIEN DOCSIS platform with the core network.

ALHINP acts as a proxy that allows the system to expose the whole DOCSIS network as a single OpenFlow switch.

It converts incoming OpenFlow messages that come from a controller (OF 1.0) into actions/messages directed to the OpenFlow switches (OF 1.2) (OUI and/or Aggregation) and the CMTS. It also processes messages incoming from the switches, to generate new ones or to modify them, before sending them to the controller.

ALHINP is designed based on ROFL libraries. It has a configuration file where the definition of the ports and other parameters are stored.

2.4.2 Supported functionalities

Currently the OpenFlow DOCSIS platform supports:

- OpenFlow 1.0
- Single VLAN tagged traffic incoming from user port.
- Statistics per port / Desc
- Packet_IN / Packet_OUT

```
sudo ./oft basic -p 6633 -i 12@eth1 -i 21@eth2
basic.EchoWithData ... ok
```

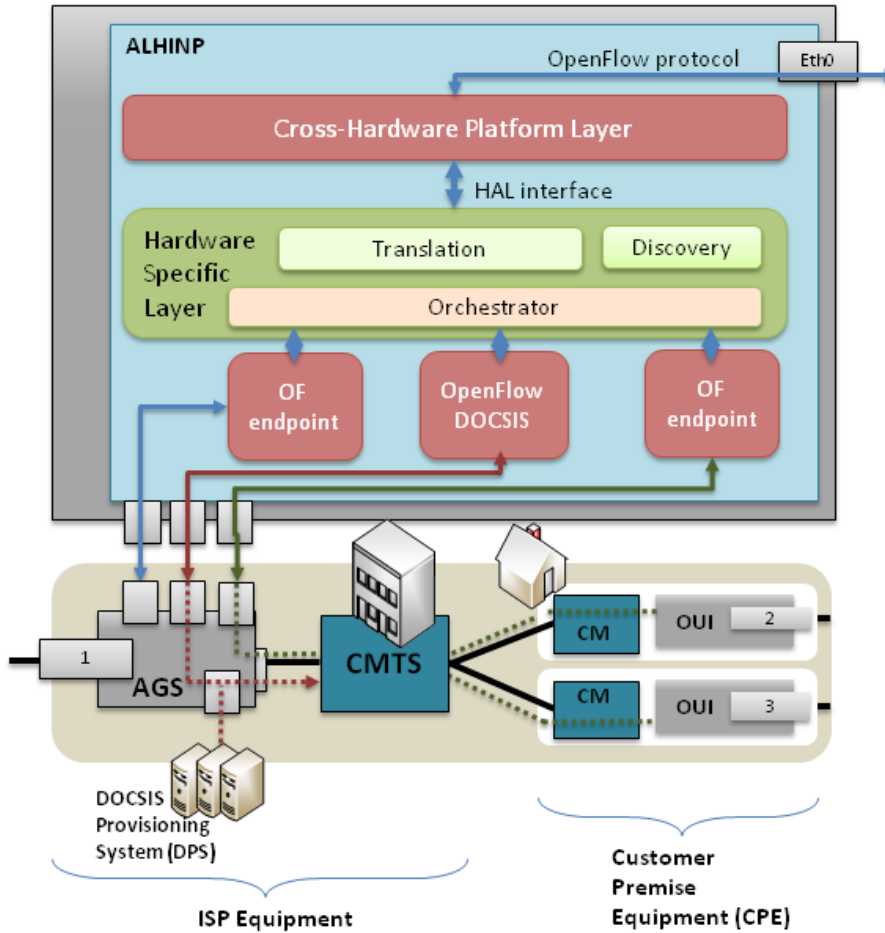



Figure 2.5: DOCSIS ALIEN-HAL based architecture

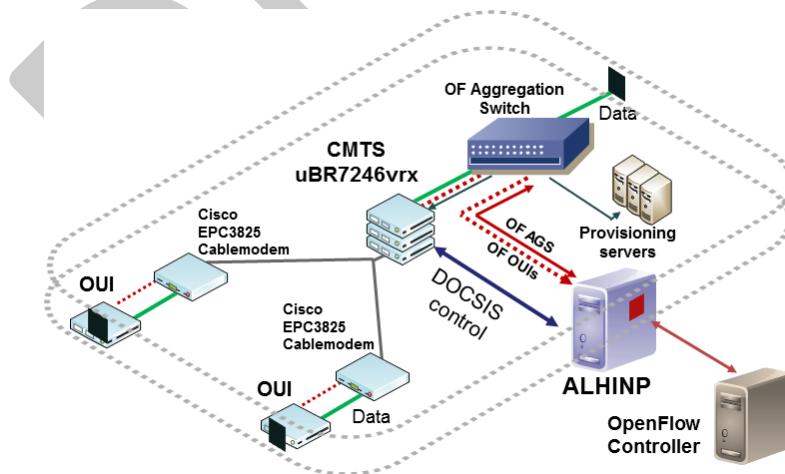


Figure 2.6: DOCSIS virtual switch model

```
basic.PacketInBroadcastCheck ... ok
basic.DescStatsGet ... ok
basic.PacketOutMC ... ok
```

```

basic.PacketOut ... ok
basic.PortConfigModErr ... ok
basic.Echo ... ok
basic.PortConfigMod ... FAIL (*)
basic.FlowMod ... ok
basic.TableStatsGet ... FAIL (**)
basic.BadMessage ... ok
basic.FlowStatsGet ... FAIL (***)
basic.PacketIn ... ok

```

- * as OFtest tries to set NO_FLOOD flag which is not used in OF1.2
- ** ALIEN DOCSIS platform has several tables virtualized as one.
- *** Related to ROFL-0.3 and stats under OF1.2 version

Supported protocol fields matching:

OFtest uses a wildcarded inport, which is not currently supported by the platform, and the results are not successful. However the platform supports the same matching fields as xDPD for Linux.

2.4.3 Requirements, installation and configuration

2.4.3.1 Requirements & Installation

ALHINP Installation ALHINP requires the Revised Open Flow Library [4] branch devel-0.3. To build all ROFL requirement libraries must be installed.

```

$ git clone https://github.com/bisdn/rofl-core
$ cd rofl-core
$ git checkout devel-0.3
$ sh autogen.sh
$ cd build
$ ../configure
$ make
$ sudo make install

```

it also requires libconfig++-dev library. For Debian-based distributions:

```
$ apt-get install libconfig++-dev
```

Compiling source code of ALHINP

```

git clone https://github.com/fp7-alien/alien-DOCSIS
cd ALHINP
./configure
make

