Network Configuration in OpenFlow Networks

Adel Zaalouk and Kostas Pentikousis

EICT GmbH, Berlin, Germany
{adel.zaalouk,k.pentikousis}@eict.de

Abstract. Software-defined networking (SDN), and in particular networks based on an OpenFlow control plane, are expected to take significant share in upcoming deployments. Network programmability has emerged as a particularly desirable property for such new deployments, in which logically centralized software will be able to both control and manage operation. This paper focuses on one aspect of network management, namely configuration, in light of the ongoing work in the FP7 ALIEN project to augment a variety of devices with an OpenFlow control plane. In particular, we review management for programmable networks and present how software-defined control can be complemented with software-defined configuration.

Key words: SDN, OpenFlow, Hardware Abstraction Layer, NETCONF

1 Introduction

Packet-switched computer networks are based on network elements that run distributed control software that is complex to configure. While network operators ought to maintain a complete view of the actual network state, in practice, they have only coarse-grained tools at their disposal. For instance, network device configuration can often require human intervention based on Command Line Interface (CLI) interaction. CLIs are cumbersome to use, error-prone, and may vary widely across different vendors, so management complexity increases even more. Network administration might lead to configuration errors, which are difficult to detect. But more in the interest of this work is that the current network configuration paradigm is not really programmable.

The emergence of software-defined networking (SDN) [1] introduced new opportunities in network research [2]. SDN advocates a logically centralized control plane with advanced programming capabilities based on a control-data plane separation. By breaking the tight coupling between the control and data plane both can evolve independently. Programmability fosters the development of software that can dynamically alter network-wide behavior, thus enabling testing of research ideas in a speedier manner without having to always resort to simulation tools. In particular, a programmable control plane based on OpenFlow [3] is expected to accelerate network innovation and the rollout of new services. OpenFlow per se, however, is not well-suited for the management plane. To address this gap, the Open Networking Foundation (ONF;

www.opennetworking.org) introduced the OpenFlow Management and Configuration Protocol (OF-CONFIG) [4], which uses the Network Configuration Protocol (NETCONF) [5] as the transport protocol.

Originally OpenFlow was designed for Ethernet and ASIC network devices, leaving behind a large set of other platforms such as wireless and point-to-multipoint (DOCSIS, optical). To address this, the FP7 ALIEN project (www.fp7-alien.eu) is working on the design and implementation of a Hardware Abstraction Layer (HAL) which enables network devices that do not support the OpenFlow protocol and switch model natively, to be included in the OpenFlow control plane of a network deployment. In particular, ALIEN aims to enable such devices (called "ALIEN devices" in the remainder of this paper) to be controlled through OpenFlow thereby extending an OpenFlow control plane to new classes of devices [6]. Within this framework, we are interested in the design and implementation of a HAL for ALIEN devices that can be both controlled and managed in a software-defined manner. As ALIEN devices will be introduced in the OFELIA OpenFlow experimental facility [7], their management should be programmatically enabled. In more general terms, this paper contributes to the ongoing discussion about what does SDN entail [8], and in particular how operators can employ software to both control and manage not only soon-to-be-deployed OpenFlow compatible devices but legacy equipment as well.

We start our exploration with an overview of related work (§2) and continue with a discussion of the motivation behind and salient characteristics of the programmable networks paradigm in light of the SDN emergence (§3). We consider recent work at the IRTF SDN RG and map accordingly the HAL design 4. Finally, we introduce network configuration based on NETCONF for the ALIEN HAL (§5) and present workflows for software-defined configuration for OpenFlow networks (§6). We conclude this paper in §7 and outline future work items.

2 Related Work

OpenFlow networks require management just like traditional networks. But, given their edge in programmability, and the fact they would be controlled by software, management should also follow along, thus enabling programmability both in the control plane and the management plane. The ONF-standardized OF-CONFIG protocol takes advantage of NETCONF as a configuration protocol and YANG [9] for OpenFlow switch data modeling.

