Design and Implementation of an OpenFlow Hardware Abstraction Layer

Damian Parniewicz[†] Roberto Doriguzzi Corin[♯] Lukasz Ogrodowczyk[†] Mehdi Rashidi Fard[*]
Jon Matias[§] Matteo Gerola[♯] Victor Fuentes[§] Umar Toseef[‡] Adel Zaalouk[‡]
Bartosz Belter[†] Eduardo Jacob[§] Kostas Pentikousis[‡]
[†]PSNC, [♯]CREATE-NET, [*]University of Bristol, [§]University of Basque Country, [‡]EICT GmbH

ABSTRACT

OpenFlow is a leading standard for Software-Defined Networking (SDN) and has already played a significant role in reshaping network infrastructures. However, a wide range of existing provider domains is still not equipped with a framework that supports wider deployment of an OpenFlow-based control plane beyond Ethernet-dominated networks. We address this gap by introducing a Hardware Abstraction Layer (HAL) which can transform legacy network elements into OpenFlow capable devices. This paper details the functional architecture of HAL, discusses the key design aspects and explains how HAL can support a number of network device classes. In addition, this paper presents the implementation details of HAL for hardware platforms such as DOCSIS (Data over Cable Service Interface Specification) and DWDM (Dense Wavelength Division Multiplexing) which have so far received little attention by the OpenFlow research community despite their wide real-world deployment.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design; C.2.3 [**Computer Communication Networks**]: Network Operations; C.2.5 [**Computer Communication Networks**]: Local and Wide-Area Networks; C.2.6 [**Computer Communication Networks**]: Internetworking—*Standards*

Keywords

SDN; OpenFlow; Hardware Abstraction; Control Plane; Network Virtualization; DWDM; DOCSIS

1. INTRODUCTION

Software Defined Networking (SDN) and in particular the OpenFlow protocol as an SDN enabler [10], is fostering a resurgence in networking research and a rethinking of network control and operations [7]. After making headline stories for data center applications, SDN is now used in infrastructure networks of a global scale [8], realizing the potential of distributed cloud computing [13]. SDN can unify the control plane for cloud and network infrastructure in order to deliver optimized performance. For example, John et al. [9] detail how carrier-grade networks can benefit from dynamic network service chaining which combines data center virtualization with in-network function virtualization. Along this line of work, we discuss in this paper how SDN principles can be employed to define a unified approach for legacy network management and new cloud management systems. Accordingly, we can thus automate and orchestrate a variety of provisioning processes making the distributed resources available on-demand through a single control plane.

OpenFlow has undergone many changes since its inception [11]. One could argue that the velocity and scope of changes in the protocol specification and its extensions have hindered to some extent wider implementation by vendors and third-party developers. Although the OpenFlow protocol has a single specification for each version, the diversity of network hardware and software platforms leads vendors and third-party users to create new OpenFlow libraries for each and every platform of OpenFlow implementation. This makes OpenFlow real-world deployment laborious and time consuming. Finally, the official OpenFlow specification is drafted with wired Ethernet platforms as a focal point and has yet to fully support other platforms such as circuit-switched and wireless platforms, despite recent efforts at ONF.

This paper addresses these problems by presenting the design and implementation of a Hardware Abstraction Layer (HAL) for non-OpenFlow enabled network elements. The main objective of HAL is to realize OpenFlow capabilities on network elements that do not have native support for OpenFlow and enable their integration in an OpenFlow deployment. In order to achieve this goal, the HAL architecture decouples the hardware-specific control and management logic, which is handled by hardware-specific software, from the network node abstraction logic which is implemented by the cross-hardware platform layer. This decoupling fosters reusability for different HAL components making them readily applicable to a range of hardware platforms as we have documented earlier [14]. In short, HAL is a viable and experimentally-tested concept for describing network device capabilities and controlling the forwarding behavior of all SDN and non-SDN capable hardware throughout a network with a single control plane based on OpenFlow. In practice, HAL hides the hardware complexity as well as the technology- and vendor-specific features, thus presenting a unified abstraction layer to standard, off-the-shelf OpenFlow controller(s). Our first contribution in this paper is a blueprint of the HAL architecture, a modular design with reusable components for faster development and deployment. The second contribution is an account of the implementation experience with several hardware platforms and the lessons learned in the process. We expect that HAL can facilitate wider deployment of OpenFlow and serve as a foundation block in distributed data center environments.

