This paper describes a viable and experimentally-tested way forward for augmenting legacy network elements with software-defined networking control. Following and implementation-driven approach we have explored the possibilities for adding SDN-based interfaces to devices which are not compatible with OpenFlow. OpenFlow is arguably a leading control-place protocol for the upcoming generation of operator networks. Yet not all domains will be equipped from the very beginning with compatible supporting frameworks. To address this gap, we introduce the Hardware Abstraction Layer (HAL) for non-OpenFlow capable devices which tackles this problem. We discuss the advantages of the proposed approach and explain how a HAL-based architecture can support different classes of network devices.

Keywords
OpenFlow, Software Defined Networking, Hardware Abstraction Layer, Network Virtualization
1. Introduction

Software Defined Networking (SDN) [McKeown 2009] capitalizes on splitting the data and control planes and offers researchers new opportunities for network control and management. Without going into full details in this paper, a logically-centralized controller that sits in the network operation system layer is capable of taking global decisions about the data plane traffic control. Although the OpenFlow protocol [McKeown 2009] is now widely used for communication between a logical network controller and network devices, a large array of network elements do not support OpenFlow natively, be that due to the nature of the forwarding paradigm or because of legacy reasons. In particular, we observe that OpenFlow was originally designed primarily for networks dominated by wired Ethernet interconnections and, as such, does not support, for example, circuit-switched and wireless network platforms.

The Hardware Abstraction Layer (HAL) [ALIEN 2014][HAL 2014], defined and developed within the FP7 ALIEN [ALIEN 2014] project, is an experimentally-verified concept for describing network device capabilities and controlling the forwarding behaviour of all OpenFlow and non-OpenFlow capable hardware throughout a network. The main contribution of this paper is the introduction of the HAL architecture, a modular design with reusable components which has been ported on several hardware platforms, as we describe later in this paper.

Abstraction layers are well known tools in operating systems development that allow direct access to the hardware through drivers which hide hardware complexity and implement platform-specific details. In operating systems such as Linux and Android, unified and standardized sets of routines allow one to write architecture-independent and portable applications. Similarly, in the SDN domain, HAL hides the hardware complexity as well as technology and vendor-specific features, thus presenting a unified abstraction layer for an OpenFlow controller.

The remainder of this paper is organized as follows. Section 2 introduces the HAL architecture, while Section 3 details the HAL implementation. Section 4 discusses the advantages of the HAL approach, and Section 5 concludes the paper pointing to future work items.

2. Alien devices categorization

In the ALIEN project, we faced with heterogeneous network devices which are “alien” to the OpenFlow protocol (they don’t support OpenFlow protocol natively): NetFPGA cards, EZChip NP-3 network processors (EZappliance), Cavium OCTEON Plus AMC network processor (a module in ATCA systems and DELL switches), ADVA optical switches, DOCSIS and GEPON systems [D3.1 2013][D3.2 2013].

Heterogeneous network devices, we met in the ALIEN project, make difficulties in defining one common view of device abstraction from the SDN controller and the OpenFlow protocol point of view. Such abstraction should allow control plane to steering the datapath elements in the SDN manner. Although all network devices differs in technology, data plane forwarding methods, configuration and management interfaces etc. some of the pieces of equipment have properties in common. The first criterion of network devices division is openness of hardware for datapath programming. Some boxes are closed and allow only configuration via standardized interfaces (e.g. CLI, SNMP), others like devices with network processors allow direct access to packet processing programming. Only limited set of devices can be reconfigurable on the fly while others work with static configurations. The ALIEN project goal was to take into account heterogeneity and constraints of different network devices from the one side - and common hardware themes from the other - to design unified Hardware Abstraction Layer which hides hardware complexities and vendor specific attributes and exposes OpenFlow protocol to the SDN controller.

Common hardware themes of abovementioned hardware presented in [12] allowed us to design the sub-layer of the HAL common for all platforms. Some common functional modules of the hardware agnostic part of the HAL may be reused for different hardware platforms, also for platforms not mentioned here.

In the following sections the Hardware Abstraction Layer adaptation will be presented for three groups of network devices:

- **Optical devices.** OpenFlow protocol is limited to only Ethernet abstraction so in case of optical devices the abstraction layer must be adopted to meet OpenFlow extension requirements for supporting circuit switched networking (see section 3.1.1).
- **Point to multi-point devices.** For these types of devices (e.g.: GEPON, DOCSIS) with “head” and “tails” some kind of orchestration is necessary for exposing several devices as a single OpenFlow enabled device through HAL (see section 3.1.2).
- **Programmable platforms.** This group refers to network devices which allow their data plane to be programmed to perform packet processing. In case of e.g. network processors (e.g.: EZchip NP-3, Cavium Octeon, NetFPGA), it is possible to implement OpenFlow pipeline directly into hardware (see section 3.1.3).