```

Running ALHINP

```
./ALHINP <ALHINP.cfg>
```

This file describes required parameters to connect all devices in the ALIEN DOCSIS platform

```
// An example of ALHINP configuration file content.
name = "ALHINP";
desc = "EHU DOCSIS OF abstracted switch";
dpid = "1000000000000001";
// configuration:
ALHINP-config = {
//CONTROLLER-parameters
    CONTROLLER_IP           = "127.0.0.1";
    CONTROLLER_OF_VERSION   = "1.0";
    CONTROLLER_PORT         = 6633;
    LISTENING_IP_AGS        = "158.227.98.21";
    LISTENING_PORT_AGS      = 6633;
    LISTENING_IP_OUIS       = "192.168.10.1";
    LISTENING_PORT_OUIS     = 6633;
    DPS_IP                  = "10.10.10.62";
    CMTS_IP                 = "158.227.98.6";
    // MAC range of OF interface of the OUIs
    OUI_MAC                 = "02:00:0C:00:00:00";
    // Cable Modem MAC mask
    CM_MAC                  = "A4:A2:4A:00:00:00";
    VLANstart               = 2;
};
// Port configuration
Port-config = {
    //AGS ports
    CMTS_PORT   = 1;
    DATA_PORT  = 2;
    DPS_PORT    = 3;
    PROXY_PORT  = 4;
    //OUIs ports
    OUI_NETPORT = 2;
    OUI_USERPORT= 1;
};
AGS-config = {
    AGSDPID     = "1";
};
```

ALHINP port mapping

Aggregation Switch An OpenFlow 1.2 compatible switch (software or hardware) is required. In this case, xDPD software is used for Aggregation switch and OpenFlow User Instances (see Figure 2.8 for port mapping)

```
$ git clone https://github.com/bisdn/xdpd
$ cd rofl-core
$ git checkout master-0.3
$ sh autogen.sh
$ cd build
$ ../configure
$ make
```

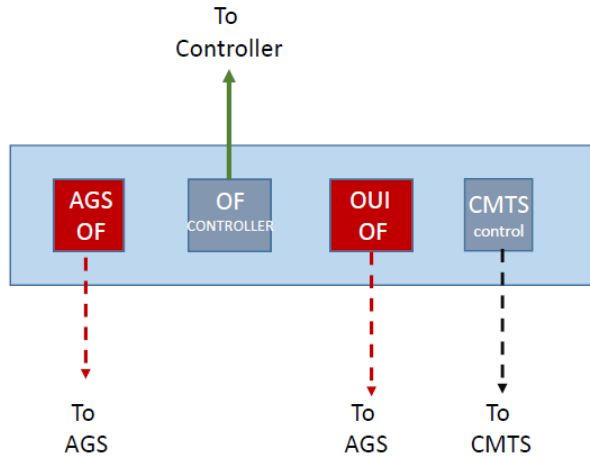


Figure 2.7: ALHINP port mapping

```
$ sudo make install
```

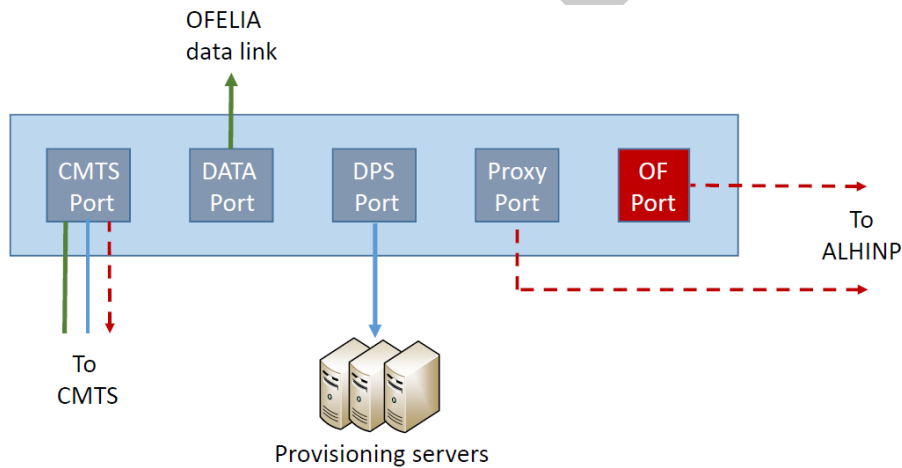


Figure 2.8: Aggregation switch port mapping

OpenFlow User Instance An OpenFlow 1.2 compatible switch (software or Hardware) is required. In this case, xDPD software is used for Aggregation switch and OpenFlow User Instances (see Figure 2.9 for port mapping). The same steps are followed as in the aggregation switch software compilation.

2.4.4 Licenses and links

DOCSIS HSP package	Licence	Link
HSP for DOCSIS (UPV/EHU)	Mozilla Public Licence 2.0	(source code) https://github.com/i2t/ALHINP

Table 2.4: Software repository links

2.5 HSP for GEAPON

2.5.1 Overview

Project:	ALIEN (Grant Agr. No. 317880)
Deliverable Number:	D3.3.1
Date of Issue:	25/07/2014

The HAL Hardware Specific Part for the GEAPON platform is implemented in two parts, one of which is off-the-shelf.

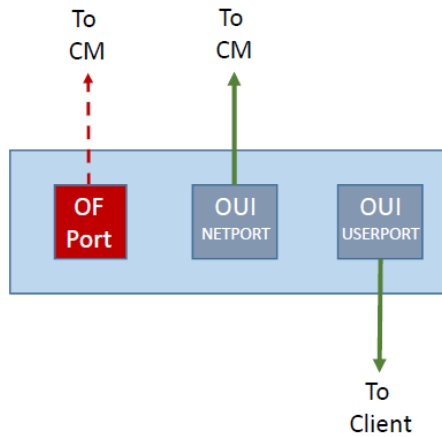


Figure 2.9: OUI port mapping

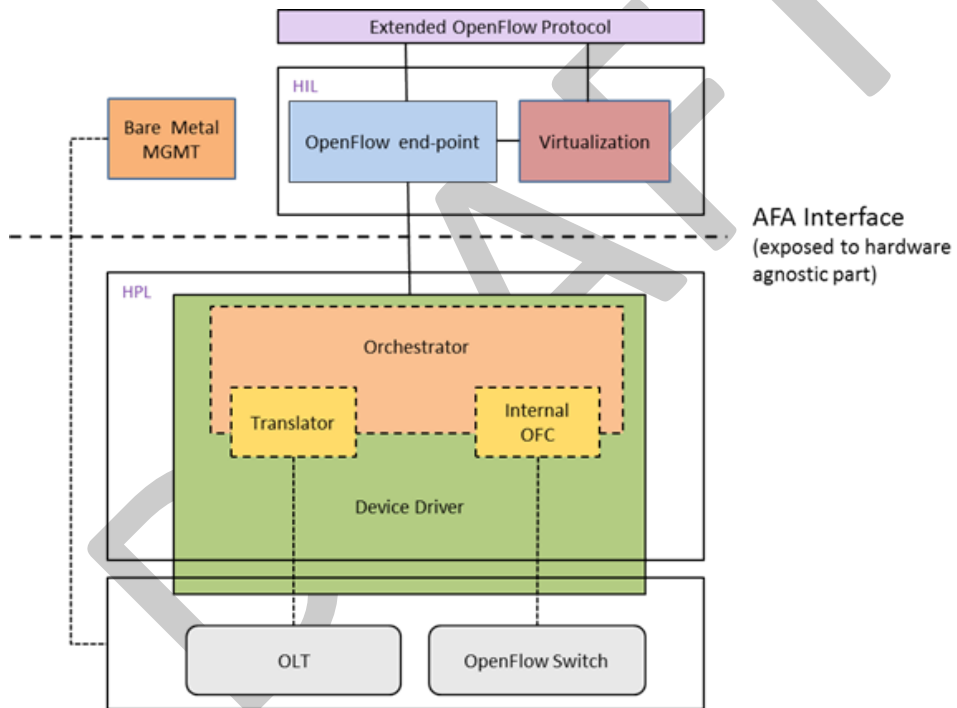


Figure 2.10: GEPON HSP general components

The lower level OpenFlow switch xDPd has two ports and communicates via OpenFlow to xCPd. xCPd pretends to be a switch with more ports and translates matches and actions to an appropriate format for the lower switch (xDPd).