Prior to its adoption by ONF, NETCONF was studied and compared against other popular configuration management protocols. Hedstrom et al. [10] compare the performance of NETCONF with SNMP in a testbed, considering protocol bandwidth use, number of packets, number of transactions, operations time, and so on. They conclude that NETCONF is much more efficient for configuration management than SNMP (e.g., requires fewer transactions over managed objects). Another empirical study [11] compares NETCONF and SNMP indicating that NETCONF is more efficient in handling a large number of configura-

tion transactions. With respect to implementation, there are several open-source frameworks for developing NETCONF clients and servers [12, 13]. Tran et al. [14] introduce a plan for testing and verifying existing NETCONF implementations. Based on performance as well as other considerations, NETCONF was chosen as reasonable candidate for configuring OpenFlow networks.

Several projects have sought to incorporate NETCONF as a tool for network configuration management in their networks. Munz et al. [15] present an XML-based data model for NETCONF to cover all common configurable parameters for network monitoring. Xu et al. [16] introduce a NETCONF implementation using a RESTful web service in the context of Internet of Things (IoT), while applications of NETCONF in a military context were introduced in [17].

None of this earlier work relates to SDN. According to [8], the SDN architecture should decouple control and management functionalities. Although several research projects focus on extending and enhancing network control using OpenFlow, the research effort towards improving the SDN management plane remains minimal to the best of our knowledge. For example, HybNET [18] is an automated network management framework for "hybrid" networks (i.e., SDN and legacy infrastructures). HybNET uses NETCONF for the management of legacy network switches. Sonkoly et al. [19], on the other hand, introduce an OpenFlow virtualization framework which employs NETCONF for managing native OpenFlow devices. Unlike HybNET which uses NETCONF to manage legacy switches, or the OpenFlow virtualization framework that uses NETCONF to manage native OpenFlow devices, this paper proposes to integrate the management capability provided by NETCONF into non-native HAL-enabled OpenFlow devices (S4). Although OF-CONFIG uses NETCONF for managing native OpenFlow devices, we employ NETCONF to manage HAL-enhanced network devices. As such, the contributions of this paper compared to earlier work include a thorough study of NETCONF in the context of OpenFlow networks, the architectural mapping of the ALIEN HAL design and the SDN Layers work currently under adoption in the SDN RG (`irtf.org/sdnrg`), and the introduction of a proposal for employing NETCONF for software-defined configuration.

3 The Programmable Networks Paradigm

Arguably, the Internet has become extremely difficult to evolve both in terms of its physical infrastructure and network protocols. As the Internet was only designed for tasks such as sending and receiving data with best effort guarantees only, there has been continuous interest to evolve the current IP packet-switched networks to address the new challenges including Quality of Service (QoS), multicasting, and network security. One of the first steps towards making networks more programmable was the introduction of Active Networking (AN) [20]. The main idea behind AN was to enable network devices to perform custom computations on packets. To do so, two different models were introduced [1]. First, in the "packet capsules model", network programs were attached to (possibly each) packet and sent across the network to target devices such as ANTS [21]. Second,

in the "programmable network devices model" [22], network devices were preconfigured with several service-logic modules. When a packet arrives, its headers are matched and sent to the appropriate module. Largely, the AN vision did not come to pass,for various reasons [1]. On the one hand, a clear migration path was not evident. On the other, no operator pressing need was basically addressed by AN. That said, at the core, AN networks aimed for having programmability in the data plane. This concept evolved over the years and took hold in network devices that are known as middleboxes now. The concept of middlebox programmability has received quite some attention recently. For example, xOMP [23], an eXtensible Open MiddleBox software architecture, allows for building flexible, programmable, and incrementally scalable middleboxes based on commodity servers and operating systems. Similarly, SIMPLE [24], a programmable policy enforcement layer for middlebox-specific traffic steering, allows network operators to specify middlebox routing policies which take into account the physical topology, switch capacities and middlebox resource constraints.