### 3. Hardware Abstraction Layer: Architecture

HAL follows a modular architecture [D2.2 2014] with unified interfaces for different types of the network equipment. The main tenants in the HAL architecture are:

- HAL resides between the OpenFlow controller and the non-OpenFlow network device. This means that the OpenFlow controller can control each data path element in the network only through HAL.
- HAL components are reusable. In particular, HAL comprises two layers: the first one is hardware-dependent while the second one is independent of the hardware specifics. This approach is similar to the one widely adopted by modern programming languages (e.g. Java), where the platform-dependent (JVM) and platform independent (java class files) parts have been identified. This approach leads to smaller code, better troubleshooting and software management.
- HAL is portable, thus allowing us to take advantage of the operational model advanced by OpenFlow on various, currently non-OpenFlow-enabled platforms.

As depicted on Figure 1, HAL comprises two parts: the Cross-Hardware Platform Layer (CHPL) and the Hardware Specific Layer (HSL). CHPL enables build-in virtualization and OpenFlow mechanisms that are independent of the underlying hardware platform. HSL is a set of hardware drivers realizing primitive network instructions, specific to different hardware platforms. These two layers are connected to each other with two interfaces, 1) Abstract Forwarding API as an interface to communicate with hardware driver, and 2) Hardware Pipeline API for hardware platforms that use the OpenFlow datapath implementation provided by Cross-Hardware Platform Layer.

The OpenFlow mechanisms, which are available in Cross-Hardware Platform Layer, are OpenFlow protocol endpoint and OpenFlow pipeline. The OpenFlow endpoint component establishes a connection channel to the controller to exchange and parse OpenFlow protocol messages. The OpenFlow endpoint can use any OpenFlow
version number in the communication with the controller and converts the abstraction of any OpenFlow version to AFA interface data model that is a superset of all OpenFlow versions features. The AFA data model can be easily extended to handle new header matches, new matching algorithms and new packet processing actions.

The CHPL pipeline component provides performance efficient OpenFlow pipeline implementation which processes packet abstractions instead of a real network packet. The packet abstraction contains a reference to a real network packet stored in memory of the network device. Any modification of packet abstraction object within the CHPL pipeline are reflected in changes of a real packet in the device.

A virtualization component is also foreseen within the architecture to provide virtualization capabilities to HAL-compatible devices. This component enables flowspace-based slicing among many controllers managing several virtual node instances using different versions of the OpenFlow protocol and it is interacting with the Network Management System (NMS) that configure the virtual network tenants.

In order to initialize Cross-Hardware Platform Layer, information about network device(s) must be discovered by Hardware Specific Layer. The HSL discovery component provides information like a list of devices working together as a single hardware platform instance, device access information, a list of all network ports and their characteristics (e.g. transmission technology, transmission speed, operational status, etc.) and the internal hardware platform topology (e.g. how all devices within a hardware platform instance are interconnected).

In some cases, the hardware platform is composed of multiple hardware components acting independently, but controlled centrally by a single SDN controller (e.g. DOCSIS, GEPON). The HSL orchestration component goal is to send configuration commands to all hardware components that must be engaged in the request handling in a synchronized, ordered and atomic fashion. The orchestration process must ensure successful application of request on all devices and be able to recover from configuration failures by initial state restoration of all the hardware components.

The translator component in Hardware Specific Layer is responsible for the translation of data and action models used in Cross-Hardware Platform Interfaces (mostly OpenFlow-based) to device’s protocol syntax and semantics. Translator acts as a middleware between OpenFlow switch model and underlying physical device. In particular, the module is responsible for translating port numbering, flow entries and packet actions from OpenFlow switch model into platform specific commands, processor instructions or configuration file modifications.

The initial HAL architecture has been described in [3], the publicly available publication released by the ALIEN Consortium. The HAL implementation follows naturally from this model. Modular software design allows the separation of the evolution of the OpenFlow endpoint (a common part for all hardware platforms) and hardware specific drivers which communicate with the network equipment. The HAL architecture makes the implementation of OpenFlow protocol easier for any new device that is “alien” for this technology.

3.1 HAL architecture adaptations

3.1.1 Programmable Network platforms

Programmable Network Platforms represent a set of network equipment containing a re-programmable hardware unit (NPU or FPGA) that can be adapted to a wide range of network processing tasks (i.e. packet switching, routing, network monitoring, firewall protection, deep packet inspection, load balancing, etc.). These platforms allow for expressing packet processing control/service logic, using a programming language, in form of compiled source code which can be implemented indefinitely on a single hardware unit.

Currently, there are many programmable network platforms available in the market produced by companies like EZchip, Marvel, Cavium, Broadcom, Freescale, PMC-Sierra, Tilera. Each company provides programmable processors using quite different processor architectures in terms of microcore types (i.e.: general core like in CPU, task optimized core), organization (e.g.: homogeneous cores loosely assigned to tasks, strict pipelines of heterogeneous cores), add-ons (i.e.: hardware accelerators for parsing, patterns matching, cryptography, packet classification, querying, etc.) and memory accessibility (e.g.: standard CPU cores with ASIC network enhancements, task optimized NPU cores). This heterogeneity of network processors is a challenge for making common assumptions and design of Hardware Abstraction Layer. Here, we would like to present an example of HAL adaptation for a network processor which was developed using EZappliance platform.
**EZappliance platform**

The heart of the EZappliance platform [EZ 2014] is EZchip NP-3 network processor (see Figure 2), which is a fully programmable chip which enables flexible parsing, classification, packet header manipulation and switching of pass through packets. It is the place when packet processing through OpenFlow pipeline must happen in order to utilize the full performance of processor. Unfortunately, the Cross-Hardware Platform Layer (CHPL) pipeline, handling packet abstractions, cannot be used because NP-3 processor has very strict time constrains for packet processing and cannot store the packets anywhere inside. For this reason, a new implementation of the OpenFlow pipeline for NP-3 task optimized cores with usage of the EZchip assembly language was developed from the scratch.