2.5.2 Supported functionalities

The GEPON installation supports:

- One flow table
- Ethernet, VLAN, IPv4, ICMP and ARP matches
- Single port packet forwarding and drop actions

- Packet-in and Packet-out
- VLAN taggings (in some circumstances only)

The more detailed information about supported functionalities are presented as OFtest tool [1] results:

Basic protocol behaviour:

```
sh# connet@netfpga1 :~/oftest$ sudo ./oft basic -p 6633 -i 1@nf2c0 -i
                                2@nf2c1 -i 3@nf2c2 -i 4@nf2c3

basic.EchoWithData ... ok
basic.PacketInBroadcastCheck ... ok
basic.DescStatsGet ... ok
basic.PacketOutMC ... ok
basic.PacketOut ... ok
basic.PortConfigModErr ... ok
basic.Echo ... ok
basic.PortConfigMod ... ok
basic.FlowMod ... ok
basic.TableStatsGet ... ok
basic.BadMessage ... ok
basic.FlowStatsGet ... ok
basic.PacketIn ... FAIL (*)
```

The PacketIn task fails because it tests VLAN tagging that, unfortunately, fails because the tags are used by the OLT. If the VLAN tagging test is removed then this test passes.

2.5.3 Requirements, installation and configuration

xCPd requires the Revised Open Flow Library [4] branch devel-0.3. To build all ROFL requirement libraries must be installed.

```
$ git clone https://github.com/bisdn/rofl-core
$ cd rofl-core
$ git checkout devel-0.3
$ sh autogen.sh
$ cd build
$ ../configure
$ make
$ sudo make install
```

Then install xcpd itself

```
$ git clone https://github.com/fp7-alien/xcpd
$ cd xcpd
$ sh autogen.sh
$ cd build
$ ../configure
$ make
$ sudo make install
```

If xcpd is to be used with a common configuration file then the master-0.3 build must be used.

```
$ git clone https://github.com/bisdn/xdpd
$ cd rofl-core
$ git checkout master-0.3
$ sh autogen.sh
$ cd build
$ ../configure
$ make
$ sudo make install
```

Configuration

This file shows a joint configuration file used by both xdpd and xcpd. The xcpd part is at the end. The first half of the configuration is a standard configuration used by xdpd version master-0.3. Both xdpd and xcpd rely on this information. It is not necessary that both share a common configuration file but this is an option for ease of use and to ensure that if xdpd is used then both have a common view of the network. This version sets up the following.

xdpd connects to two ports, eth0 (network facing) and eth1 (OLT facing). xcpd pretends that the switch has five ports, port 1 is eth0, port 2,3,4 and 5 correspond to eth1 tagged with vlan tags 10,11,12,13. The sections below:

```
queue-command-handling="drop";
port-stat-handling="passthrough";
port-config-handling="passthrough";
```

refer to the handling of OpenFlow messages related to queues, port stats and port config commands. The options are drop, passthrough or hardware specific handler. These commands are special in the sense that they refer to OpenFlow functionality which cannot simply be mapped by VLAN tagging. For example a request for port statistics cannot just be translated to a request for statistics for a port and VLAN pair as detailed statistics are not kept. Three options are provided, drop (the command is ignored), passthrough (the command is passed through to the corresponding underlying physical port) and "hardware" (the command is routed via the MGMT port of the OLT and hardware specific code).

#Example of configuration single using xcpd -- xcpd section is at end

```
config:{
  openflow:{
    logical-switches:{
      #Name of the switch dp0
      dp0:{
        #Most complex configuration
        dpid = "0x101"; #Must be hexadecimal
        version = 1.0;
        description="xdpd lower switch";
        #Controller
        mode="passive"; #active, passive, (TODO: both)
        bind-address-ip="127.0.0.1";
        bind-address-port=6633;
        reconnect-time=1; #seconds
        #Tables and MA
        num-of-tables=1;
        #Physical ports attached to this logical switch. This is mandatory
        #The order and position in the array dictates the number of
        # 1 -> eth0, 2 -> eth1
        ports = ("eth0", "eth1");
```

```

    };
};
};
xcpd: {
  higher-controller-ip="127.0.0.1";
  higher-controller-port=6634;
  upward-mode="active";
  queue-command-handling="drop";
  port-stat-handling="passthrough";
  port-config-handling="passthrough";
  virtual-ports: {
    dp0: {
      port1: {
        physical= "eth0";
      };
      port2: {
        physical= "eth1";
        vlan=10;
      };
      port3: {
        physical= "eth1";
        vlan=11;
      };
      port4: {
        physical= "eth1";
        vlan=12;
      };
      port5: {
        physical= "eth1";
        vlan=13;
      }
    };
  };
};
};
};

```

2.5.4 Licenses and links

GEPON HSP package	Licence	Link
xCPd	Mozilla Public Licence 2.0	(source code) https://github.com/fp7-alien/xcpd

Table 2.5: Software repository links

2.6 HSP for L0 switch

2.6.1 Overview

The HAL Hardware Specific Part (HSP) for the ADVA Reconfigurable Add/Drop Multiplexers (ROADMs) has been implemented around ROFL library and also by extending the library functionalities not supported yet. Openflow's standard versions do not support optical switches by default. The extensions made to the protocol are based on the circuit switch extensions v.0.3 [7] and are described in more detail in D2.3 (§4.1). UNIVBRIS has built a datapath entity on of the extended ROFL library.

The agent connects to the ADVA's FSP3000 management interface and uses the SNMP protocol to control and fetch characteristics from the device. As shown in the picture below (Fig.7) the agent creates a resource model that is specific to this device and contains a plethora of characteristics, not necessarily relevant to the OpenFlow abstraction. Then the layer above is responsible for picking and doing the mapping only for the characteristics that are relevant to OpenFlow protocol.

The OpenFlow datapath can be installed and executed anywhere in any Linux host inside the network as long as the device has been assigned an IP address for the management interface. However in terms of resilience and also to avoid network latency it is advised to have it as much as possible closer to the device itself. Also the agent is running per device, so if it is needed to control multiple optical switches in the network multiple instances of the agent need to run simultaneously as described in configuration section.

