ALIEN Project - Abstraction Layer for Implementation of Extensions in programmable Networks

Grzegorz Danilewicz, Marcin Dziuba, Janusz Kleban, Marek Michalski, Remigiusz Rajewski, Mariusz Żal
Poznan University of Technology
Chair of Communication and Computer Networks
Polanka 3, 60-965 Poznan, Poland
e-mail: {grzegorz.danilewicz, marcin.dziuba, janusz.kleban, marek.michalski, remigiusz.rajewski, mariusz.zal}@put.poznan.pl

Bartosz Belter, Artur Binczewski, Krzysztof Dombek, Artur Juszczyk, Łukasz Ogrodowczyk, Iwo Olszewski, Damian Parniewicz, Maciej Stroiński
Poznan Supercomputing and Networking Center
Network Technologies Department
Poznan, Poland
e-mail: {bartosz.belter, artur, kdombek, juszczyk, lukaszog, iwo, damianp, stroins}@man.poznan.pl

***Abstract*—This paper presents motivation and goals of the ALIEN - EU FP7 project. The main goal of the project is to provide an experimentally verified OpenFlow Hardware Abstraction Layer (HAL) for controlling the forwarding behavior of non-OpenFlow capable hardware. The general idea of HAL and its implementation in non-OpenFlow capable hardware are presented. The performance of HAL implementation will be tested using OFELIA experimental infrastructure, especially, the OFELIA Control Framework. Proposed solution will extend OpenFlow functionality on different kind of programmable devices, which will be used in SDN network.**

I. Introduction

ALIEN (Abstraction Layer for Implementation of Extensions in programmable Networks) project has started at 1st October 2012 and will be finished at 30 September 2014. It is the FP7 STREP (Specific Targeted Research Project), multipartner research project. The consortium consists of: Poznan Supercomputing and Networking Center (PSNC) - Poland; Center for Research and Telecommunication Experimentation for Networked Communities (CREATE-NET) - Italy; European Center for Information and Communication Technologies GmbH (EICT) - Germany; DELL France SA (FORCE) - France; Poznan University of Technology (PUT) - Poland, University College London (UCL) - UK; University of Bristol (UNIVBRIS) - UK, and University of the Basque Country (UPV/EHU) - Spain.

The main objective of the ALIEN project is related to Software Defined Networking (SDN) [1] and the OpenFlow protocol. The SDN is an idea of network which allows network administrators or operators to flexibly manage network devices like routers or switches using software running on special dedicated to this purpose external servers. The main idea of this solution is to separate the control plane/layer from the packet forwarding plane/layer [2], [3]. The control plane is realized as a "network operating system" (more simply "network OS") which manages the entire current network state just from one central point in such a network. OpenFlow is an open standard which allows centralized way of programming flow tables in heterogeneous switches and routers [4], [5]. Such tables include flow entries with actions to determine instructions for routing and packet processing e.g. actions decide how packets should be modified and which packet should be send through which port. This idea allows to manage flow tables (adding, removing or editing table entries to switches or routers) via a central controller. Strong point of this solution is that researchers can experiment with a new kind of flows not interfering with existing ones. It makes also easiest to test a new kind of routing and switching protocols. Of course, OpenFlow is not the only one method supporting the SDN concept. Cisco introduced its own SDN idea called Cisco ONE [6], [7]. This solution, unlike OpenFlow, allows to program layers in the network both above and below the data and control plane/layer [1]. Another solution for SDN – as an alternative to OpenFlow and Cisco ONE – is ForCES introduced by IETF [4], [8], [9], [10], [11], [12], [13], [14], and [15].

The ALIEN project aims at delivering innovative network abstraction layer to connect non-OpenFlow capable equipment (called also by us as "alien hardware") to OpenFlow environment. It can be achieved by providing the Hardware Abstraction Layer (HAL) for non-OpenFlow capable network devices. Significant effort has been allocated to the specification of the HAL architecture that will be common for different hardware platforms considered in the project. Implementation of the HAL elements in these non-OpenFlow capable hardware platforms will allow to connect them to the OFELIA European OpenFlow test-bed and manage as standard OpenFlow switch. "Alien hardware" is any type of network device that does not support natively an OpenFlow. It could be:

- packet switching equipment: traditional L2 packet switches without OpenFlow support;
- non-packet switching equipment: optical switches, EPON devices, etc.;
- packet processing and monitoring equipment: FPGA cards, network processors;
- CATV equipment: HFC modems etc.