Another important step that has been taken towards network programmability is the separation of the control and data planes [1]. This decoupling enables the two planes to evolve separately, and allows new paradigms where logically centralized control can have a network-wide view making it easier to infer and direct network behavior. Examples of such separation can be proposed earlier in Routing Control Point (RCP) [25] and, of course, ForCES [26]. So far, the OpenFlow approach towards control and data plane separation focused on control plane rather than data plane programmability as compared to AN. Despite the intellectual contributions that resulted from such separation, the AN deployment strategy was not pragmatic (i.e., required deployment of new hardware), as a result AN was not widely adopted. OpenFlow defines a standard interface between the control and the data plane that goes hand in hand with the concept of Network Operating Systems (NOS) [27] with the goal of providing an abstraction layer between network state awareness and control logic.

Haleplidis et al. [8] provide a detailed description of the SDN layers architecture, aiming to provide a clearer view of the emerging paradigm, which is often cluttered with marketing terms. By dividing the SDN architecture into distinct planes, abstraction layers and interfaces, the draft aims to clarify SDN terminology and establish some commonly accepted ground across the SDN community.

As shown in Fig. 1, the *Forwarding Plane* represents parts of the network device which are responsible for forwarding traffic. The *Operational Plane* in the part of the network device which is responsible for managing device operation. The *Control Plane* instructs the network devices, especially the forwarding plane on how to forward traffic. The *Management Plane* is responsible for configuring and maintaining one or more network devices. Most of the management plane interactions happens with the forwarding plane. The draft also defines three layers, as follows. The *Device and Resource abstraction Layer (DAL)* provides a point of reference for the device's forwarding and operational resources. The *Control Abstraction Layer (CAL)* provides access to the control plane southbound interface. Finally the *Management Abstraction Layer (MAL)* provides

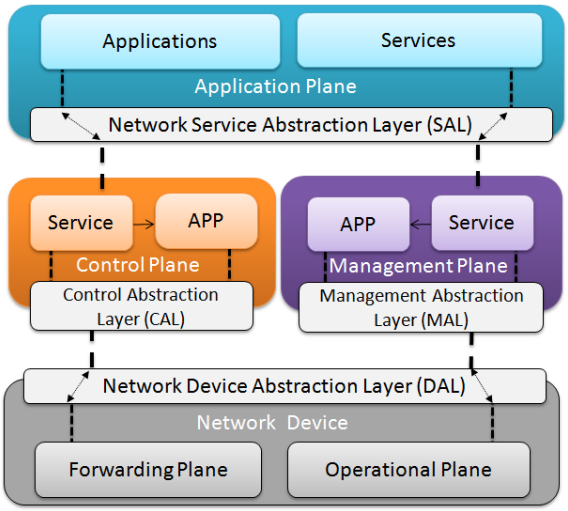

Fig. 1. High-level View of the SDN architecture

access to the management plane southbound interface. Fig. 1 illustrates all functional components of the SDN architecture and provides a high-level overview of the SDN architecture abstractions including control and management plane abstractions. The architecture visibly decouples management, control and forwarding functions including their interfaces. Of course, this is an abstract model. In practice, the entities providing these functions/planes could be collocated. In this paper, our focus lies on the management and control southbound interfaces, as we explain next.

4 Abstraction Layer for ALIEN Devices

As mentioned earlier, the OpenFlow protocol was mainly designed to support ASIC and campus Ethernet switches with little or no regard for other plat-forms such as circuit-switched, wireless and optical. Unfortunately, these platforms are often closed and changes cannot be made to the device per se in order to make it natively compatible with an OpenFlow control plane. To overcome this challenge, the FP7 ALIEN project defines a Hardware Abstraction Layer (HAL) [6] which aims to enable communication with devices that do not natively support OpenFlow through a set of hardware abstractions. These HAL-enhanced devices will be controlled in the same manner as their native OpenFlow counterparts. Each hardware device that does not natively support OpenFlow will have a Hardware Specific Layer (HSL) that translates OpenFlow protocol messages coming from the controller to device-specific commands. In addition to having

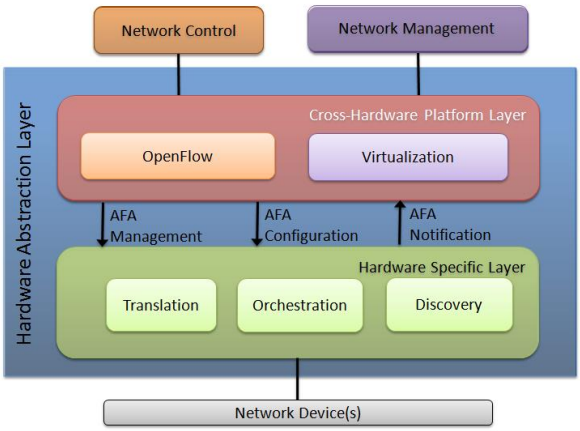

Fig. 2. HAL Architecture

the ALIEN devices controlled via OpenFlow, HAL provides configuration management functionalities through protocols such as NETCONF as we discuss later in this paper (§5).