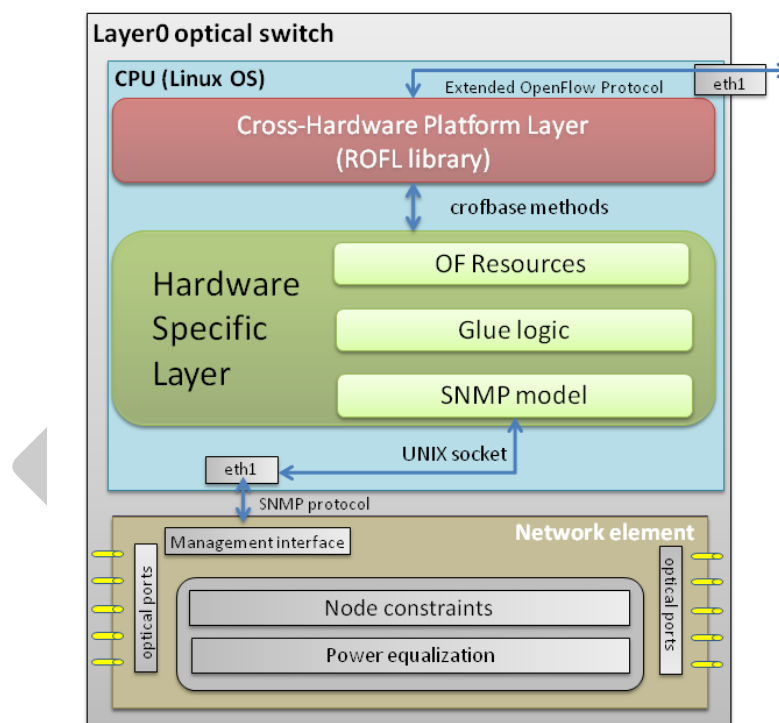


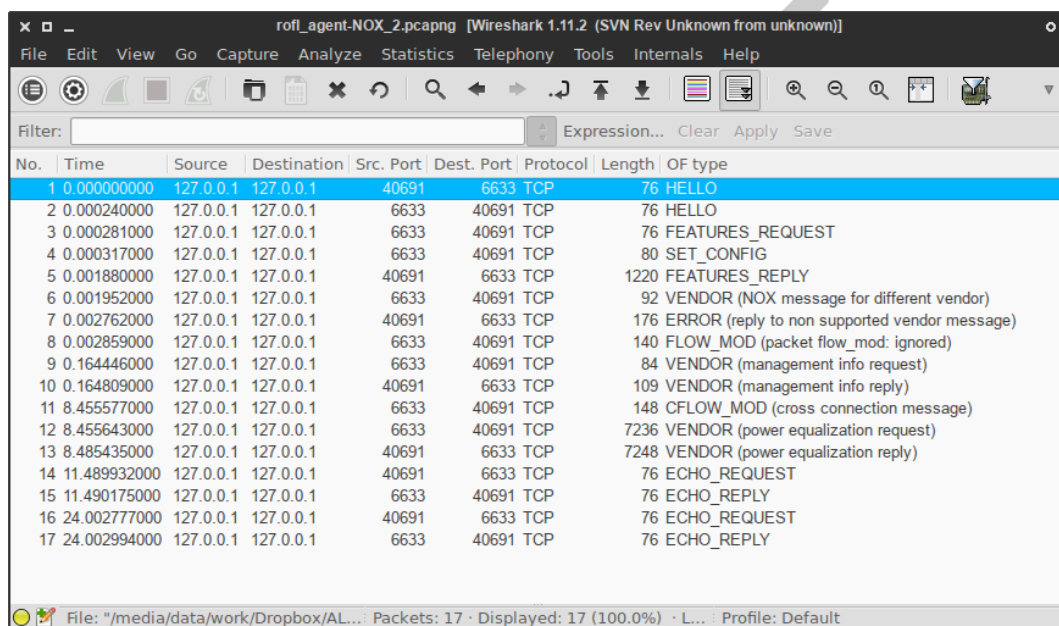
Figure 2.11: ADVA ROADM ROFL based datapath

2.6.2 Supported functionalities

Optical switches compared to packet ones are lacking the notion of packet and there is no visibility in the payload thus no packets can be matched to be sent to the controller. The datapath implementation for the ADVA optical switch has been build to comply with OpenFlow v1.0 and the Circuit Switch Addendum v0.3 [7]. The SDN controller is making decisions in advance all paths in the network have to get provisioned. Hence the OFMatch structure is in a way replaced by the CFlowMod structure which is used to describe the cross-connections inside an optical node.

Due to the fact the OpenFlow version we are using (1.0 with Circuit Extensions v0.3) is not supported by OFtest validation tool [1], we have performed the evaluation of the agent using an version of NOX OpenFlow controller [13] that is able to handle optical devices that support OF1.0 with v0.3 optical extensions. We are providing in the sections below screenshots of the communication between the developed agent and the aforementioned SDN controller. For this reason We have used wireshark network protocol analyser to show the packet exchange between these two entities.

The following figure shows the initial communication and secure channel setup with the exchange of *HELLO* messages. After establishment of the connection the controller requests for features of the switch and gets a modified version of *FEATURES_REPLY* as described below. Also the controller send a *management request* message to get information about the device's management interface. After the establishment of the connection between the controller and the device we attempt to create a cross-connection between 2 ports of the device. So, the controller has to send a *CFLOW_MOD* message followed by a *POWER_EQ_REQUEST* message. After the establishment of the cross-connection the agent replies with a *POWER_EQ_REPLY* message to the controller.



No.	Time	Source	Destination	Src. Port	Dest. Port	Protocol	Length	OF type
1	0.000000000	127.0.0.1	127.0.0.1	40691	6633	TCP	76	HELLO
2	0.000240000	127.0.0.1	127.0.0.1	6633	40691	TCP	76	HELLO
3	0.000281000	127.0.0.1	127.0.0.1	6633	40691	TCP	76	FEATURES_REQUEST
4	0.000317000	127.0.0.1	127.0.0.1	6633	40691	TCP	80	SET_CONFIG
5	0.001880000	127.0.0.1	127.0.0.1	40691	6633	TCP	1220	FEATURES_REPLY
6	0.001952000	127.0.0.1	127.0.0.1	6633	40691	TCP	92	VENDOR (NOX message for different vendor)
7	0.002762000	127.0.0.1	127.0.0.1	40691	6633	TCP	176	ERROR (reply to non supported vendor message)
8	0.002859000	127.0.0.1	127.0.0.1	6633	40691	TCP	140	FLOW_MOD (packet flow_mod: ignored)
9	0.164446000	127.0.0.1	127.0.0.1	6633	40691	TCP	84	VENDOR (management info request)
10	0.164809000	127.0.0.1	127.0.0.1	40691	6633	TCP	109	VENDOR (management info reply)
11	8.455577000	127.0.0.1	127.0.0.1	6633	40691	TCP	148	CFLOW_MOD (cross connection message)
12	8.455643000	127.0.0.1	127.0.0.1	6633	40691	TCP	7236	VENDOR (power equalization request)
13	8.485435000	127.0.0.1	127.0.0.1	40691	6633	TCP	7248	VENDOR (power equalization reply)
14	11.489932000	127.0.0.1	127.0.0.1	40691	6633	TCP	76	ECHO_REQUEST
15	11.490175000	127.0.0.1	127.0.0.1	6633	40691	TCP	76	ECHO_REPLY
16	24.002777000	127.0.0.1	127.0.0.1	40691	6633	TCP	76	ECHO_REQUEST
17	24.002994000	127.0.0.1	127.0.0.1	6633	40691	TCP	76	ECHO_REPLY