DCC'14, August 18, 2014, Chicago, Illinois, USA.

http://dx.doi.org/10.1145/2627566.2627577.

The remainder of this paper is organized as follows. Section 2 introduces HAL and Section 3 presents the HAL interfaces. Section 4 details HAL implementation on different hardware platforms and Section 5 relates HAL to previous research work. Finally, our conclusions and future work are presented in Section 6.

2. HARDWARE ABSTRACTION LAYER

The main purpose of HAL is to transform legacy network elements into OpenFlow-compatible devices through a set of abstractions. This approach will allow operators, on the one hand, to extend their OpenFlow-based control plane to legacy but valuable infrastructure and, on the other hand, to network modern OpenFlow switches with non-OpenFlow capable devices in a seamless manner. Considering the large array of devices that can be supported by HAL, the architecture must be based on a modular design which is easily extensible and compatible with heterogeneous network devices. By following a modular design approach, the behavior of any platform can be modified/extended without compromising the overall HAL architecture. From an implementation perspective, this approach also facilitates and accelerates HAL development for similar network platforms by exploiting module reusability. For instance, in [14] we discuss how HAL is used to produce adaptations to different programmable network platforms. The decoupling of the hardware-specific control and management logic from the network node abstraction allows us to hide the device complexity as well as the technology- and vendor-specific features from the control plane logic.

Figure 1 is a high-level schematic of the HAL functional architecture. The aforementioned decoupling is achieved through a split into two distinct sublayers, namely, the Cross-Hardware Platform Layer (CHPL) and the Hardware-Specific Layer (HSL). CHPL is responsible for node abstraction, virtualization and configuration mechanisms. HSL takes care of discovering the particular hardware platform and performing all required configuration using hardware-specific modules. The two sublayers communicate with each other through one of two interfaces, namely the Abstract Forwarding API and the Hardware Pipeline API depending on the type of the network device (see Section 3).

Note that the proposed HAL blueprint can also be employed by OpenFlow-capable devices as well. For example, a carrier can employ HAL to extend the functionality of OpenFlowd switches which have already been deployed in the field but support only a particular OpenFlow version (OF) out-of-the-box. HAL could be employed, for instance, in combination with an OF v1.0 switch to allow support for OF v1.2 features as we explain later.

2.1 Cross-Hardware Platform Layer

The Cross-Hardware Platform Layer (CHPL) is the hardware-agnostic software component which is common across all HAL-capable platforms. It comprises several independent modules responsible for device management, monitoring and control.

The CHPL *OpenFlow Endpoint* encapsulates all necessary control plane functionalities, maintains connectivity with the OpenFlow controller, and manages the forwarding state all the way to the platform drivers. On the management plane, CHPL presents a unified abstraction of the physical platform (physical ports, virtual ports, tunnels, and so on) to plugin modules hosted by a plug-in manager. This enables various plug-in modules to perform a variety of configuration- and management-related operations. Examples of plugin modules include a NETCONF/OF-CONFIG agent, a file-based configuration facility, and a Virtualization Agent (VA). The VA, as the name implies, adds resource virtualization features to the platform, such as, for instance, slicing the device to be shared

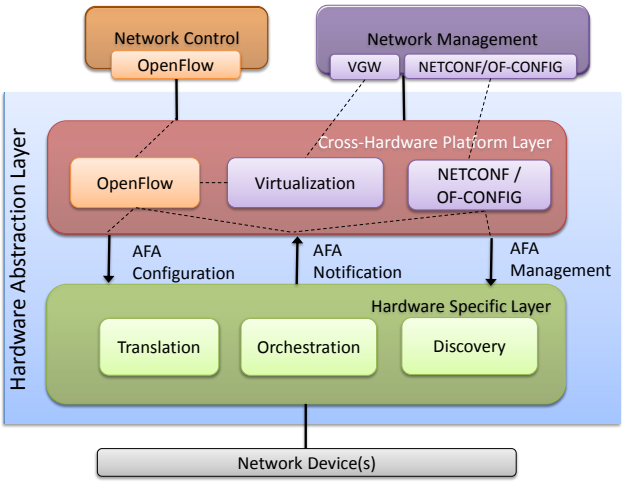

Figure 1: HAL high-level functional architecture schematic.