Fig. 2 illustrates the HAL architecture [6]. Network Control represents controlling elements such as the OpenFlow controllers. Network Management allows network administrators to configure the underlying ALIEN devices with parameters such as controller's IP address. The Cross-Hardware Platform Layer contains hardware-agnostic components such as the OpenFlow-endpoint that mediates the communication between the ALIEN devices and the OpenFlow controllers, and the virtualization agent which enables the device to be controlled by multiple controllers. The HSL contains hardware specific sub-components to enable translation of OpenFlow messages to device specific messages [7], or to discover the underlying hardware device components and relay incoming messages to each of these components (i.e., Orchestration). Finally, the network devices constitute the data plane in an ALIEN network deployment. Finally, the Abstract Forwarding API (AFA) is the interface used for relaying, management and control messages from the Cross-Hardware Platform Layer to the HSL. Due to space restrictions we cannot delve into details here; interested readers are referred to [7, 6].

We map the ALIEN HAL to the SDN layers in Fig. 3. Essentially, DAL in the SDN architecture is realized by the HAL in ALIEN, which handles the translation of OpenFlow messages to device-specific messages. Generic network devices are mapped to ALIEN devices, which do not support OpenFlow natively. The control plane is realized by OpenFlow controllers, and the management plane is implemented using NETCONF. The application plane is mostly outside the scope of the ALIEN architecture.

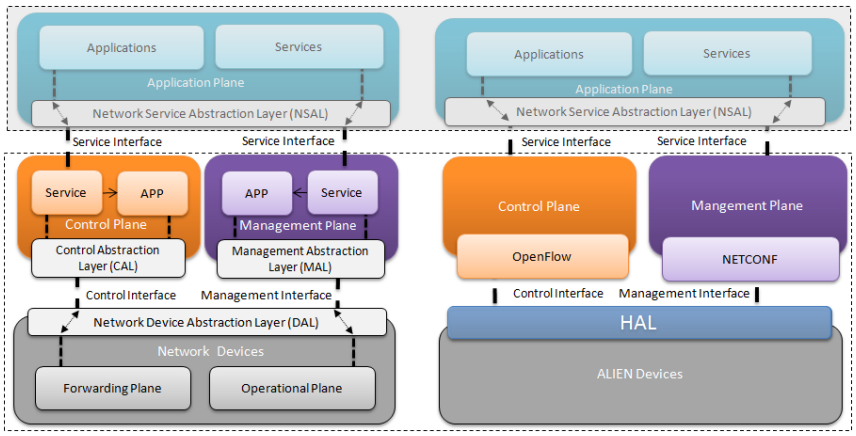

Fig. 3. Mapping of HAL to the SDN Layers Architecture

5 HAL Network Configuration

Using OpenFlow the control plane can communicate with the data plane to perform several functionalities such as adding or removing flow-rules and collecting per-flow, per-table statistics. However, this assumes that the OpenFlow switches are already configured with various parameters such as the IP address(es) of the controller(s). Here it is important to distinguish time-sensitive control functionalities for which OpenFlow was designed (e.g., modifying forwarding tables, matching flows) from non-time-sensitive management and configuration management functionalities which are essential for the operation of the OpenFlow-enabled device (e.g., controller IP assignment, changing switch port administrative status, configuring datapath-ids, etc.) SNMP could be used for such configuration tasks.