Figure 2.12: Wireshark capture of initial communication and circuit port cross-connection

Figure 2.13 shows the messages printed in the agent's console while connecting to the controller. NOX also sends some vendor messages to the agent. The first one is not for ADVA devices and has a different *vendor_id* so it is being ignored by the controller while the second one is requesting some management information and it being handled by the ADVA agent. All messages with their corresponding structures are described in the end of this section.

ROFL library provides to the developers some convenience by providing ready-to-work methods to establish a secure channel to the controller, maintain this connection, handling messages etc. These facilities have been used where possible and in other cases have been extended to meet the device's requirements to ease the development of the agent. Also other OpenFlow messages have been added to allow fetching features and controlling the device and are described in the following paragraphs.

Handle features request/reply: Upon connection establishment with the controller the agent has to inform about its capabilities. The *Features_Reply* message has different attributes compared to the corresponding one for packet switches, especially the structure used to describe the optical ports. We have included attributes like bandwidth and transmission characteristics, as well as the neighbours and their attributes since there is no neighbour discovery protocol, like LLDP in packet, for the optical domain.

Handle CFLOW_MOD messages: CFlowMod messages (OFType 255/22) are used in order to create cross-connections inside the optical node. The agent parses the OpenFlow message received from the controller, discovers the ports of the

```

root@cseegitdesktop:/home/git_admin/ALIEN/rofl-adva/examples/advaswitch# ./advaswitch config10.xml 6633
ADVA ROADM OpenFlow agent just started. (Default OpenFlow port is 6653)

ROFL [main]      Retrieving Openflow attributes from ADVA ROADM...
ROFL [main]      Established connection to SDN controller (localhost:6633).

ROFL [handle_message]:OpenFlow (1) message received: 0 (bytes 2048)
ROFL [send_message]:Send OpenFlow message: 0 (bytes 8)
ROFL [send_message]:Send OpenFlow message: 2 (bytes 8)
ROFL [handle_message]:OpenFlow (1) message received: 5 (bytes 2048)
ROFL [send_message]:Send OpenFlow message: 6 (bytes 1152)
ROFL [handle_message]:OpenFlow (1) message received: 9 (bytes 3072)
ROFL [handle_message]:OpenFlow (1) message received: 3 (bytes 2048)
ROFL [handle_message]:OpenFlow (1) message received: 4 (bytes 6144)
ROFL [handle_experimenter_message]:Vendor id:8992, exp type:8 [ADVA vendor id is: 1094997569]
ROFL [handle_experimenter_message]:NOT_VALID experimenter message

ROFL [send_message]:Send OpenFlow message: 1 (bytes 108)
ROFL [handle_message]:OpenFlow (1) message received: 14 (bytes 18432)
ROFL [handle_message]:Packet FLOW_MOD received from controller but will be ignored.
ROFL [handle_message]:OpenFlow (1) message received: 4 (bytes 4096)

ROFL [experimenter_rcvd]:ADVA vendor message detected. Will call advaswitch handler.
ROFL [handle_experimenter_message]:Vendor id:1094997569, exp type:4 [ADVA vendor id is: 1094997569]
ROFL [handle_experimenter_message]:OOE_MGMT_INFO_REQUEST message detected
ROFL [handle_mgmt_info_req]:unlock MUTEX
ROFL [handle_mgmt_info_req]:Print experimenter message created. cmemory(0x8635800) somem()[0x8634070] len[41] data
a.first: 0x8634070 data.second: 41 data
0x8634070 : 01 04 00 25 00 00 00 00 41 44 56 41 00 00 00 00
0x8634080 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x8634090 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
ROFL [send_message]:Send OpenFlow message: 4 (bytes 37)

```

Figure 2.13: OpenFlow agent console startup and messages exchanged

switch that are defined and finally attempts to perform the cross-connection by sending the appropriate SNMP commands.

In order to facilitate the control of the device and enable the SDN controller to fetch more useful information we have implemented a number of new OpenFlow messages that are not part of the protocol neither the Circuit Switch Addendum. In order to achieve that we have used the OFPT_VENDOR (4) experimenter message type and a special header after the usual OpenFlow one to differentiate between the experimenter messages defined.

```

/* Vendor messages structs - ADVA specific */

struct ooe_header {
    struct ofp_header header;    // openflow header
    uint32_t vendor;            // vendor id
    uint32_t type;              // message type (OOE_message type)
    uint8_t data(0);            // message payload
}

```

The VENDOR messages we are using for the ADVA ROADM have the same *vendor id* (OOE_VENDOR_ID), however what differentiates these experimenter messages is the 'type'. The values of this field are listed in the following enum and are explained below.

```

/* ADVA specific VENDOR experimenter messages */
enum ooe_type {
    OOE_SWITCH_CONSTRAINTS_REQUEST,    /* switching constraints message */

```

```

OOE_SWITCH_CONSTRAINTS_REPLY,      /* switching constraints message */
OOE_POWER_EQ_REQUEST,              /* power equalization */
OOE_POWER_EQ_REPLY,                /* power equalization */
OOE_MGMT_INFO_REQUEST,             /* management info msg */
OOE_MGMT_INFO_REPLY                /* management info msg */
};

```

Power equalization messages: When creating a cross-connection in the optical switch we also need to perform power equalization between the connected ports to achieve the actual establishment of the lightpath. Thus, after the CFlow_Mod the controller also sends a ooe_power_equalization message to the agent. The agent then needs to parse and perform the power equalization between the connected

Switching constraints: This message provides information to the network administrator about the cross-connections that can be made in the optical node. The switch does not have the ability to direct the optical signal from any input port to any output port. Each input port can only be connected to a list of output ports of the switch. The reply to the OOE_SWITCH_CONSTRAINTS_REQUEST message returns exactly this information to the controller.

Management info: The *mgmt_info* message is used for communicating information relevant to the management interface (SNMP) of the device to the controller. The attributes we push to the controller are shown in the structure below.

```

/* ADVA management info extensions */
struct ooe_mgmt_info {
    struct ooe_header    header;          // vendor message header
    uint64_t             dpid;           // datapath id
    uint32_t             snmp_ip;        // address of SNMP agent
    uint32_t             snmp_port;     // port no of SNMP agent
    uint8_t              pad;           // pad
    uint8_t              num;           // number of bytes in 'community' array
    uint8_t              snmp_community[0];
};

```

2.6.3 Requirements, installation and configuration

As described in the sections above the ADVA ROADM OpenFlow agent is using the SNMP management channel in order to communicate in the device. The code to make these calls is supplied by ADVA, is proprietary and not publicly available by the company. Consequently it cannot be distributed as part of the datapath implementation. The aforementioned code is responsible for sending the actual command to the device using the SNMP channel. Thus, the agent on receiving a message from the controller it will parse it, decide what are the actions that it needs to take and finally, if necessary, send a message either/or both to the controller and the device.