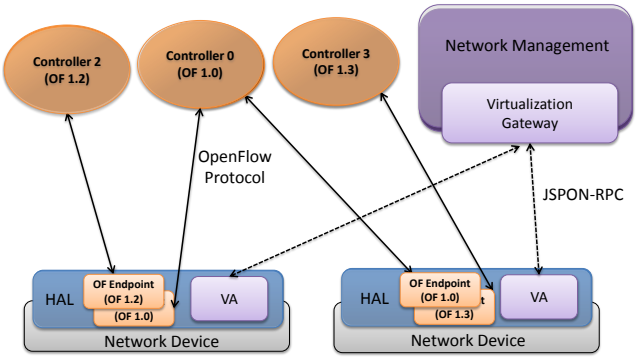

Figure 2: The HAL OpenFlow and virtualization interfaces.

among multiple users. VA interacts with the OpenFlow endpoint to perform flowspace slicing operations. VA, finally, obtains configuration details from a network management system (NMS) through the virtualization gateway (VGW).

As illustrated in Figure 2, the OpenFlow endpoint is a key CHPL component. It establishes the OpenFlow channels to the controller, exchanges OpenFlow protocol messages, implements OpenFlow session negotiation and takes care of state maintenance. In order to make the communication with different controller(s) OpenFlow protocol version-agnostic, the endpoint abstracts different protocol versions (e.g., OF v1.0, v1.2, or v1.3.2) to a common data model with a superset of features from all versions. The OpenFlow endpoint is initially configured with a specific OpenFlow version number so that it can inform accordingly a given controller about which version is to be used. The OpenFlow endpoint communicates with the HAL Hardware Specific Layer (HSL) via the Abstract Forwarding API (AFA) as illustrated in Figure 1. The OpenFlow endpoint common data model is part of AFA and can be easily extended to handle new header matches, new matching algorithms and new packet processing actions. See Section 3 for more details about the Abstract Forwarding API.

The CHPL *OpenFlow Pipeline* component implements the OpenFlow table(s) as illustrated in Figure 3. Due to performance considerations, the pipeline processes a packet abstraction of a given real network packet instead of processing the packet per se. The packet abstraction object consists of three attributes: i) a hardware plat-

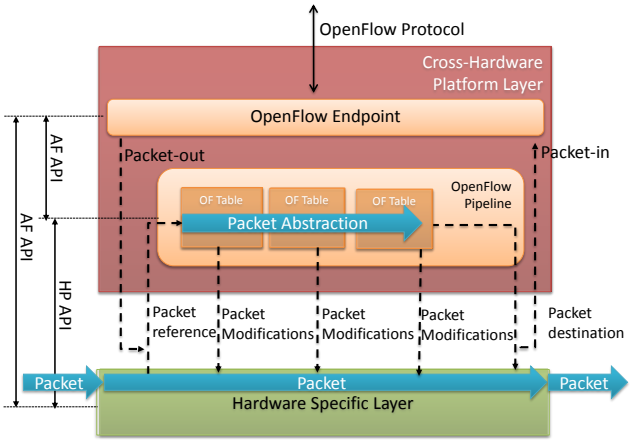

Figure 3: Workflow for a real packet and its abstraction by the CHPL OpenFlow pipeline.

form reference to a real network packet which, for instance, could be a pointer to the memory location where the packet is stored; ii) the OpenFlow action-set which is passed from one OpenFlow table to the next one and is modified according to the matched flow instructions; and iii) metadata about all successful match entries for diagnostic purposes.

Figure 3 illustrates how a network packet is processed by the HAL OpenFlow pipeline through its abstraction. Upon arrival of a network packet into the HSL, a packet reference is generated and is handed over to the CHPL OpenFlow endpoint. This triggers the creation of a packet abstraction object and the subsequent processing by the OpenFlow pipeline. Each OpenFlow table can immediately apply changes, such as header field value modification, tag additions and removals, to the packet located in HSL through the associated packet reference, or it can modify the OpenFlow action-set. Each table can also be used to determine the final destination of the packet. This includes requesting to send the packet to a specific network port on the device, dropping it, or forwarding it to the controller. In the latter case, the pipeline is effectively requesting a *packet-in* event generation from the HSL, which must send the entire or a part of the real packet to the OpenFlow endpoint. The corresponding *packet-out* event which contains a packet from the OpenFlow controller is also processed via the pipeline. However, the packet bytes must be stored at the HSL before pipeline processing can begin. In addition to supporting multiple OpenFlow tables and action-set as mentioned above, the OpenFlow pipeline provides all other OpenFlow features such as priority matching, flow entry expiration, group table, meter table and counter objects.