The following equipment is considered in the ALIEN project: NetFPGA cards, EZChip NP-3 network processors (in EZappliance platform), Cavium OCTEON Plus AMC network processor (module in an ATCA systems and DELL switch), optical switches, GEPON OLT and ONU units, and DOCSIS hardware. All this hardware is not yet OpenFlow capable.

The main project objectives are as follows:

- To define HAL enabling non-OpenFlow capable hardware to be controlled by the OpenFlow protocol. This layer must facilitate unified integration of alien types of network hardware elements with usage of hardware description language and common interfaces that can facilitate uniform representation of any type of alien hardware and their capabilities.
- To implement the proposed HAL layer using a selected list of network equipment. The modular software design should be applied for the HAL implementation in order to allow the separation of the evolution of the OpenFlow management endpoint (a common part for all hardware platforms) and hardware specific drivers which communicate with the network equipment.
- To integrate HAL controlled devices into the OFELIA facility. The integration approach should take into account the requirements coming from current OFELIA CF architecture as well as capabilities of alien hardware.
- To test proposed solutions in experiments performed within the OFELIA test-bed. The experiments should allow to validate and demonstrate the usage of alien hardware controlled via the HAL. Each experiment should be performed with a network slice managed within the OFELIA Control Framework. The experiments will focus on the Content-Centric Networking (CCN) usage scenarios.
- To disseminate and publish project results across related research and standardization forums. High visibility of the project should be ensured as well as the cooperation and integration with other international projects related to Software-Defined Networking and Future Internet architectures should be achieved.

The paper is organized as follows: Section II presents main ideas of OpenFlow standard, Section III describes HAL architecture, and Section IV discusses basics of HAL implementation. Section V is devoted to presentation of OFELIA Control Framework. The paper is concluded in Section VI.

II. OpenFlow Standard

OpenFlow is an open standard supporting communications interface defined between the control and forwarding planes of the SDN architecture. The OpenFlow ecosystem consists of routers, switches, virtual switches, and access points. The main idea of the OpenFlow is to give access to and facilitate manipulation of the forwarding plane of network devices. It provides an open interface to control how data packets are forwarded through the network, and a set of management abstractions used to control topology changes and packet filtering. The behavior of network devices may be modified through a well-defined "forwarding instruction set". In this case network control may be moved out of the networking nodes to logically centralized control software. The OpenFlow protocol specifies a set of instructions that can be used by an external application to program the forwarding plane of network devices.

Currently, OpenFlow is implemented by major vendors in commercial switches, routers and wireless access points to allow researchers to run experimental routing protocols for example in campus networks without needing to reconfigure the internal workings of network devices. Now, users can control how traffic flows through a network defining flows and determining what path those flows take through a network, regardless of the underlying hardware. The OpenFlow standard may be used for applications such as virtual machine mobility, high-security networks and next generation IP based mobile networks [16].

OpenFlow consists of three parts (see Fig. 1) [4]:

- Flow Tables installed on switches. The switch is informed how to process the flow by an action associated with each flow entry.
- A Controller, which uses the OpenFlow protocol to communicate with switches to impose policies on flows. The OpenFlow protocol provides an open and standard way for the controller to communicate with switches and allows entries in the flow table to be defined externally. Flows are transmitted through the network on paths predefined using the controller and enforced by switches.
- A Secure Channel that connects the remote controller (remote control process) to switches and allows secure communication between them. The SSL protocol may be used to securely send commands and packets from the controller to switches using the OpenFlow protocol.