At the end of this section we are providing the link to source code of agent's implementation which includes the extensions we have developed for OpenFlow protocol as well as the methods for marshalling and unmarshalling the OpenFlow messages. The following text explains how to build and run the datapath in a modern Linux distribution that offers the essential build tools (autoconf, automake, libtool..). Also the user will with the addition of *libxml2-dev* if not already installed. The instructions are similar to the ones for building the *rofl-core* library. After installing the required build tools and libraries you should run the following commands:

```

./autogen.sh
./configure
make
make install (optional)
export LD_LIBRARY_PATH=/path/to/adva-rofl-dp/adva-agent-rofl (ADVA library folder stub)

```

If everything goes well then you should have the datapath binary in the following folder:

```
../adva-rofl-dp/examples/advaswitch
```

and you can execute the agent by pointing to your configuration file as explained below.

The agent is implemented in such a way so that is able to manage one switch at a time. In order to manage more than one a device at a time multiple instances of the same agent need to my executed simultaneously using different configuration files. The contents of such a file are shown at the snippet below.

```
<config>
  <host>10.0.34.10:161</host>
  <community>private</community>
  <trap>0.0.0.0:1620</trap>
  <poll>0</poll>
  <cpreload>10</cpreload>
  <openflow>tcp:10.0.34.133</openflow>
</config>
```

The management IP (*host* attribute) is given to the agent so it will connect to the desired optical switch. Furthermore, the IP at which the OpenFlow controller is installed and listening for new devices, is given to the agent (*openflow* attribute). The default port the agent tries to bind with the controller is set to 6653 since starting with OpenFlow version 1.4 this port number is defined for OpenFlow control channel communication and any other port numbers used so far should be considered deprecated. However, the user can define his own port number (if needed) from the command line when executing the ADVA agent. An example of running the controller is given below:

```
./advaswitch config10.xml [6633]
```

2.6.4 Licenses and links

Layer0 Switch	Licence	Link
OF Datapath for ADVA FSP3000 (UNIVBRIS)	Mozilla Public Licence 2.0	(source code) https://github.com/fp7-alien/adva-rofl-dp

Table 2.6: Software repository links

3 Lessons Learned

During the phase of implementation of the Hardware Specific Parts, the teams gathered experiences and opinions related to porting xDPd/ROFL to a given ALIEN platform as well as new recognized challenges related to enabling OpenFlow over the hardware platforms. This section summarizes lessons learned, grouped by a specific platform.

3.1 EZappliance HSP

xDPd and its AFA interface framework perfectly fitted EZappliance HSP development requirements. It minimized required coding to have the OpenFlow protocol controlling the platform. The xDPd framework and corresponding ROFL library were used without any major problems.

The minor problem experienced during HSP implementation were:

- In the beginning, there were no detailed guidance about creating a new driver in xDPd. The corresponding guide was added to the xDPd project later on, in xDPd version 0.3.
- Compilation failed in some software environments, depending on a specific OS and 32-bit vs 64-bit systems (i.e.: Debian on 32-bit platform). The problem was solved in xDPd version 0.3.
- In xDPd version 0.3, a possibility to include EZappliance HSP specific configuration parameters was hardly available. Other means for storing these values had to be implemented. Later, this issue was solved in xDPd version 0.4.
- The EZappliance HSP does not require root privileges whereas xDPd framework requires this rights during the xDPd instance launch.
- The EZappliance HSP uses a specific hybrid implementation of the hardware driver which uses AFA with the ROFL pipeline, so the EZappliance Specific Part developer needed to understand well GNU/Linux pipeline implementation and its usage.
- The EZappliance driver hybrid implementation makes it more difficult to migrate xDPd/ROFL from version 0.3 to version 0.4 because all ROFL OpenFlow pipeline source files had to be located properly and linked within EZappliance driver directory.
- A few software bugs have been discovered during the EZappliance development phase:
 - in BadMessage error message generation (solved in xDPd version 0.3)
 - in handling driver extra-parameters from configuration file (solved in xDPd version 0.4)
 - logging setup with debug level in the config file (marked as very low priority problem in xDPd project and currently skipped)

All bugs have been efficiently corrected by the xDPd development team.

3.2 NetFPGA HSP

Usage of NetFPGA requires its proper installation in hosting operating system. It is possible to use not fully correct installed software bundle, but in such a situation not all functionalities will be realized properly and some features may be not active and even cause crash of software or whole system. For this card, its developers prepared software environment which was fully tested and is supported on Fedora 14. In the Internet there are available hints how to install NetFPGA cards on different and newer operating systems, but some of them are defective and led to improper configuration, which seems to be valid (compilation of code is succeed), but detailed tests for particular functionalities (which are available in NetFPGA source code) can fail.

Basis installation of NetFPGA driver in operating system creates network interfaces in system for each physical port. By default, they have the same default MAC addresses starting from the same value. When more than one NetFPGA card is

used in the same L2 segment based on Ethernet switch, all hosts (i.e. ports of NetFPGA cards) have to use different MAC addresses. In this case, the user (developer) has to change default MAC addresses to custom ones. In case of usage of the same MAC addresses by different NetFPGA cards, the problems do not have to occur in each case -- in some situation (mainly with traffic with low intensity), switch can treat the same address on different ports as a re-plugged devices and mechanisms from higher layers can mask gaps in communications. Due to this seeming improvements, real problems are difficult to be identified. In case of intensive traffic from/to two hosts with the same MAC address, the performance is significantly reduced and problems with proper functionality are visible immediately.

Complementary situation consists on usage of more than one NetFPGA cards in the same host -- their interfaces will obtain automatic names (in form nFX, where X starts from 0 and represents numbers of physical plugs), but in this case necessity of proper script and parameters adaptation is obvious. Users also have to remember to reserve higher amount of RAM memory for next NetFPGA cards.

3.3 DOCSIS HSP

As a result of the DOCSIS HSP implementation several valuable lessons were learned, which can be beneficial for future implementations with similar characteristics based on a proxy. The most relevant lessons are highlighted below.