The OpenFlow pipeline is currently implemented in software based on the Revised OpenFlow Library (ROFL; see `http://roflibs.org`) as detailed in [16, 20]. It is a high performance implementation of the OpenFlow pipeline which can be deployed on a broad spectrum of available CPUs (for software-switch solutions) and on modern NPUs, which can be programmed in ANSI-C and provide better traffic throughput for a software-based packet processing implementation.

The *Virtualization Agent* (VA) is a CHPL plugin module which provides a distributed slicing mechanism for HAL devices. Like other virtualization approaches [17, 3], its main objective is to allow the isolated execution of multiple parallel experiments on the same physical substrate. VA has been designed with the following goals: a) avoid single points of failures, b) provide an OpenFlow version-agnostic slicing mechanism, and c) minimize the latency caused by slicing. VA avoids single point of failures by taking a different path from earlier approaches which virtualize network resources through an additional layer on the control channel. Unfortunately, upon failure, said approaches take down all running slices. In contrast, the VA architecture has only one centralized element: the Virtualization Gateway (VGW) as illustrated in Figure 1. If VGW fails, new slices cannot be instantiated, but on the other hand, slices which are already in operation are not affected by the failure. Similarly, in order to reduce the latency overhead in slicing operations, VA is spared from inspecting the OpenFlow protocol and establishing the associated TLS connections. In doing so, the VA becomes OpenFlow version-agnostic and can work with any control plane protocol (even other than OpenFlow).

For each incoming packet, the header fields 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 generates the *packet-in* message by using the protocol version used for the communication with the controller. However, if no match is found, the VA tells the lower layers to drop the packet. On the other end, 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 the OpenFlow protocol version, the VA intercepts the actions and the related flow match after they are decapsulated from the OpenFlow message and before they are inserted into the switch flow table. The actions are checked against the controller flowspace to ensure that the controller is not trying to control traffic outside of its own 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, thus preventing interference between different slices.

2.2 Hardware Specific Layer

The key idea behind the Hardware Specific Layer (HSL) is to deal with the diversity of network platforms and their communication protocols and thus overcome the complexity of implementing the OpenFlow protocol on different hardware platforms. In the real world, every network element or platform comes with its own protocol or API for communicating, controlling and managing the underlying system. In HAL, HSL is responsible for hiding the complexity and heterogeneity of underlying hardware control for message handling and providing a unified and feature-rich interface to CHPL via two northbound interfaces. On the southbound side, HSL is in direct contact with, and dependent upon, the underlying hardware in terms of communication protocol(s) and programming language(s). As a result, HSL developers must deal with the implementation peculiarities for each platform.

The embraced modular design approach makes HAL flexible enough to support several hardware platforms as different HSL modules take care of the underlying hardware heterogeneity. HSL has been designed so that module changes do not affect CHPL, which is hardware independent and, in most cases, there is no need to alter the overall software/hardware architecture. Next, we describe the main HSL modules as illustrated in Figure 1.

Discovery – In order to initialize CHPL, information about the network device(s) must be provided by HSL. Bootstrapping information includes: i) a list of devices working together as a single hardware platform instance and controlled by a single OpenFlow agent instance. For each device, 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, that is, how all devices within a hardware platform in-

stance are interconnected. Discovery can be manual (e.g., the platform administrator creates static configuration files containing the required information which are loaded during the HSL initialization phase) or, perhaps preferably, automatic: HSL queries each device for all necessary information and reacts to new notifications coming from the device. Combinations of both approaches are also possible. Depending on the implementation and the platform, the discovery process could be active solely during HSL initialization or executed on a continuous basis.