The OpenFlow architecture separates data path and control path functions. The data path functions are still implemented on the switch, while routing decisions are moved to the controller, which is a standard server. This solution is different from a classical router or switch, where a data and control planes occur in the same device. Now, the controller

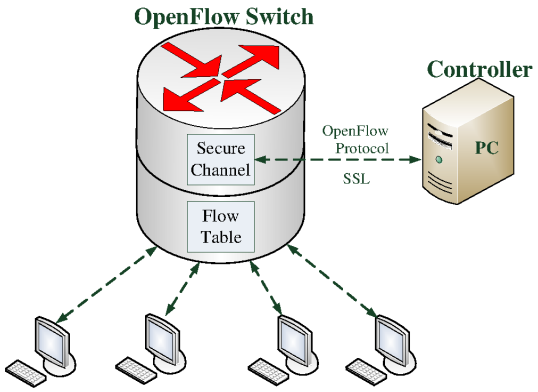

Fig. 1: Main components of the OpenFlow switch

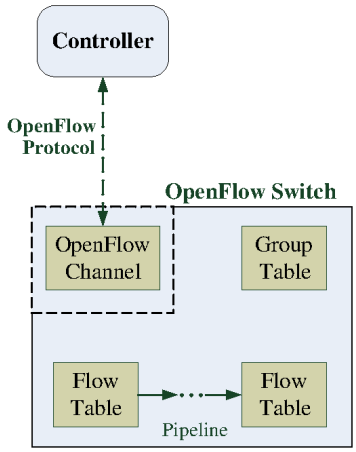

Fig. 2: The architecture of an OpenFlow switch

communicates via the OpenFlow protocol with switches using such messages as packet-received, send-packet-out, modify-forwarding-table, and get-stats. The data plane is based on the flow table, where each entry contains a set of packet fields to match, and an action e.g. send-out-port, modify-field, drop. If there is no matching flow entry for a received packet the OpenFlow switch sends this packet to the controller, which, in turn, decides how to handle the packet. The packet may be dropped or a flow entry may be added to the flow table to inform the switch how to forward similar packets in the future.

The main components of an OpenFlow switch are shown in Fig. 2 [17]. It consists of one or more flow tables and an OpenFlow channel to an external controller. The flow tables are used to manage flows and perform packet lookups, and forwarding. The OpenFlow channel is used by the controller to manage the switch via the OpenFlow protocol. Using this communication channel the controller can add, update, and delete flow entries in flow tables. Each flow table contains a set of flow entries with match fields, counters, and a set of instructions to apply to matching packets. Matching starts at the first flow table and continues to additional flow tables until a matching entry is found. In the case of matching, the instructions associated with the matching entry are executed. If no match is found in the flow table the packet may be forwarded to the controller over the OpenFlow channel, may be dropped or may continue to the next flow table. Actions performed on packets and pipeline processing are defined by a set of instructions associated with each flow entry. Packet forwarding, packet modification and group table processing are described by actions. Pipeline processing instructions define how subsequent tables process packets and what kind of information, in the form of metadata, is sent between tables. If a next table is not specified by the instruction set associated with a matching flow entry the table pipeline processing stops and the actions in the pre-defined action set of the packet are executed.

Packets may be forwarded to a physical or logical port. The logical port may be defined by the switch or by the OpenFlow switch specification. Ports defined by the OpenFlow switch specification are called reserved ports. These ports may specify generic forwarding actions such as: sending to the controller, flooding, or forwarding using non-OpenFlow methods. The switch-defined logical ports may specify link aggregation groups, tunnels or loopback interfaces.

Additional processing is specified by a group table. The group table consists of group entries and represents sets of actions for flooding, as well as more complex forwarding semantics (e.g. multipath, fast reroute, and link aggregation). Each group entry contains a list of actions buckets with specific semantics dependent on group type. The actions in one or more action buckets are applied to packets sent to the group. The detailed information about each component of the OpenFlow switch as well as the OpenFlow protocol may be found in [17].

III. Hardware Abstraction Layer Architecture

Hardware Abstraction Layer provides an abstracted version of hardware, for different types of network devices, to make the device compatible with OpenFlow protocol. To do that, it decouples hardware-specific control and management logic from the network-node abstraction logic (i.e. OpenFlow). Decoupling in the HAL hides the device complexity as well as technology and vendor specific features from the Control Plane logic. The decoupling is done by splitting the HAL (see Fig. 3) into two layers: 1) Cross-Hardware Platform Layer which is in charge of node abstraction, virtualization and communication mechanisms and 2) Hardware Specific Layer which is in fact a collection of the hardware specific software modules, collectively called driver, responsible for discovering hardware platform resources and configuring network devices. 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 gradual and modular abstraction in the HAL architecture gives the possibility of changing and extending any platform without compromising the whole HAL architecture. It also makes HAL's implementations easier for similar network platforms by module reusability in common components (i.e. OpenFlow pipeline implementation).