- ROFL libraries have made easier the conception of the proxy and its development.
- As the DOCSIS network uses VLAN_VID tags to identify traffic for each cable modem, it has been required to implement multiple tables, which implies the use of OF version greater than 1.1 at the aggregation switch for supporting incoming VLAN traffic handling. OF1.2 was available at ROFL when the implementation phase begun.
- Orchestrating the FlowMod messages has been one of the biggest challenges of the proxy development. Having several switches with DOCSIS access network behaving transparently as a unique one is a great challenge when trying to orchestrate rules that have either the in-port wildcarded or the output is OFPP_ALL.
- The usage of METADATA and OF1.2 is required as QinQ traffic handling is a must. For that, the outermost tag is removed and its value is written into the metadata field. This way, it is possible to have both VLAN fields available for matching (one in the METADATA field and the innermost in the VLAN field, as it has not been removed).
- The WRITE_METADATA was not properly working at xDPd 0.4 version and it was reported to xDPd development team who fix the implementation releasing patch for xDPd 0.4.
- Obtaining the MAC address from the DPID didn't work properly when using xDPd 0.4, and it was also reported to xDPd team.
- For testing and benchmarking the DOCSIS ALIEN, it must be taken into account that, although it behaves as a OF switch, it is an access network device and any comparison should be done against other access network devices (and not against a pure vendor switch) to obtain the proper figures from this comparison.
- A validation of an OpenFlow device cannot be done only by using the OFtest tool and it has to be validated under real environment and real rules, as sometimes OFtest can provide misunderstanding results.

3.4 GEPON HSP

Several important lessons were learned from the UCL implementation. The experience of working with the ROFL libraries was overall a good one and these libraries were a great benefit to the project. Without ROFL the implementation would be much more difficult than it was already.

A small number of bugs and omissions in the ROFL 0.3 libraries were found and corrected. These have been merged into the devel-0.3 branch.

The solution of using VPN tags to "virtualise" the GEAPON/ONU is simple to state, however, in practice the implementation was not so straightforward. Every OpenFlow message needed a solution. The usual case this required "matches" and "actions" to be translated. If, for example, a match was on "port 2" in the virtualised switch this needs to be translated to a match on port and possibly VLAN tag on the real hardware. Similarly an action of "output to port 2" might translate to "output to port 1 and tag with tag 11". Action lists could, therefore, become longer.

With regard to flowMod it became necessary to have the upper level controller (xCPd) to maintain a list of translated and untranslated flowMods. In this way a request, for example, to delete flowMods matching a pattern could be matched against the list of flow-mods and then translated to several strict deletion. Statistics replies require similar modifications.

In general, implementing an entire protocol requires consideration of all the corner-cases in that protocol. Some of these were not considered in our initial design. For example, an action to output to all ports or to flood to all ports needed to be translated to several tag then output matches.

One problem which proved insurmountable was that of using VLAN tagging within the system. If a user wished to use a VLAN tag through the virtualised switch then this tag is overwritten by the system. Similarly, if an unknown VLAN tag arrives it would be impossible to know from which virtualised port it arrived.

3.5 L0 switch HSP

UNIVBRIS built the OpenFlow agent for the ADVA optical switch using the ROFL library v0.3. The library offers a great support for the developer and facilitates a lot the development time for building a fully operational agent. We did not have to build the agent from scratch since the library provides easy to use methods for secure channel establishment and keeping alive the connection between the agent and the controller. If we divide the datapath implementation in protocol message handling and control channel handling the latter is completely implemented by ROFL methods.

The message handling was done using the methods provided by the library and namely the *crofbase* class. In the case of the Optical switch there some methods missing from the OpenFlow messages since the protocol has been extended to accommodate these devices. The library's support has included the standard OpenFlow versions. Thus, in order to develop a datapath for a device using the circuit extensions of the protocol [7] we had to extend the methods provided by *crofbase*. For example CFlowMod messages and modified versions of FeaturesReply had to be added to the library for the purpose of the optical agent implementation.

The process of adding these additional messages is streamlined which facilitates the development of the OpenFlow agent. Furthermore, there is support for experimental/vendor messages that we used to the ADVA optical switch. Surely, it requires that the developer will invest some effort to get hold of the many different components of the library. The support of different versions of the protocol add to the complexity of the library. In general, the learning curve is smooth and the benefits apparent compared to designing and implementing all the required functionalities starting from a white paper.

4 xDPd/ROFL Improvements for ALIEN HSPs

There are four major improvements in ROFL and xDPd:

1. Renaming AFA to HAL
2. Driver specific initialization parameters
3. Increased code reuse among drivers
4. Increased code quality

The first two deal with specific changes while the latter are more general. At first there is a refactoring of the code to rename AFA to HAL, to have a more common terminology. In the end this led to a shorter learning curve of the code and the existing drivers. Further parameters were needed to initialize some of the new hardware platforms. Therefore the HAL was extended to pass additional parameters to the driver, which is exposed for example in the default management plugin (i.e. config). A user can specify the driver-extra-params in the system section of the config file. In a more general view, the multiple platforms added in Alien led to an increased code reuse. Most of the new platforms use the pipeline provided by ROFL, thus many platforms had to copy parts of the original code to create the HSP. Better integration of the generic packet classifier in the pipeline was possible due to Alien. Drivers can benefit from improvements in ROFL and xDPd because of this. Increased code quality and lots of bug fixes as a result of the interaction with a developer community were an outcome of Alien.

DRAFT

5 Conclusions

In this document HSP prototypes have been presented, which allow an OpenFlow control over a very diverse set of ALIEN platforms. Some of these platforms are considered as for the first time enabled to participate in the SDN experiments (i.e.: DOCSIS, GEAPON) or the first open implementation provided to the community (i.e.: EZappliance).

The critical mission of the ALIEN project was to design, implement and validate the Hardware Abstraction Layer concept [D2.2] which must be applicable to a wide range of network platforms. Taking this into account, all HSP software packages have been implemented as proof-of-concept prototypes to validate if the ALIEN HAL design is flexible enough and easily adaptable to devices with very different set of management and configuration requirements. From this point of the view it is important to analyse how different HSP prototypes fit into the HAL architecture and how xDPd/ROFL realizes platform-independent components of the HAL architecture. This analysis is introduced in the deliverable D2.3 [D2.3], but focused mainly on a functional level only. This document extends further the analysis from D2.3 and specifically focuses on a software level in relation to the usage of ROFL (the library providing OpenFlow-related functionality to controllers and datapath elements) and xDPd (a framework enabling OpenFlow datapath element for different platforms; based on the ROFL library).

Figure 5.1 presents how all HSP prototypes use xDPd, ROFL and HAL interfaces.