Orchestration – In some cases, the hardware platform comprises multiple components acting independently, but controlled centrally. This is the case of Data Over Cable Service Interface Specification (DOCSIS) and Gigabit Ethernet Passive Optical Network (GEPON) deployments, for example. The orchestration procedure is intended to send configuration and control commands to all hardware components that must be engaged in the request handling in a synchronized, ordered and atomic fashion. Orchestration should be able to report failures, recover from configuration errors on a single hardware component, and restore the initial state of all hardware components.

Translation – The Translator module in HSL is responsible for the translation of data and action models used in CHPL (mostly OpenFlow-based) to the specific device protocol syntax and semantics, and vice versa. The Translator acts as middleware between the OpenFlow switch model and the underlying physical device. Due to the 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 the OpenFlow switch model into platform-specific interface commands and processor instructions. For the majority of the hardware platforms considered, the translation functionality is foreseen to be stateful, which requires storing information about every handled OpenFlow entry and its translation to specific device commands. This allows one to modify or delete the device applied reconfiguration which refers to a given flow entry.

3. HAL INTERFACES

As per Figure 3, two common interfaces are exposed by HSL towards the Cross-Hardware Platform Layer. Both interfaces have been designed to minimize the effort required to implement a new hardware driver for obtaining OpenFlow control over the targeted hardware platform. In addition, the HAL northbound interfaces allow a particular component of a network device to communicate with higher-level components. HAL provides three northbound interfaces for OpenFlow-based control, network virtualization support, and configuration management. The former enables the communication between OpenFlow controller(s) and the devices while the virtualization management interface is used to configure the Virtualization Agent via a Network Management System (NMS).

3.1 Abstract Forwarding API

The Abstract Forwarding API (AFA) can be used for any hardware platform including closed-box platforms. It provides interfaces for management, configuration and receiving HSL event notifications. The management and configuration parts of AFA must be implemented by HSL and called by CHPL. Respectively, notifications are provided to CHPL and are invoked by HSL. The AFA management part is in charge of hardware driver initialization, network interface discovery, logical switch creation and destruction, network interface attachment and detachment to/from logical switches, and administratively enabling and disabling network interfaces. The AFA notification part generates events related to adding and removing network interfaces within the hardware platform, switch port attribute or state modifications, flow entry expiration and incoming packet arrival for the controller. By using AFA a hardware platform can be logically partitioned into several OpenFlow-controlled data path elements.

3.2 Hardware Pipeline API

The main goal of the Hardware Pipeline API (HPA) is to minimize the development effort required to implement the HAL hardware driver on programmable network platforms. This allows us to deploy and run generic C/C++ code on different hardware, for instance, on Cavium Octeon, Broadcom Triumph2, Intel DPDK, and EZchip NPS processors. HPA is a low-level interface which provides access to network packet operations, memory management, mutex and counter operations, which are typically realized differently on each programmable platform. The key benefit from using HPA is that the hardware driver does not have to implement the OpenFlow pipeline itself and can thus reuse the CHPL pipeline implementation presented earlier in this paper. Due to space considerations, we refrain from going through the low-level implementation details, which are documented in [14, 16, 20].

3.3 Northbound Interfaces

As illustrated in Figure 2, the northbound OpenFlow interface connects each HAL-running device (and OpenFlow switches, of course) to a controller. Through this interface, the controller controls the device, receives events from the device, and sends packets out of the device as expected [11]. The OpenFlow channel is encrypted using TLS, but could also be operated directly over TCP. It should be noted that the HAL OpenFlow interface is provided by OpenFlow endpoints instances (see Figure 2). Multiple instances are required when different versions of the protocol are used on the same device. On the other hand, multiple controllers using the same version of the protocol are handled by a single OpenFlow endpoint.

The JSON-RPC Interface is used by the CHPL Virtualization Agent (VA) to slice the overall flowspace across many OpenFlow controllers based on the configuration received from the NMS, which communicates with the VA through the Virtualization Gateway (VG). The management interface between VA and VG is implemented using JSON RPC 2.0 where each request exchanged between VG and VA will be implemented following a wire protocol. Finally, the CHPL OF-CONFIG/NETCONF plugin illustrated in Figure 1 equips network administrators with a management interface to configure the underlying HAL devices with several parameters, such as the OpenFlow controller IP address and switch datapath IDs.