However, as per [9], SNMP has several drawbacks, including unreliable transport of management data (e.g., UDP); no clear separation between operational and configuration data; no support for roll-backs in case of errors / disaster; lack of support for concurrency in configuration (i.e., N:1 device configuration); and no distinction between transaction models (e.g., running, startup, and candidate). To address such shortcomings, the NETCONF protocol was developed. NETCONF provides several key features such as the ability to retrieve configuration as well as operational data, rich configuration management semantics including validation, rollbacks and transactions, and configuration extensibility based on the capabilities exchange that occurs during initiating the session initiation. Furthermore, NETCONF's transactional models constitutes candidate, running and startup data-stores.

The NETCONF protocol stack can be divided into four layers:

– **Content**: represents data such as configuration and operational data

- **Operations**: the operations that are to be supported by NETCONF (e.g., get-config, edit-config, delete-config, discard-changes, etc.)
- **Messages**: wraps the content and operations into RPC messages
- **Transport**: defines the protocol for delivering NETCONF messages

The NETCONF protocol provides general guidelines for configuration and management of "any" underlying network device. OF-CONFIG customizes the use of NETCONF to OpenFlow switches. In simple terms, the difference between NETCONF and OF-CONFIG, is that the latter defines XML-models for OpenFlow-specific instances rather than general underlying devices. For example, OF-CONFIG defines the OpenFlow Capable Switch (OCS) which can have one or more OpenFlow Logical Switches (OFLS). i.e., an entity that manages a subset of resources in the OCS. Listing 1 presents an XML model which defines the configuration of a OFLS with the size of the Virtual Local Area Networks (VLAN) table. In the remainder of this paper we explain high-level NETCONF commands only, without delving into the OpenFlow-specific message details (i.e., OF-CONFIG).

```
<capable-switch xmlns="urn:onf:of111:config:yang"
xmlns:ndm="urn:opennetworking.org:yang:ndm"
xmlns:l2l3="urn:opennetworking.org:yang:ndm:l2l3">
  <logical-switches>
    <switch>
      <id>LogicalSwitch5</id>
        <resources>
          <ndm:ndm-implementation>
            <l2l3:l2l3>
              <vlan-table-size>128<vlan-table-size>
            </l2l3:l2l3>
          </ndm:ndm-implementation>
        </resources>
    </switch>
  </logical-switches>
<id>capable-switch-0</id>
```

Listing 1. XML-Model for configuring logical switches

As described in Section 4, the HAL architecture comprises two layers, one of which is hardware-specific (i.e., HSL) while the other is hardware-agnostic (i.e., CHPL). All management and control specific modules (e.g., OpenFlow-endpoint and NETCONF server) should reside in the CHPL. Furthermore, the network management block incorporates the NETCONF client. Furthermore, NETCONF can be abstracted for networks administrators by providing a customized user interface for managing the underlying devices. The integration of the NETCONF server / client is shown in Figure 4. The Figure illustrates that the ALIEN management plane includes also a virtualization gateway (VGW) as discussed in [6].

An example of the commands that can be provided to the administrators as an extra layer of abstraction is shown in Listing 2.

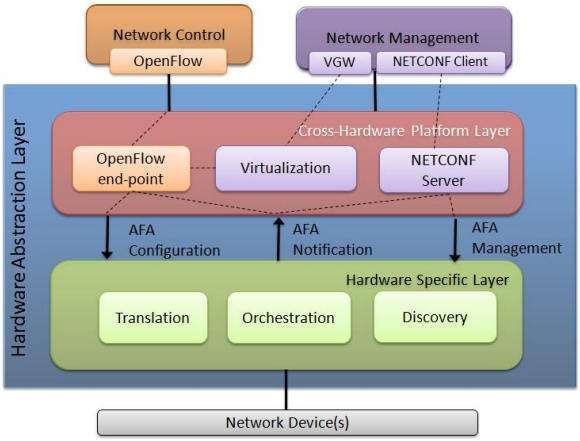

Fig. 4. NETCONF Implementation and HAL Integration

```
1 user@network-management-pc:~$ list-capabilities <ofcs_id>
2 user@network-management-pc:~$ list-ports <ofcs_id>
3 user@network-management-pc:~$ disable-port <ofcs_id, port_no>
4 user@network-management-pc:~$ list-logical_datapaths <ofcs_id>
```

Listing 2. Illustrative example for configuring logical switches

6 Software-defined Configuration

We now proceed to illustrate three software-defined configuration workflows for OpenFlow networks based on a simple but representative topology. The workflows showcase the combined use of OpenFlow in the control plane with NETCONF in the management plane. We consider interactions with devices such as middleboxes, ALIEN switches, and native OpenFlow switches. All three workflows share the same network tolopogy, illustrated in Fig. 5. The topolgy maps elements in the network into four planes: management plane, control plane, forwarding plane, and applications plane as described in Section 3.