Platform	xDPd usage	ROFL usage	HAL interface(s)
EZappliance	Yes	Yes	AFA
NetFPGA	Yes	Yes	AFA
Cavium Octeon	Yes	Yes	AFA and HPA
DOCSIS	No	Yes	(croftbase methods)
GEAPON	Only as library	Yes	(croftbase methods)
L0 switch	No	Yes	(croftbase methods)

Figure 5.1: Usage of xDPd and ROFL

As one can easily spot from the figure, all HSPs use the ROFL library to provide an OpenFlow endpoint functionality, AFA/HPA interface definitions and the OpenFlow pipeline. xDPd is used by several platforms (not all of them) as a software framework which helps enabling OpenFlow in alien equipment as defined in the HAL architecture. The situation of DOCSIS, GEAPON and L0 switch is different, mainly caused by different timelines set to xDPd and ALIEN development processes. When the HSP development process started, xDPd has been focused on Ethernet switch programmable devices. For all platforms which could not be represented as a single switch but were composed of a set of devices (i.e. GEAPON, DOCSIS and L0 switch), the xDPd software stack was missing proper configuration capabilities, e.g. to express topology of interconnected devices. For that reason some development teams started implementing OpenFlow for their platforms, basing directly on the ROFL library. This approach has an impact on a possibility of the usage of HAL interfaces like AFA and HPA. For these platforms, development teams decided to use ROFL croftbase interface (defined in `rofl-core/src/rofl/common/croftbase.h`) which is a direct OpenFlow protocol interface. AFA and croftbase have many similarities because AFA is also made on the ROFL croftbase foundation. In contrast to ROFL croftbase, the AFA interface is hiding OpenFlow protocol mechanisms and thus is much better for the usage during the hardware driver implementation. For this reason, the description of the ROFL croftbase interface was skipped in the HAL specification [D2.2] and also it has not been documented in the HAL implementation report [D2.3]. Last but not least, the hardware specific code based on croftbase methods can be ported to the xDPd framework, because of AFA and croftbase semantic similarities. Overview of both approaches to HAL implementation is presented in Figure 5.2.

With this deliverable WP3 is officially finalizing its activity and the HSP prototypes development, however each HSP prototype has still long list of features identified which should be implemented in order to fully support OpenFlow specifications. Figure 5.3 presents summary of the WP3 activity in the context of each HSP.

The documentation for each HSP has been prepared in a form of deliverables (D2.3 and D3.3), software prototypes have been released and validated. The WP3 activity has also performed its own demonstrations during the FIA2014 and TNC2014

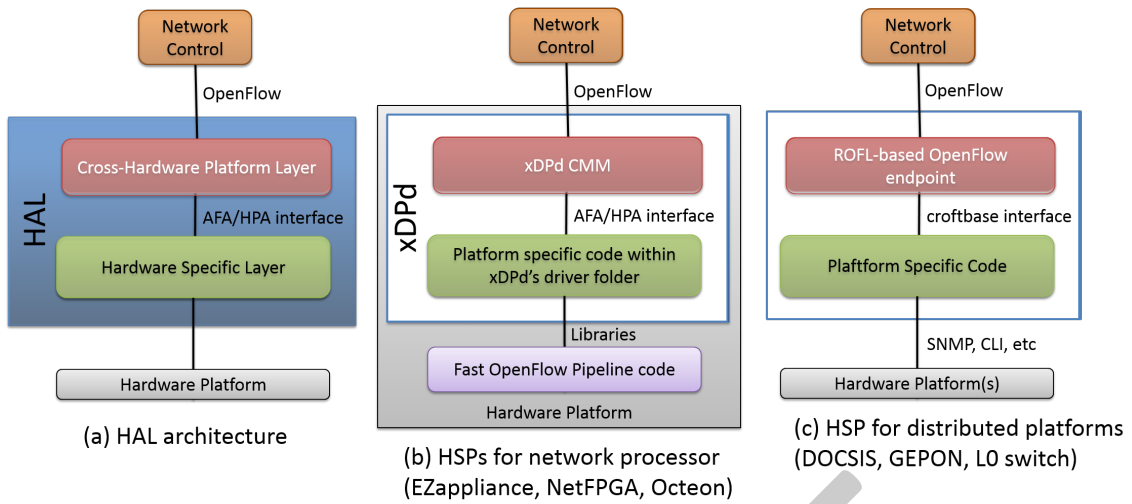


Figure 5.2: HSP implementation approaches (b) and (c) in the context of HAL architecture (a)

Platform	Partner	Documentation	Software	Validated (oftest)	Demonstrated
EZappliance	PSNC	Done	Done	Done	Yes
NetFPGA	PUT	Done	Done	Done	No
Cavium Octeon	BISDN	Done	Done (no publicly available)	Done	No
DOCSIS	UPV/EHU	Done	Done	Done	Yes
GEAPON	UCL	Done	Done	Done	No
LO switch	UNIVBRIS	Done	Done	Done (meesages dumps)	No

Figure 5.3: Development of HSP prototypes - summary

conferences using EZappliance and DOCSIS HSPs. Further validation and demonstration of all HSPs will be performed within the OFELIA environment in WP5 activity ("Experiments on OFELIA").

References

- [1] Floodlight project, OFtest validation tool. <http://www.projectfloodlight.org/oftest/>.
- [2] Future Internet Assembly 2014. <https://www.fi-athens.eu/>.
- [3] Object Management Group, Corba specifications. <http://www.omg.org/spec/index.htm>.
- [4] Revised OpenFlow Library. <http://www.roflibs.org/>.
- [5] Terena Networking Conference 2014. <https://tnc2014.terena.org/>.
- [6] The OpenFlow eXtensible DataPath daemon project. <http://www.xdpd.org/>.
- [7] Extension to the OpenFlow Protocol in support of Circuit Switching. http://archive.openflow.org/wk/images/8/81/OpenFlow_Circuit_Switch_Specification_v0.3.pdf, 2010.
- [8] Deliverable D3.2: Specification of hardware specific parts. <http://www.fp7-alien.eu/files/deliverables/D3.2-ALIEN-final.pdf>, 2013.
- [9] Deliverable D2.2: Specification of Hardware Abstraction Layer. <http://www.fp7-alien.eu/files/deliverables/D2.2-ALIEN-final.pdf>, 2014.
- [10] Ł. Ogirodowczyk et al. Hardware abstraction layer for non-openflow capable devices. *The TERENA Networking Conference (TNC)*, May 2014.
- [11] Bartosz Belter et al. Hardware abstraction layer as an sdn-enabler for non-openflow network equipment. In *European Workshop on Software Defined Networking (EWSDN)*, September 2014. Accepted for publication.
- [12] N. Katta, O. Alipourfard, J. Rexford, and D. Walker. Infinite cacheflow in software-defined networks. In *HotSDN Workshop*, August 2014.
- [13] M. Channegowda et al. Experimental demonstration of an openflow based software-defined optical network employing packet, fixed and flexible dwdm grid technologies on an international multi-domain testbed. *Opt. Express* 21, pages 5487--5498, 2013.

Acronyms

CMTS	Cable Modem Termination System
DOCSIS	Data Over Cable Service Interface Specification
FPGA	Field Programmable Gate Array
GEPON	Gigabit Ethernet Passive Optical Network
HAL	Hardware Abstraction Layer
HSP	Hardware Specific Part
IDE	Integrated Development Environment
IP	Internet Protocol
IPC	Inter-Process Communication
OF	OpenFlow
OLT	Optical Line Terminal
ONU	Optical Network Unit
OUI	OpenFlow User Instance
PCI	Peripheral Component Interconnect
ROADM	Reconfigurable Add/Drop Multiplexer
SNMP	Simple Network Management Protocol
TOP	Task Optimized Processor
VLAN	Virtual Local Area Network

DRAFT