4. IMPLEMENTATION

HAL has been implemented and is in active use over a variety of programmable as well as closed-box hardware as illustrated in Figure 4. The Figure clearly indicates the demarcation points for AFA and HPA as introduced in the previous section. The remainder of this section summarizes the implementation particulars for three types of hardware platforms, namely programmable platforms, DOCSIS, and DWDM.

The term **programmable platforms** refers to network devices which allow their data plane to be programmed to perform packet processing. HAL has been implemented using xDPd (available from `http://xdpd.org/`) on several types of programmable platforms, including programmable silicon gateways (e.g. NetFPGA), traditional NPU (e.g. EZchip NP-3), multicore CPUs with hardware network enhancements (such as the Cavium Octeon Plus CN5650) and standard CPUs with software network enhancements (such as Intel Atom, Core, Xeon with DPDK); see also [14, 20]. In

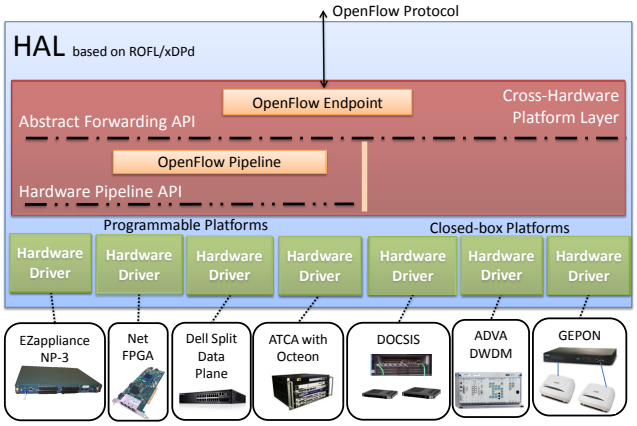

Figure 4: HAL implementation over different hardware.

all cases, the CHPL OpenFlow pipeline was reused in coordination with the Hardware Pipeline API. However, in order to fully utilize the hardware performance of NetFPGA and NP-3 processors, the OpenFlow pipeline was also implemented directly in hardware and controlled via AFA. In such platforms, the CHPL OpenFlow pipeline acts as the slow-path packet processing containing OpenFlow flow entries which are not supported by the hardware fast-path pipeline implementation.

Since **DOCSIS** is a closed-box platform, the residential gateway (RG) and aggregation switch are used as helper boxes to implement all functionality not supported by DOCSIS. In the end, the whole setup operates as a single OpenFlow device. A HAL-based proxy sits between the OpenFlow controller and the DOCSIS platform. The most relevant part from the HAL-based proxy is the HSL implementation for DOCSIS. Since the topology model used for DOCSIS is known (i.e. DOCSIS resources surrounded by helper boxes), the discovery functionality has been reduced to the dynamic detection of the customer devices (CPE) in the system, that is, adding and removing cable-modems (CMs). When a new CPE is detected, it is automatically exposed by the proxy as a new virtual interface to the controller.

The most relevant HSL component for DOCSIS is the orchestration module, which exposes the whole set of resources as a unique DataPath Identifier (DPID) with unique virtual port identifiers. By using this virtual model, the messages coming from/to the controller are processed to hide the network internals. For instance, the orchestrator is responsible for splitting the incoming OpenFlow messages into several messages for each particular device, i.e. the residential gateway, DOCSIS and aggregation switch, to perform the requested action (e.g. stats report, configuration setting etc.). Finally, the translation module is responsible for mapping the virtual identifiers (i.e. DPID and ports) to the physical ones, and vice versa. This functionality is closely related to orchestration due their inter-dependence.

The HAL implementation for **DWDM** ROADM (Reconfigurable Optical Add/Drop Multiplexer) is different from that found on packet switching network devices. In the optical domain, the control and data planes are separated as lightpaths in the optical data path need to be provisioned before transmission can commence. Prior to that, all device configuration and path setup signaling is completed. This is due to the fact that we are lacking the notion of packets in the optical domain. Further, circuit switches have no visibility of the payload, thus no packets can be forwarded to the OpenFlow network controller to make a decision. Due to this nature of the optical domain, the OpenFlow protocol had to be extended and aligned with OpenFlow addendum v0.3 for packet switches to support this differentiation in operation.