![EZappliance diagram](image)

**Figure 2 HAL adaptation for EZappliance network processor platform**

The NP-3 processor is accompanied with a standard CPU foreseen for the deployment of control and management plane functionalities. The standard CPU was used to deploy both layers of HAL: CHPL and HSL. As mentioned above, CHPL pipeline is not used, thus Hardware Specific Layer (HSP) could be controlled by Cross-Hardware Platform Layer through the AFA interface only. HSL for EZappliance device supports discovery and translator functionalities. The discovery functionality is based on automatic retrieving of information about all data plane ports, its attributes and statuses. In the case of EZappliance, which is a standalone device, topology discovery is not required (for the same reason, HSL for EZappliance doesn’t contain the orchestrator functionality). The most complex part of HSL is an implementation of the translation functionality which transforms OpenFlow-based AFA messages into memory structures located within NP-3 network processor. The NP-3 memory structure are accessed with usage of EZdriver provided by EZchip company. The semantics used for EZappliance memory structures is quite similar to OpenFlow (e.g. memory contains a structure with flow entries) but syntax is mostly different (i.e.: property binary coding of packet matching and actions). The translation in the HSP is stateless because of very close semantics to OpenFlow.

**NetFPGA cards**

NetFPGA cards [NETFPGA 2014], similarly to NP-3, can be treated as a programmable packet processors. They have four 1Gbps Ethernet interfaces (or 10Gbps in a newer version). Both card versions can work as a separate network nodes, however, typically they are mounted in a PC and they are integrated with an operational system via PCI or PCIe bus. The program, which is running in the FPGA chip, has to be prepared in Verilog or VHDL language. Due to specific architecture and technology, its performance is very high and it is willingly used by designers of different prototypes. In case of ALIEN project, NetFPGA cards are used as a hardware platform. The OpenFlow pipeline is almost fully implemented in the NetFPGA logic which offers much better performance characteristics comparing to usage of the CHPL pipeline, which has been deployed in the PC operating system as part of HSP. In the NetFGPA HAL realization (see Figure 4), the CHPL pipeline is used as full OpenFlow featured but slower implementation which process packets not handled by hardware pipeline due...
to OpenFlow features missing in the current hardware pipeline implementation. The CHPL for NetFPGA cards is located in the PC operating system and has connection with network controller using OpenFlow protocol via NIC of PC. The HSP for NetFPGA also is realizing discovery and translation. The translation functionality is responsible to recoding of OpenFlow flow entries into a binary representation accepted by hardware OpenFlow pipeline in NetFGPA card.

Figure 3 HAL adaptation for NetFPGA cards

Proper control information (flowmods) are stored in hardware chip of NetFPGA card and all possible flow actions (packet forwarding, dropping, etc.) are realized in hardware chip, only few first packets per each flow or flows which cannot be served in hardware pipeline are performed in software realization of HSP.

Programmable processors are ideal hardware platforms to introduce and validate new networking concepts. To take advantage of this possibility, in the ALIEN project, dynamic adaptation of network node capabilities is investigated in order to “teach” new protocols to a data path element with new processing actions which later could be added to the OpenFlow protocol action set.

3.1.2 DOCSIS system

The Access Network (AN) provides the connectivity between the home/business customer's location (i.e. subscribers) and the operator’s premises. This part of the network is known as the last mile and it is considered as the bottleneck in terms of bandwidth and the most expensive part of the operator's network. There are several technologies currently used in commercial deployments depending on the available media, such as xDLS (copper), DOCSIS (cable) or GPON (fiber). In order to deploy the system in the most cost-effective manner, this media is shared by a set of subscribers. As a result, the bandwidth sharing is one of the goals of any of those AN technologies. Regardless of the specific technology used, all these systems can be abstracted as a point-to-multipoint (i.e. operator-to-subscribers) device.

One of the main challenges of these systems is that they are so specific in nature (i.e. focus on the Access Network) and technology that it is hard to integrate their control and management planes in a more generic frameworks, such as an application oriented and multi-access technology solution. In this context, the SDN paradigm and OpenFlow are the tools that enable this integration by introducing a common abstraction for networking devices, the HAL. This layer deals with specific interfaces and hides the dependence with the technology. In the end, a HAL-based AN is agnostic with respect to the actual technology deployed. In this section we present an example of this proposal for a DOCSIS system, which exposes OpenFlow as its northbound interface. By doing this, the whole system can be abstracted as a single OpenFlow device.