The topology in Fig. 5 is composed of the following entities, *OF Switches* these are OpenFlow-enabled switches, some of which are equipped with sFlow [28] agents for collecting monitoring information, *ALIEN Devices* which can be controlled via OpenFlow through HAL., *Software-configured Middleboxes* with configurable functions i.e., can act as firewalls, Intrusion Detection Systems (IDS) and so on, based on their real-time configuration, *OF Controllers* to control entities for our OpenFlow-based networks, *sFlow Collectors* for collecting the monitoring information sent by the sFlow agents running in the OF switches for further analysis, *NETCONF Clients* which are devices that run the NETCONF

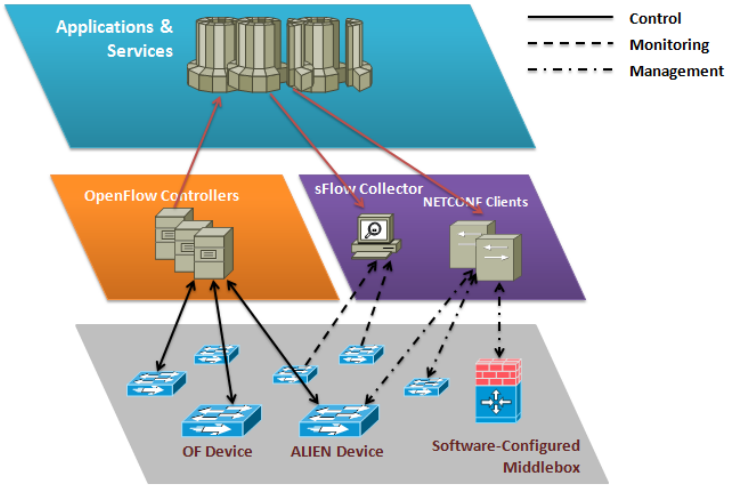

Fig. 5. Software Defined Configuration Topology

client, *Application & Services* represented as the logical entities that make use of the underlying management and control planes as per [8].

First consider the case where an network application employs NETCONF to configure the sFlow agents monitoring parameters such as sampling rate, sFlow collector IP, etc. This workflow is shown in Fig. 6. First, the application specifies the configuration parameters and sends it to the NETCONF client. In turn, the NETCONF client will form an *edit-config* message with the specified parameters and will send it to the NETCONF server running on the target OF switch(es). Once the NETCONF server receives the message, it will update the switch configuration and will send a reply back to the NETCONF client regarding the success or failure of the operation. Subsequently, upon the instruction on the application, the OF controller will interact with the underlying switches using OpenFlow to control the flow of traffic.

Another scenario for software-defined configuration using NETCONF is applying security through real-time middleboxes configuration. Usually the functions embedded in middleboxes are fixed or static. In this example we examine how NETCONF can be used to configure the middleboxes to run security modules based on the traffic monitored in the network. For example, if a Denial of Service (DoS) attack is suspected, then NETCONF client informs the middlebox to run the DoS detection module. For simplicity, the steps related to monitoring the network for suspicious behavior are omitted in this workflow (more details about these steps can be found in [29]). In the workflow shown in Fig. 7, the application is already aware of a DoS attack likelihood (with the help of network monitoring). Accordingly, the NETCONF client is instructed to configure the middleboxes to run the DoS detection modules. Consequently, the NETCONF client will form an *edit-config* message with the specified parameters and send