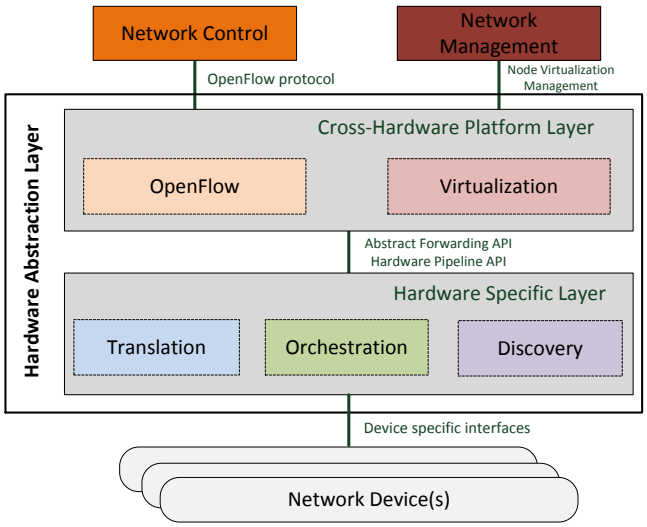

Fig. 3: The HAL architecture

The Cross-Hardware Platform Layer is shared layer between all different platforms and composed of independent modules dealing with device or system management, monitoring and control plane (OpenFlow). On the management side, this layer presents a unified abstraction of the physical platform (fundamentally physical ports, virtual ports, tunnels, etc.) to plugin modules. The plugin modules can steer the configuration of the OpenFlow endpoints, for instance, defining the OpenFlow controller. Examples of plugin modules are NetConf/OFConfig agent or a file-based configuration reader on management section and Virtualization Agent (VA) on virtualization section.

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] and [VeRTIGO]), 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 idea behind the Hardware Specific Layer is to deal with diversity of the network platforms and their communication protocols to overcome the complexity of implementing OpenFlow protocol on different hardware. In the real world, every network equipment or platform has its own protocol or API for communicating, controlling and managing the underlying system. In the proposed HAL, the Hardware Specific Layer is responsible to hide the complexity and heterogeneity of underlying hardware control for message handling and provide a unified and feature rich interface in its northbound for the upper layer i.e. Cross-Hardware Platform Layer. Although the Hardware Specific Layer on its northbound has a unified interface, on the southbound, it is in direct contact with the underlying hardware, which makes it dependent to the hardware in terms of communicating protocol and programming language. This results the layer to have different implementation method for each platform. Following the modularity principle and also in order to make the HAL flexible enough to support different hardware platform, different modules in Hardware Specific Layer take care of supporting hardware platforms heterogeneity. The layer has been designed in a way that the changes inside its modules do not affect the upper layer (hardware independent) functionality and in most cases there is no need to manipulate the architecture. In the following the modules inside the Hardware Specific Layer and their functionalities are explained.

DISCOVERY MODULE - In order to initialize Cross-Hardware Platform Layer, a set of information about network device(s) must be provided from Hardware Specific Layer. The information needed are: (i) a list of devices working together as a single hardware platform instance and controlled by a single OpenFlow agent instance. For each device, the access information is also required. (ii) a list of all network ports and their characteristics (e.g. transmission technology, transmission speed, operational status, etc.) from every device. (iii) the internal hardware platform topology (e.g. how all devices within a hardware platform instance are interconnected) must be recognized. This is required for orchestration functionality to work properly in Hardware Specific Part (HSP) in HAL. There are various design options to implement discovery functionality. The discovery can be manual (e.g. platform administrator creates static configuration files containing some part of required information and HSP loads that configuration file during initialization) or automatic (e.q. Hardware Specific Layer queries each device for all information and reacts to new notifications coming from the device) or combination of both approaches. Depending on the implementation and also the platform, the discovery process could be active just only during Hardware Specific Layer initialization or executed continuously (e.g. periodical queries in order to discover changes in the hardware).