The DOCSIS platform [DOCSIS 2014] comprises three main elements: the CMTS, the cable, the cable-modems (CMs). The CMTS is the head-end and 'intelligent' part of the system, which determines the use of the shared media by the CMs. The CMTS must be configured in the bridge mode (i.e. TLAN or L2VPN) to be compatible with OpenFlow abstractions. The cable is the shared media (coaxial) between the CMTS and several CMs.
Finally, the CMs are the tails of the system located at the subscriber's location. Collocated with the CMs, it is customary to deploy a managed residential gateway (RG) to implement some service related networking logic. In order to implement connectivity between any CMs in the bridge mode an external device (i.e. aggregation switch) is needed adjacent to the CMTS.

Since the DOCSIS platform is a closed device we cannot reprogram the devices and configuration is only possible through standard interfaces. In principle, this limits the integration of DOCSIS under an OpenFlow interface. However, by adding the RG and aggregation switch in the picture (i.e. as helper boxes), we can orchestrate the whole system to overcome these limitations and implement a full compatible solution. As a result, a DOCSIS proxy performs the proper abstraction from the whole system by sitting (in the control plane) between the set of network devices and the OpenFlow controller. This proxy is based on the AFA interface, since the actual data plane remains outside the DOCSIS proxy.

As previously mentioned, the DOCSIS proxy resides on an external box logically located between the platform and the controller and implements both layers of HAL: CHPL and HSL. The Cross-Hardware Platform Layer consists of the OpenFlow endpoint using the AFA as its southbound interface. Therefore, the CHPL interacts with the HSL thought the AFA. The HSL for DOCSIS platform implements the discovery, orchestration and translator functionalities. The discovery component provides information each time a new CM is connected to the system. As a consequence, the DOCSIS proxy dynamically updates the virtual ports exposed to the controller, since each CM is abstracted as a new virtual port of the virtual OpenFlow switch. The orchestration component allows to coordinate multiple hardware components (i.e. RG, CMs, CMTS and aggregation switch) to act as a single device. All the control plane interactions between the controller and the proxy must be handled to achieve a similar/emulated response from the set of devices. In order to improve the modularity of the system...
the orchestrator assumes three domains: RG, AN and Aggregation. Each domain implements its own driver to interact with the target device by the available interface. Moreover, by doing this separation the proxy can be easily developed and the AN technology can be changed just by developing a new driver for it. Finally, the translation component implements the logic to map the virtual ports (from a single virtual DataPath Identifier) to real ports and physical DataPath Identifiers, and vice versa. This functionality is implemented in coordination with the orchestration module and gets input from the discovery module.

The DOCSIS proxy can be used as an example of how any other AN technology can be abstracted following a similar approach and be exposed through an OpenFlow interface. By doing so, the AN can be controlled as any other OpenFlow resource, and even more, any previously developed OpenFlow application can run without any adaptation. As next steps we are investigating how the management of the shared media (i.e. Bandwidth assigned to each subscriber) can be exposed through the OpenFlow interface of the proxy. The appropriate extensions are currently under development.

3.1.3 Optical devices
Optical transport networks are the heart of carrier grade or metro networks and although most of SDN research and efforts are concentrated in packet domain, the fact that most use cases of SDN are within data centre and cloud computing environment forces the optical transport network to be compatible with SDN concept and specially OpenFlow protocol [Chanegowda 2013].

Generally, in an optical network device, control and data plane are separated and the notion of packet is not valid in circuit switched domains. In result, although the circuit switched device abstraction may look similar to packet switched device at the highest level, at lower levels it is different compared to packet switched devices. As an example, in an optical device the physical port abstraction has different characteristics and attributes. Moreover, the message types such as error messages or status messages are not the same for circuit switched devices.

The OpenFlow addendum v0.3 [EXT 2010] is an extension to make the OpenFlow protocol compatible for circuit switched devices. The CHPL data model and the OpenFlow endpoint were extended to meet all the requirements to support the circuit switched platforms in the HAL. The current specification of OpenFlow extension does not support optical features like switching constraints and optical impairments [Saradhi 2009]. We addressed these shortcomings in our HAL implementation for optical switched devices (see Figure 5).

![Figure 5 HAL adaptation for optical switch](image-url)
The hardware driver in HSP layer for optical devices uses an SNMP library to connect to network management interface on the control plane in order to communicate with the underlying data plane. The driver implements an SNMP trap and upon receiving an SNMP message it sends the message to the translator module. The translator module which is another part of HSP layer converts the SNMP messages to compatible OpenFlow messages and vice versa. The translator on its northbound interface exposes all the required features and information that can be used in an OpenFlow-based network. The translator is designed in such a way to present the optical device class based on OpenFlow addendum v0.3. On top of the HSP layer a resource model is formed which represents the abstracted underlying optical device in forms of cross-connections in a flow table. In nutshell the flow table for each flow consists of ingress port, egress port and a wavelength (this implementation is based on a DWDM ROADM). The OpenFlow endpoint on top of the resource model takes care of establishing connection to the OpenFlow messages.