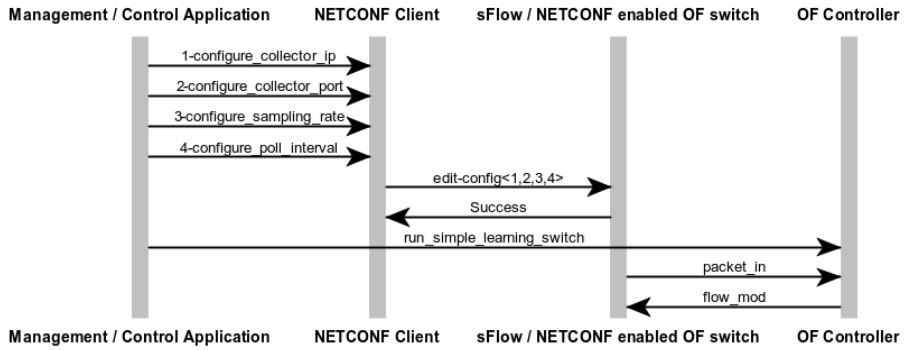

Fig. 6. Monitoring configuration workflow

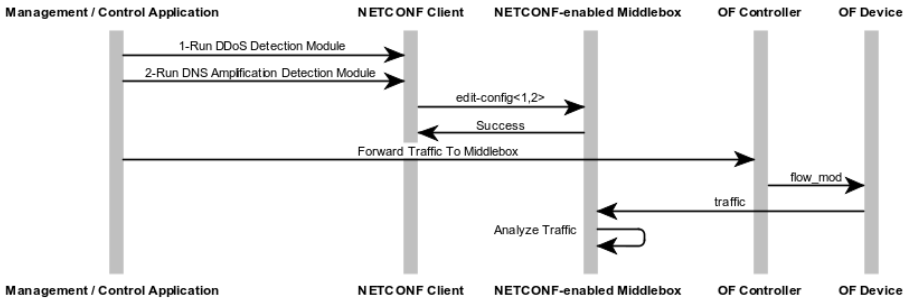

Fig. 7. Software-configured middlebox workflow

it to the NETCONF server running on the middlebox. Once the NETCONF server receives the message, it will update the device configuration, switch on the corresponding security modules, and send a reply back to the client to inform it about the success or failure of the operation. In addition to configuring the middlebox using NETCONF, the network switches are instructed by the OpenFlow controller to redirect suspicious traffic to the middleboxes for further inspection, thus utilizing the combined functionalities of network configuration and control. The middlebox functions can be provided through a physical device or using a Network Function Virtualization (NFV) approach where functions are provided as a service. Furthermore, NETCONF could be used to configure and attach virtual function and services together to provide a fully fledged network function through the assembly of smaller sub-functions.

Finally, the NETCONF implementation in HAL (§4) is used to configure the underlying ALIEN device. For example, it might be desirable to switch off a faulty interface on an ALIEN switch; alternatively, the network admin might want to list all the available interfaces on the ALIEN switch. The workflow for this scenario is shown in Fig. 8. First, the NETCONF client is instructed by the application to list all the available switch ports. Consequently, an *edit-config* message is formed and sent to the NETCONF server embedded in the HAL ar-

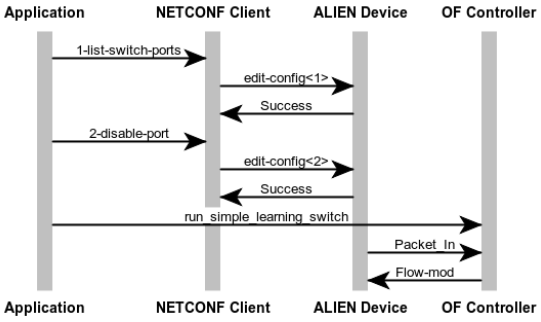

Fig. 8. Configuration of HAL-enhanced Devices Workflow

chitecture of the ALIEN switch. In turn, the NETCONF server will apply the command and send the reply back to the client. Furthermore, the NETCONF client will follow the same steps when instructed to disable a switch port. In addition to configuring the underlying ALIEN devices using NETCONF, OpenFlow will be used to control the forwarding behavior for the ALIEN devices by installing the appropriate flow rules according to the logic running on-top of the controller. In addition to using OpenFlow for network control, NETCONF automates configuration management functions and thus making network management tasks much simpler.