We have further extended Stanford's addendum to the protocol to add switching constrains and power equalization messages specific to the device. A software entity (e.g. a virtual machine) acts as an agent for the optical device and creates the OpenFlow abstraction for the controller on top. HAL is implemented inside the agent and a proprietary SNMP library is used as the management interface of the optical switch. A resource model has been developed at HSL that glues the SNMP and OpenFlow abstraction layers. The translation module takes care of the translation between OpenFlow messages and SNMP hardware-specific messages. The OpenFlow cross-connect table (equivalent to Hardware OpenFlow pipeline) is constructed on top of the device driver giving an abstracted view of the circuit switch using an extended version of OpenFlow.

Further implementation details as well as pointers to open source code are available in [14, 20].

5. RELATED WORK

The work presented in this paper is the culmination of the efforts in the FP7 ALIEN project which aims to realize OpenFlow capabilities on network elements that do not have native support for OpenFlow and therefore enable their integration in an OpenFlow deployment, such as an SDN experimental facility. In doing so, the work on HAL can be used as a basis for further development in carrier-grade networks to bring legacy infrastructure in line with expectations for network function chaining and distributed cloud computing. For example, a cable operator could employ HAL to control the already-deployed DOCSIS infrastructure using OpenFlow as discussed in Section 4, and combine it with distributed mapping and routing algorithms [13], content-aware traffic engineering [15], and dynamic service definitions as discussed in [9].

In the area of SDN experimental facilities, previous work has been undertaken during the FP7 OFELIA project [19] to provide one of the first OpenFlow-enabled experimental facilities for third-party use which includes an optical OpenFlow testbed [1]. The successful implementation of an OpenFlow lightpath testbed in OFELIA was an enabler for other projects such as Fed4Fire and FIBRE [2]. Although these projects do implement an OpenFlow lightpath data plane, they do not follow a standard-compatible approach towards enabling OpenFlow protocol support for lightpath devices. The ALIEN HAL in this regard is the first step towards establishing a standard framework approach for lightpath OpenFlow implementation.

Pyretic [12], Fresco [18], and Frenetic [5] are, among others, tools to develop sophisticated policy-based controller applications using a high-level language. Our work on HAL extends the reach of such tools to new domains. Virtualization has also been researched and discussed for enabling an SDN node to execute multiple applications simultaneously. FlowVisor [17], XNetMon [4], and Vertigo [3] are examples of virtualization mechanisms which feature varying levels of isolation. Having an adequate view of the network is also of utmost importance to SDN applications. This is achieved either by representing the global topology as an annotated graph that can be configured and queried [6], or through a mapping between a representation of the physical topology and a simplified representation of the network [3].

Overall, and to the best of our knowledge, few initiatives have been undertaken to empower legacy network elements with even basic OpenFlow features. Notably, the ONF Forwarding Abstractions working group has initiated work in this direction, but this is still at an early stage. In contrast, our work on the ALIEN HAL

provides a fully-fledged design specification and implementation which addresses in a thorough and detailed manner the problem.

6. CONCLUSION

OpenFlow paves the way for advanced functionalities in new programmable network deployments. Currently, however, OpenFlow support is lacking in production environments where most of the forwarding devices are based on either closed platforms or legacy hardware which is incompatible with the protocol. This work addresses this gap through the introduction of the ALIEN Hardware Abstraction Layer (HAL), which provides a platform for development and deployment of OpenFlow on network elements that do not support the protocol out-of-the-box. In short, the ALIEN HAL is a software architecture and implementation which aims to complement conventional hardware platforms and provide a framework for network infrastructures that are not natively designed to support an OpenFlow-based control plane.

The HAL architecture emphasizes the decoupling of hardware-specific control and management logic from the OpenFlow node abstraction logic. HAL comprises two sub-layers, namely, the Cross Hardware Platform Layer (CHPL) and the Hardware Specific Layer (HSL). By doing so, we can add new features, such as on-demand programmability, or introduce an OpenFlow control plane to new hardware platforms without redesigning the HAL architecture. We also described how the HAL design incorporates a distributed virtualization mechanism that enables multi-version OpenFlow switch network deployments and allows the HAL-enabled devices to be shared among multiple experimenters.