3.2 Virtualization aspects in HAL

The Virtualization Agent (VA) is a HAL’s internal module which aims at providing a distributed slicing mechanism for the ALIEN devices. Like other virtualization approaches (FlowVisor [Sherwood 2010] and VeRTIGO [Doriguzzi Corin 2012]), the VA’s main objective is to allow multiple parallel experiments to be executed on the same physical substrate without interfering each other. The VA has been designed with the following goals: (i) avoid Single Point of Failures (SPoF) through a distributed slicing architecture, (ii) provide an OpenFlow version agnostic slicing mechanism and (iii) minimize the latency overhead caused by the slicing operations.

The HAL VA architecture is designed in a distributed fashion in order to avoid SPoFs. A failure of the Virtualization Gateway (see section 2.3.1.2), the only centralized element in the architecture, can only prevent the instantiation of new slices without affecting the ones already in operation. Other approaches like FlowVisor or VeRTIGO introduce an additional layer on the control channel to obtain the virtualization of the network resources where a failure of that layer would bring down all the running slices. The VA is protocol agnostic and does not inspect the control protocol (here OpenFlow) to perform the slicing process therefore it can, in principle, support any control protocol (even other than OpenFlow).

The resource virtualization operations have a cost in terms of additional latency on actions that cross between the control and the forwarding plane. The latency overhead depends on how the virtualization mechanism is implemented but other elements can contribute to the total latency too. In particular, contrary to FlowVisor and VeRTIGO, the HAL VA neither inspects the OpenFlow protocol nor needs to establish additional TLS connections.

3.2.1 The slicing mechanism

In OpenFlow protocol, for each incoming packet that does not have an entry in the switch flow table, a PacketIn event is generated and sent to the controller through the control channel. The controller, in turn, can answer with a FlowMod message to modify the flow tables of the switch. For each new packet, the fields of its header are matched against the flowspaces assigned to the configured slices. If the VA finds a match, the header is sent to the related OpenFlow endpoint which builds the packet-in message by using the protocol version used for the communication with the controller. Vice-versa, if no correspondence is found, the VA tells the lower layers to drop the packet.

On the other side, the VA applies the slicing policies to the OpenFlow messages sent by the controller to the switch. In order to keep the VA internal processes agnostic to protocol version, the VA intercepts the actions and the related flow match after they are de-capsulated from the OpenFlow message and before they are inserted into the switch’s flow table. The actions are checked against the controller’s flowspace (i.e. the VA checks if the controller is trying to control traffic outside its flowspace) and the match is intersected with the flowspace. The latter operation ensures that the actions are only applied to the flows matching the flowspace assigned to the controller, i.e. the VA prevents interference between different slices.

3.2.2 Virtualization management

The VA slices the overall flowspace among many OF Controllers based on the configuration received from a NMS. The NMS communicates to the VA through the Virtualization Gateway (VG) sub-module (see Figure 6).
While the VG communicates with the VA instances through a JSON-RPC interface, on the northbound the VG accepts GENI v3 rspecs with OpenFlow extensions [GENI 2014]. In particular, the VG northbound interface allows the communication with the FOAM aggregate manager [FOAM 2014] and, therefore, a seamless integration of the HAL-enabled devices within testbeds already in operation like OFELIA [Suñé 2013].

4. Hardware Abstraction Layer: Implementation

Based on the above-mentioned dual-layered concept, HAL is actively implemented and tested in the FP7 ALIEN project on foundation of ROFL [ROFL 2014] and xDPd [XDPD 2014] open-source projects. ROFL is a set of libraries providing support for various versions of OpenFlow protocol and OpenFlow pipeline (OF1.0, OF1.2 and OF1.3). xDPd is a software framework for HAL instantiation for various platforms and it uses ROFL library extensively. Both project are written in C/C++ and built focusing on performance and extensibility.

The Abstract Forwarding API (AFA) and Hardware Pipeline API (HPA) interfaces [ROFL 2014][XDPD 2014], which integrate the hardware-agnostic and hardware-specific sub-layers of HAL, are defined as part of ROFL library. Both interfaces binds the platform-specific forwarding module, which must be created as a subproject inside the xDPd project, with the control and management modules of xDPd representing CHPL part of the HAL architecture [D2.2].

The xDPd implements extensibility in form of plugin mechanism. This plugin mechanism allows for adding a new functionality to the HAL instance, e.g.: adding new datapath northbound protocols for the configuration and management of the network node, adding OpenFlow extensions, network node configuration utilities, etc. This plugin mechanism of xDPd framework was used to add distributed flowspace slicing mechanism to the HAL in the form of Virtualization Agent (VA). VA plugin interacts with OpenFlow endpoint before creating the OpenFlow messages directed to the controller and after the payload is decapsulated from the OpenFlow messages coming from the controller.

In the ALIEN project, the HAL implementation is validated on several network equipment platforms [D2.3 2014] such as EZappliance with EZchip NP3 network processor, GEPON, NetFPGA, DOCSIS, ATCA with Cavium Octeon network processor, ADVA Layer 0 Switch, Dell/Force10 7024 switch with ASIC and Cavium Octeon. At the time of this writing, the xDPd forwarding module has been already implemented and tested on several platforms, including OCTEON, Broadcom and GNU/Linux. All these implementations are publicly available [ROFL 2014][XDPD 2014].