ORCHESTRATION MODULE - In some cases, the hardware platform is composed of multiple hardware components acting independently but controlled centrally (e.g. DOCSIS, GEPON). The orchestration procedure 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 identify if coming request from Cross-Hardware Platform layer was successfully applied to all hardware components. Also, the orchestration process should be able to recover from configuration failures on a single hardware component and restore initial state of all the hardware components. The orchestration process is initialized by a request (e.g. Add-flow method of AFA interface) from Cross-hardware Platform interface.

TRANSLATION MODULE - The Translator module 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 and vice versa. Translator acts as a middleware between OpenFlow switch model and underlying physical device. Due to heterogeneity of the network devices, translation specification and implementation is different for each network device. Generally the module is responsible for translating all port numbering, flow entries and packet related actions from OpenFlow switch model into platform specific interface commands and processor instructions or configuration modifications D3.2. In most cases of hardware platforms, the translation functionality will be stateful and requires storing of information about all handled OpenFlow entries and its translation to specific device commands. It allows to modify or delete a device's applied re-configuration which strictly refers to a given flow entry.

IV. Hardware Abstraction Layer Implementation

The main components of HAL implementation for ALIEN harware platforms are shown in Fig. 4. In the ALIEN project, xDPd/ROFL (eXtensible DataPath Daemon/Revised OpenFlow Library) libraries [18], [19], [20] are used as a reference HAL concept implementation and the framework for creating OpenFlow agents communicating with different types of hardware platforms.

The ROFL library is a set of modules to build both OpenFlow data paths and controllers. On one hand, the controller ("rofl-common") part of the library includes a full C/C++ OpenFlow v1.0, v1.2 and v1.3.2 endpoint to be embedded

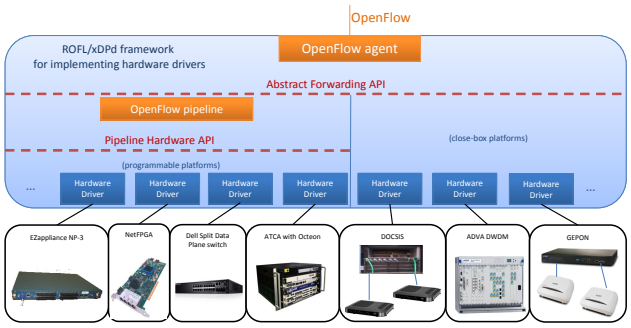

Fig. 4: The implementation of HAL for ALIEN hardware platforms using xDPd/ROFL framework

both in data path elements as well as in controllers, including hierarchical (like FlowVisor) or recursive controllers. On the other hand, the ROFL library includes two main libraries for building data path elements, the Abstract Forwarding API (”rofl-afa”) and the Pipeline (”rofl-pipeline”) sub-libraries. The Abstract Forwarding API (AFA) is a set of header files which define a hardware agnostic API for heterogeneous forwarding engine management. ROFL-pipeline in its turn, is a complete OpenFlow engine and forwarding data model, for OpenFlow versions v1.0, v1.2 and v1.3.2. The ”rofl-afa” library currently uses ”rofl-pipeline” as its data model.

The xDPd is a multi-platform OpenFlow data path element, build using ROFL libraries. The architecture of xDPd is made of basically three components:

- The Control and Management Module in charge of the common control part of the logical switches (OpenFlow endpoint) and the management;
- The AFA interface of HAL, which abstracts the details of the underlying platform via the hardware agnostic C interface;
- The forwarding modules (also known as drivers) for the specific platform, which fulfills the AFA interface.

A. AFA Interface

Abstract Forwarding API (AFA) is binding the platform specific, forwarding module with control and management module, which is keeping the abstracted switching information. AFA interface provide below features:

- Management actions: allow for creating and destructing logical switches within the forwarding module (a ”driver”). It binds the logical switch instance and ports or network interfaces.
- Configuration actions: passing OpenFlow requests to the forwarding module (e.g.: flow tables entry addition, flow statistics query, etc.).
- Notification of events : Passing events generated by the forwarding module to CMM module (e.g.: port status changes, OpenFlow packet-in and other OpenFlow messages and errors).