As part of our ongoing work, we are introducing HAL devices to the OFELIA pan-European SDN experimental facility [19], significantly extending the options that researchers have at their disposal when it comes to experimenting with OpenFlow-capable network equipment.

7. ACKNOWLEDGMENT

This work was conducted within the framework of the FP7 ALIEN project, which is partially funded by the Commission of the European Union under grant agreement no. 317880. We are grateful to Andreas Köpsel, Hagen Woesner, Marc Suñé Clos, Tobias Jungel, and all those who have been actively contributing to the development of ROFL and xDPd over the years.

8. REFERENCES

[1] S. Azodolmolky, R. Nejabati, E. Escalona, R. Jayakumar, N. Efstathiou, and D. Simeonidou. Integrated OpenFlow–GMPLS control plane: an overlay model for software defined packet over optical networks. *Opt. Express*, 19(26):B421–B428, Dec 2011.

[2] M. Channegowda, R. Nejabati, and D. Simeonidou. Software-Defined Optical Networks Technology and Infrastructure: Enabling Software-Defined Optical Network Operations. *J. Opt. Commun. Netw.*, 5(10):A274–A282, Oct 2013.

[3] R. Doriguzzi Corin, M. Gerola, R. Riggio, F. De Pellegrini, and E. Salvadori. VeRTIGO: network virtualization and beyond. In *EWSDN*, pages 24–29. IEEE, 2012.

[4] N. C. Fernandes and O. C. M. B. Duarte. XNetMon: A network monitor for securing virtual networks. In *ICC*, pages 1–5. IEEE, 2011.

[5] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. Frenetic: A network programming language. In *ACM SIGPLAN Notices*, volume 46, pages 279–291. ACM, 2011.

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[7] E. Haleplidis, S. Denazis, K. Pentikousis, J. H. Salim, D. Meyer, and O. Koufopavlou. SDN Layers and Architecture Terminology. Internet Draft, draft-haleplidis-sdnrg-layer-terminology (work in progress), 2014.

[8] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM*, pages 3–14. ACM, 2013.

[9] W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu. Research directions in network service chaining. In *SDN4FNS*, pages 1–7. IEEE, 2013.

[10] N. McKeown. Software-defined networking. *INFOCOM keynote talk*, 2009.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[12] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, et al. Composing software defined networks. In *NSDI*, pages 1–13, 2013.

[13] S. Narayana, W. Jiang, J. Rexford, and M. Chiang. Joint server selection and routing for geo-replicated services. In *DCC*, pages 423–428. IEEE, 2013.

[14] L. Ogrodowczyk, B. Belter, A. Binczewski, K. Dombek, A. Juszczyk, I. Olszewski, D. Parniewicz, R. Doriguzzi Corin, M. Gerola, E. Salvadori, K. Pentikousis, U. Toseef, H. Woesner, M. Rashidi Fard, M. Huarte, E. Jacob, J. Matias, V. Fuentes, M. Michalski, and R. Rajewski. Hardware Abstraction Layer for non-OpenFlow capable devices. In *TERENA Networking Conference*, May 2014.

[15] I. Poese, B. Frank, G. Smaragdakis, S. Uhlig, A. Feldmann, and B. Maggs. Enabling content-aware traffic engineering. *ACM SIGCOMM Computer Communication Review*, 42(5):21–28, 2012.

[16] M. Rashidi (Ed.) et al. Specification of Hardware Abstraction Layer. FP7 ALIEN Deliverable D2.2, available at `http://www.fp7-alien.eu`, 2014.

[17] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. M. Parulkar. Can the production network be the testbed? In *OSDI*, pages 1–6. USENIX, 2010.

[18] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, and M. Tyson. Fresco: Modular composable security services for software-defined networks. In *NDSS*, 2013.

[19] M. Suñé, L. Bergesio, H. Woesner, T. Rothe, A. Köpsel, D. Colle, B. Puype, D. Simeonidou, R. Nejabati, M. Channegowda, et al. Design and implementation of the OFELIA FP7 facility: the European OpenFlow testbed. *Computer Networks*, 61:132–150, 2014.

[20] U. Toseef (Ed.) et al. Report on implementation of the Common Part of an OpenFlow Datapath Element and the Extended FlowVisor. FP7 ALIEN Deliverable D2.3, available at `http://www.fp7-alien.eu`, 2014.