This xDPd/ROFL are subsequently mapped to the HAL architecture, as illustrated on Figure 7.
5. HAL Design Considerations and Advantages

The OpenFlow protocol itself is an abstraction layer for network devices, but its implementation is still a quite complex task. This makes SDN implementation based on OpenFlow a really coarse-grained system, far behind the level of granularity observed in computer operating systems. The pace of development in these two areas is also still incomparable despite the tremendous advances which have already been made after the first introduction of OpenFlow protocol.

The main purpose of HAL is to make any legacy network device OpenFlow-compatible through a set of abstractions. This should allow operators on the one hand to extend their OpenFlow-based control plane to legacy valuable infrastructure, and on the other hand to network modern OpenFlow switches with non-OpenFlow capable devices in a seamless manner. This means that similar to computer operating systems HAL has to have the ability to interact with various hardware devices with different architecture. Comparing the HAL design for computer and network devices, one will notice that for network devices in some cases the nature of modular hardware platform makes them reconfigurable and also the data plane in some of network devices are distributed, i.e. the hardware is not placed in one fixed box. These are the fundamental differences between computer and network device hardware architecture which makes the HAL for network devices to have a more complex and different design than the computer HAL. Moreover, in the ALIEN project, the proposed HAL for network devices should communicate with platforms which do not have standard interface for abstraction i.e. closed-box platform.

Considering the large array of devices (e.g., x86-based packet processing devices, lightpath devices, point to multipoint devices, programmable network processors etc.) that can be supported by HAL, the architecture has been based on a modular design approach which is easily extensible and compatible with heterogeneous network devices. Moreover with modular design approach the behaviour of any platform can be modified and extended without compromising the overall HAL architecture. It also makes HAL’s implementation easier and faster for similar network platforms by exploiting the module reusability.

A key design choice for HAL is to decouple the hardware-specific control and management logic from the network node abstraction. This decoupling allows HAL to hide the device complexity as well as the technology- and vendor-specific features from the control plane logic. This solution is particularly addressed to network equipment vendors who can make an extensive code reuse in various use cases due to aforementioned
decoupling. Moreover, the decoupling enables OpenFlow protocol implementation for a new device (or a new version of an existing device) just through the adaptation of the HSP code.

The proposed architecture design allows HAL to be employed by OpenFlow-capable devices, i.e. devices which support a particular OpenFlow version out-of-the-box, in order to extend their functionality. For example, HAL can be employed by an OpenFlow version 1.0 switch to allow support for OpenFlow version 1.2 features. Similar approach can also be used for a rapid implementation of new versions of the OpenFlow protocol. Such operation will require only a new HAP that will be reused without modifications on all devices produced by a given vendor.

Another design aspect of CHPL (and therefore of HAL) is the extensibility through independent plug-in modules dealing with device or system management, monitoring. For example, on the management side, CHPL presents a unified abstraction of the physical platform (fundamentally physical ports, virtual ports, tunnels, etc.) to plug-in manager. The plug-in modules hosted by plug-in manager are capable of steering the configuration of the OpenFlow endpoints, for instance. Examples of plug-in modules (as shown on Figure 8) are NetConf/OFConfig agent or a file-based configuration reader on management section and Virtualization Agent (VA) on virtualization section. The plug-in manager is fully capable of supporting other plug-in modules like AAA etc.

![Figure 8 Plug-in Manager in Hardware Agnostic Part](image)

6. Summary and Future Work

The main objective of the FP7 ALIEN project is to define the Hardware Abstraction Layer (HAL) enabling non-OpenFlow capable hardware to be controlled through the OpenFlow protocol in a consistent manner. This abstraction layer is the key enabler towards enhanced network programmability on a diverse set of devices that do not adhere to the Ethernet model upon which the original OpenFlow work is based. The HAL architecture presented in this paper, as well as its implementation, may be reused by hardware vendors and developers when implementing OpenFlow on these hardware platforms that do not support it natively.

The Hardware Abstraction Layer exposes OpenFlow protocol to the Network Controller. The rationale behind network devices abstraction are twofold: facilitate the resolution of problem of controlling the forwarding behaviour inside different network devices, increase its generality and propose common interface which hides underlying complexity of its implementation. Following this idea OpenFlow as a practical realization of such abstraction defines abstract data plane structures. The flow abstraction allows unifying the behaviour of different types of network devices. Thanks to the HAL all the underlying network hardware is addressable through a common abstraction protocol (OpenFlow). HAL hides the complexity of network devices behind common interface and uses available methods, libraries, and drivers to achieve desired functionality. Of course this results in some restrictions. Abstraction as a way of generalization causes some simplifications. In OpenFlow protocol we are restricted to packet headers of common network protocols defined by the protocol itself with limitations of supporting the user-defined or new protocols. We are restricted also to capabilities of network devices e.g. closed-box platforms don't offer access to program the datapath so they usually can't offer full OpenFlow implementation. From the other side OpenFlow abstraction get rid of capabilities of fine-grained processing systems like programmable switches with access to every bits of processed frames. Instead of this HAL offers common abstraction view defined by OpenFlow protocol to the Controller and network applications.
By September 2014, the work described in this paper will be integrated with the OFELIA Control Framework [Suñé 2013] and tested with the CCN application developed by CONET [Detti 2011].