B. Hardware Specific Parts

A northbound bridge of HPL is defined by AFA interface, its behavior and functionality. It connects HPL and CMM. It defines logically and functionally rules on this interface. They are platform independent. Their assumptions come from OpenFlow specification. They will be realized by ”alien hardware”. As it was mentioned earlier, there are considered different platforms. They have some similarities, but deep details of their nature are fundamentally different. Hence, the code realizing AFA interface for each of them will be very specific for each platform. This part of software we called a Hardware Specific Parts (HSP). Each partner responsible for particular platform will prepare software and hardware modules for the platform which is responsible for. It can be said that HSP is a southbound bridge of HPL, it is a part of HAL, it implements functionality of platform specific driver.

V. OFELIA Control Framework

As part of SDN activities the ALIEN project will extend the OFELIA Control Framework. OFELIA is a FP7 project aiming towards creation of an OpenFlow test facility allowing researchers from academia as well as industry to develop new control protocols in controlled networking environments on dedicated OpenFlow enabled carrier-ready hardware devices. ALIEN will extend OFELIA Control Framework and its architecture to support abstraction of network information of equipment that is alien to the OpenFlow technology.

The current OFELIA experimental facility is oriented to enable the experimentation with OpenFlow at European level. Therefore, the OpenFlow infrastructure is exposed to the researchers in order to test novel networking approaches. Basically, OFELIA is a shared facility with two types of resources: Virtual Machines (VMs) based on Xen and OpenFlow enabled switches. The researchers apply for both types of resources and obtain computational capacity at VMs and a control interface to the networking elements. In a nutshell, OFELIA allows researchers to not only experiment on a test network but to control the network itself.

OpenFlow (OF) has evolved quickly in the last years with several different versions (v1.0, optical extensions v0.3, v1.1, v1.2, v1.3, and the last version v1.3.1) in a short period of time, which in fact are incompatible between them. This situation causes some chaos and has been very demanding for vendors, developers and researchers. Currently, the most widely deployed and implemented version of OpenFlow (both by vendors and open source community) is OFv1.0. This is the reason why OFELIA has deployed OpenFlow version 1.0 (OFELIA started on October 2011).

As an experimental facility, OFELIA needs to share its resources between several experiments and researchers. The OFELIA Control Framework (OCF) is in charge of the management of the whole testbed and its resources. The OCF also manages the life-cycle of the experiments and which resources are assigned to them. As previously mentioned, there are two types of resources to be shared: VMs and OpenFlow switches. On the one hand, the sharing of computational resources is a well-known technique with several commercial and open source products in widespread use. The virtualization software used by OFELIA is Xen. The OFELIA project has developed a

mechanism to integrate the management of the VMs under the OCF. On the other hand, the sharing of OpenFlow switches is based on FlowVisor, which is a special OpenFlow Controller that acts as a transparent proxy between the resource (i.e. the switch) and multiple Controllers. The FlowVisor is an external entity, which is able to delegate parts of the flowspace of the switch to different Controllers and isolate the control plane associated with each part. Therefore, the FlowVisor allows the slicing and virtualization of the switches, enabling the sharing of OpenFlow switches between several experiments at the same time. OFELIA has defined the VLAN tag as the mechanism to enforce the isolation, that is, the field to isolate the flowspace between experiments. FlowVisor inspects the OpenFlow protocol to enforce the isolation between experiments, and consequently, it depends on the OpenFlow version. Currently, FlowVisor only supports OpenFlow version 1.0. This means that OFELIA only allows sharing OFv1.0 resources due to its tight relation with FlowVisor.

VI. Conclusions

The achievements of the ALIEN project will have a large impact on research and commercial world. As a result, non-OpenFlow devices will be connected to the OFELIA infrastructure (OpenFlow network) creating an unique heterogeneous environment for testing protocols as well as control and management mechanisms. The research concerning the three components seems to be crucial for SDN deployment models. Three such models were proposed till now:

- switch based – SDN controller can send messages directly to the data plane implemented in network equipment;
- overlay network (tunnel based overlay approach) – hypervisor environment is implemented in a data source and end host; the SDN controller sends control messages directly to the SDN hypervisor switches;
- and hybrid (a combination of the switch based and overlay models) – this approach can be used to gradually migrate existing equipment to a new switch based model.