Acknowledgment

This work was conducted within the framework of the FP7 ALIEN project, which is partially funded by the Commission of the European Union. Study sponsors had no role in writing this report. The views expressed do not necessarily represent the views of the authors’ employers, the respective projects, or the Commission of the European Union.

Moreover, ALIEN uses the PL-LAB infrastructure of Future Internet Engineering [FIE 2014], the Polish national research project to implement and deploy Hardware Specific Parts of EZappliances and NetFPGA.

References


[Accessed 15th April 2014]

Vitae

Łukasz Ogrodowczyk received the M.Sc. in the Electronics and Telecommunications, majoring in the Networks
of Information Transport, from the Poznan University of Technology in 2007. From 2010 he works in Poznan
Supercomputing and Networking Center as a Computer Systems Analyst. Co-author of some papers for
international conferences. He was involved in the Polish project Future Internet Engineering. Currently he works
in the international project FP7 ALIEN in the area of OpenFlow and SDN. Łukasz is familiar with embedded
systems, programming languages like C/C++ and Python, network protocols and Quality of Service in next
generation packet networks.

Bartosz Belter received the M.Sc. degree in Computer Science from the Poznan University of Technology in
2002. He works in Poznan Supercomputing and Networking Center as the leader of the Network R&D unit. He
participated in several FP6/FP7 IST projects: GEANT, GEYSERS, NOVI, BonFIRE and PHOSPHORUS .
Currently he is involved in several FP7 projects, acting as the Technical Coordinator of ALIEN and FELIX. He
was a member of the TERENA Networking Conference Programme Committee. His main research activities
concern the architectural aspects network function virtualization, network control and management planes and
Quality of Service in next generation packet networks.

Artur Binczewski received the M.Sc. degree in Computer Science from the Poznan University of Technology in
1993. He is the Manager of Network Division at the PSNC. He was involved in several EC projects: SEQUIN
(IST-1999-20841), ATRIUM (IST-1999-20675), 6NET (IST-2001-32063). He coordinated the Porta Optica
Study (RJ026617), PHOSPHORUS (IST034115), ADDONAS (PIANO+) and ALIEN (317880) projects. He is
author or co-author of papers in major professional journals and conference proceedings.

Krzysztof Dombek received the M.Sc. degree in 2008 from the University of Technology and Life Sciences in
Bydgoszcz, Poland. He's a Computer System Analyst in Polish NREN – Poznan Supercomputing and
Networking Center. He was a participant of many European research projects e.g. PHOSPHORUS, GEANT3 or
GEYSERS. He was also involved in national project of the Future Internet Engineering. At present he's a
participant of ALIEN project founded by EU in FP7 where he's involved in integration with the OFELIA Control
Framework.

Artur Juszczyk received the M.Sc. degree in the Electrical Engineering, majoring in the Microprocessors and
Control Systems, at the Poznan University of Technology in 2005. He works in the Poznan Supercomputing and
Networking Center since 2007 as a computer systems analyst. He took part in the several European and national
projects, PHOSPHORUS, GEYSERS, GEANT and Future Internet Engineering as a programmer, tester, system
engineer. He is currently involved in the international project ALIEN. His interests focus on the Unix base
systems, virtualization of resources, management of network infrastructure and next-generation networks.

Iwo Olszewski received the M.Sc. degree in Computer Science, majoring in the Computer Networks and
Distributed Systems, from the Poznan University of Technology in 2010. He started working in PSNC NOC in
2006. In 2010 he has joined the NGN Team as an analyst and software engineer. He has experience in
programming in Java and C and deep knowledge about network protocols and devices. He is familiar with
EZchip network processors and Juniper Junos platform development. He was involved in the national project
Future Internet Engineering and FP7 project NOVI. Currently he devotes his time to the ALIEN project.

Damian Parniewicz received the M.Sc. degree in Telecommunication from the Poznań University of
Technology (2005) and currently is a network systems analyst in Poznan Supercomputing and Networking
Center. He has participated in several European Union Research and Development projects related to network
management, optical network equipment and control planes (FP6 and FP7 IST projects: MUPBET,
PHOSPHORUS, GEANT3, GEYSERS, ALIEN and Polish national project: Future Internet Engineering). His
major interest areas are control planes protocols for networks and VoIP systems, traffic engineering and software
technologies.

Roberto Doriguzzi Corin received his degree in Mathematics in 1996 from the University of Trento, Italy, with
a thesis in calculus of variations. From 1999 to 2008 he was employed in Neuricam S.p.A. in Trento as Linux
Matteo Gerola received his Master degree on Telecommunications Engineering in 2007 from the University of Trento. From October 2005 to July 2006 he has been working on WiMAX experimentation in CSP, an innovation and research centre based in Turin. He joined CREATE-NET in July 2007 where he is currently working as Research Engineer. He is now on an internship at ON.Lab, the no-profit research lab founded by SDN inventors and leaders from Stanford University and UC Berkeley whose aim is to foster an open source community to develop SDN tools and platforms. His main research interests are network virtualization and software-defined networking.