It is very likely that the hybrid model will be used by companies to evolutionary transform existing networks to SDN networks. Furthermore, in transition period they will require non-SDN equipment to communicate with SDN-native equipment. To perform this migration smoothly it is necessary to use a Hardware Abstraction Layer (HAL), which can help to force the non-SDN equipment to be controlled by the SDN controller. The Hardware Abstraction Layer for applying the OpenFlow protocol to the non-OpenFlow hardware which will be proposed by the ALIEN project seems to be a good solution of this problem. The ready-to-use software enabling selected non-OpenFlow devices to communicate with OpenFlow controller within OFELIA infrastructure will be an outcome of the ALIEN project.

In the heterogeneous environment consisted of non-OpenFlow and OpenFlow equipment it is possible to do a research on SDN network control and management issues, traffic control, scalability, security, etc. The results of ALIEN project will show that HAL allows to use non-OpenFlow equipment in the OpenFlow environment. The impact of the ALIEN project achievements to commercial world lies mainly in application of the HAL functionalities and extensions of OpenFlow protocol to selected non-OpenFlow devices. This will show that it is possible to extend OpenFlow functionality on different kind of programmable devices, and transition to the SDN network does not mean that all network equipment has to be new and SDN-native. The software produced by the ALIEN project may be used to connect selected devices to the OpenFlow network.

VII. Acknowledgments

The work described in this paper was financed from the EU-FP7 ALIEN project which is partially funded by the European Commission under grant agreement no. 317880.

References

[1] D. Dineley, Software-defined networking, 2012, [Online] http://www.infoworld.com/t/networking/software-defined-networking-206740

[2] K. Greene, MIT Technology Review, 2009, [Online] http://www.technologyreview.com/biotech/22120/

[3] B. Heller, N. McKeown B. Lantz, A Network in a Laptop: Rapid Prototyping for Software-Defined Networks, pp. 19:1-19:6, New York, NY, USA, 2010.

[4] T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner N. McKeown, OpenFlow: Enabling Innovation in Campus Networks, in ACM SIGCOM Computer Communication Review, vol. 38, pp. 69-74, April 2008.

[5] OpenFlow Switch Consortium, OpenFlow version 1.1.0 Switch Specification, February 2011, [Online] http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf

[6] Cisco Open Network Environment (ONE), [Online] http://newsroom.cisco.com/cisco_one

[7] Cisco Open Network Environment, [Online] http://www.cisco.com/web/solutions/trends/open_network_environment/index.html

[8] R. Dantu, T. Anderson, R. Gopal L. Yang, Forwarding and Control Element Separation (ForCES) Framework, RFC3746, April 2004.

[9] J. Hadi Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, J. Halpern A. Doria, Forwarding and Control Element Separation (ForCES) Protocol Specification, RFC5810, March 2010.

[10] K. Ogawa J. Hadi Salim, SCTP-Based Transport Mapping Layer (TML) for the Forwarding and Control Element Separation (ForCES) Protocol, RFC5811, March 2010.

[11] J. Hadi Salim J. Halpern, Forwarding and Control Element Separation (ForCES) Forwarding Element Model, RFC5812, March 2010.

[12] R. Haas, Forwarding and Control Element Separation (ForCES) MIB, RFC5813, March 2010.

[13] H. Khosravi, A. Doria, X. Wang, K. Ogawa A. Crouch, Forwarding and Control Element Separation (ForCES) Applicability Statement, RFC6041, October 2010.

[14] K. Ogawa, W. Wang, J. Hadi Salim E. Haleplidis, Implementation Report for Forwarding and Control Element Separation (ForCES), RFC6053, November 2010.

[15] O. Koufopavlou, S. Denazis E. Haleplidis, Forwarding and Control Element Separation (ForCES) Implementation Experience, RFC6369, September 2011.

[16] Open Networking Foundation: Into to OpenFlow, [Online] http://www.opennetworking.org/standards/into-to-openflow

[17] Open Networking Foundation, OpenFlow Switch Specification, September 2012.

[18] xDPd, [Online] https://www.codebasin.net/redmine/projects/xdpd/wiki/Architecture

[19] M. Suñé, A. Köpsel, V. Alvarez, T. Jungel, xDPd: eXtensible DataPath Daemon, EWSDN, Berlin, Germany, 2013.

[20] ROFL, [Online] https://www.codebasin.net/redmine/projects/rofl-core/wiki/Wiki?version=12