Dr. Elio Salvadori is the Research Director at CREATE-NET. After his M. Sc. degree at Politecnico di Milano, he worked as systems engineer until 2001 and then moved to the University of Trento to pursue his Ph.D. degree on optical networking. Within CREATE-NET he has been leading an Engineering & Fast Prototyping Area from 2008 on. His scientific background is on Control Planes for transport networks and software-defined networks developed in both industrial- and EC-funded research projects. Moreover, he has been involved in several EU projects (FIRE, FI-PPP and EIT ICTLabs) related to Future Internet experimental testbeds.

Kostas Pentikousis is the Head of IT Infrastructure at EICT GmbH, Berlin, Germany. He earned his Bachelor’s degree in informatics from Aristotle University of Thessaloniki, Greece, and his Master’s and doctoral degrees in computer science from Stony Brook University, USA.

Umar Toseef received his Master of Science in Communication and Information Technology degree from University of Bremen, Bremen, Germany. After completing his master studies he was appointed as a research assistant at the University of Bremen in the Communication Networks (ComNets) group. He has contributed to several research projects like ScaleNet, SAIL etc. His areas of interest include LTE (Long Term Evolution) system level studies, network dimensioning, heterogeneous wireless networks, and SDN. He received his PhD degree with Honors in 2013 from ComNets, University of Bremen. His contributions to research literature include several scholarly articles and patents.

Dr.-Ing. Hagen Woesner studied computer science in Chemnitz and Berlin and received the diploma and doctoral degree from TUB. He coordinated the FP7 OFELIA project and is an active contributor to FP7 ALIEN.

Mehdi Rashidi Fard received his M.Sc in Computer and Information Networks in 2011 at University of Essex. After graduation, he started his job as a research officer in High Performance Networks group in the Essex University and University of Bristol. He is a participant of ALIEN project where his role is to integrate circuit (optical) switching functionality into the OpenFlow protocol within the HAL concept. His interests focus on network virtualization, cloud computing, cloud orchestration and software defined networking.

Dr. Maider Huarte is a lecturer at the University of the Basque Country since 2002, where she started teaching in computer science subjects, like programming languages and computer communications. Currently she teaches advanced Java EE programming and networking. She has been researching on cryptographic security, distributed systems and software fault tolerance. She obtained her Ph.D. in 2009 on i-voting systems, with a thesis that involved designing and validating cryptographic communication protocols. Nowadays she is also researching in Software Defined Networks and OpenFlow.

Dr. Eduardo Jacob spent a few years in the industry as network manager first and R&D project leader later. He came back to the University of the Basque Country where he received his Ph.D degree in ICT in 2001. He is the Head of the Department of Communications Engineering. He also leads a research group where he has directed several R&D projects at European level. His research interests are tied to Advanced Networks (nowadays SDN/OpenFlow and NFV), connectivity for ITS and security in distributed systems with recent results in authentication and virtualized access networks and authorization in sensor networks.

Jon Matias received his BS and MS degree in Electrical Engineering from the University of the Basque Country (UPV/EHU, Spain) in 2003. He currently works as a lecturer and researcher in the Electronics and Telecommunications Department at the Faculty of Engineering of Bilbao (UPV/EHU). He is also pursuing the PhD degree in electrical engineering at the same University focused on access networks and security. His research interests include Computer Networks, Broadband Access Networks, Software Defined Networking, Services Provisioning and Security.

Victor Fuentes received his BSc and MSc degrees in Telecommunication engineering in 2013 from the University of the Basque Country (UPV/EHU). His research interests include Software Defined Networking, network virtualization and integration of non-Openflow access networks -GPON and DOCSIS- into Openflow controlled domains.
Dr. Marek Michalski (IEEE, IEICE member) received MsC and PhD degree in Electronics and Telecommunications from Poznan University of Technology in 2002 and 2011 respectively. His scientific interests and PhD thesis concern optical and electronic switching systems, especially control algorithms for them. He is also interested in: IP/Ethernet networks (he works as CCNA/CCNP instructor), software (C, C++, C# and other) and hardware (Verilog, VHDL) programming (FPGA), recently he is focused on NetFPGA cards and Software Defined Networks mainly working with OpenFlow protocol. He was involved in ICT projects as BONE, Future Internet Engineering, ALIEN and several COSTs.

Remigiusz Rajewski received MsC degree in Electronic and Telecommunications from Poznan University of Technology in 2006. His scientific interests concern optical and electrical switching systems, multirate connections, switching/control algorithms, and switching architectures. He is also interested in software (C, C++, C#) and hardware (Verilog) programming. Recently, he is working with Software Defined Networking. He has participated in few projects: COST-291, BONE, Future Internet Engineering (Polish national project). Currently he is involved in the ALIEN project.