7 Conclusion and Future Work

OpenFlow networks are expected to proliferate in the coming years. Although up to now SDN R&D has placed more emphasis on control and data plane development, we expect that operations and management (OAM) aspects deserve more attention. In this paper we focused on network configuration in OpenFlow networks, and in particular on how software-defined configuration can enhance SDN deployments. After reviewing OF-CONFIG and NETCONF in an OpenFlow network context we discussed the evolution of the programmable networks paradigm and drew parallels between the configurable middleboxes line of work and OpenFlow-based SDN. Given the original focus of OpenFlow switches on campus and data center environments, the FP7 ALIEN projects aims to extend devices which do not natively support OpenFlow and introduce them to SDN experimental facilities such as OFELIA in Europe. For this type of devices, configuration is necessary and as we have seen NETCONF can serve as the basis for further development in this domain. In this respect, we overviewed the ALIEN Hardware Abstraction Layer (HAL) and mapped it to recent work in the IRTF SDN RG. This paper also introduced a range of workflows which illustrate how software-defined configuration can work in practice, enabling network applications and services to dynamically configure and control a virtual infrastructure which includes native OpenFlow switches, ALIEN devices and configurable mid-

dleboxes. Namely, we discussed software-defined configuration of sFlow monitoring agents in OF switches, ALIEN device configuration and security-related middlebox configuration. We are currently working on the NETCONF implementation in the ALIEN HAL and aim to release the code during summer 2014. This implementation will be part of the ALIEN demos and will be introduced in the OFELIA experimental facility once it is mature for production. Until then, we plan to have several experiments performed, illustrating the feasibility and potential of our approach while measuring its performance on an OpenFlow testbed. Furthermore, we consider mapping the architecture defined in this paper to a virtual environment , where network functions are defined as a server and where NETCONF could be used to configure the necessary parameters for these virtual network functions.

Acknowledgment

This work was conducted within the framework of the FP7 ALIEN project, which is partially funded by the Commission of the European Union.

References

1. N. Feamster, J. Rexford, and E. Zegura. The Road to SDN. *Queue*, 11(12):20:20–20:40, December 2013.
2. W. John, K. Pentikousis, et al. Research directions in network service chaining. In *Future Networks and Services, 2013 IEEE SDN for*, pages 1–7, Nov 2013.
3. N. McKeown, T. Anderson, et al. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.
4. ONF. OF-CONFIG 1.2. OpenFlow Management and Configuration Protocol version 1.2, 2014.
5. R. Enns, M. Bjorklund, and J. Schoenwaelder. NETCONF configuration protocol. *IEEE Network*, 2011.
6. D. Parniewicz, R. Doriguzzi Corin, et al. Design and implementation of an openflow hardware abstraction layer. In *SIGCOMM DCC 2014*, pages 1–6, 2014.
7. L. Ogrodowczyk, B. Belter, et al. Hardware abstraction layer for non-OpenFlow capable devices. In *TERENA Networking Conference*, pages 1–15, 2014.
8. E. Haleplidis, S. Denazis, et al. SDN layers and architecture terminology. *Internet Draft: draft-haleplidis-sdnrg-layer-terminology (work in progress)*, 2014.
9. J. Schönwälder, M. Björklund, and P. Shafer. Network configuration management using NETCONF and YANG. *IEEE Communications Magazine*, 48(9):166–173, 2010.
10. B. Hedstrom, A. Watwe, and S. Sakthidharan. Protocol Efficiencies of NETCONF versus SNMP for Configuration Management Functions. page 13, 2011.
11. J. Yu and I. Al Ajarmeh. An empirical study of the NETCONF protocol. In *Networking and Services (ICNS), 2010 Sixth International Conference on*, pages 253–258. IEEE, 2010.
12. Sh. Bhushan, H. M. Tran, and J. Schönwälder. NCClient: A python library for NETCONF client applications. In *IP Operations and Management*, pages 143–154. Springer, 2009.

13. R. Krejci. Building NETCONF-enabled network management systems with libnetconf. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 756–759. IEEE, 2013.
14. H. M. Tran, I. Tumar, and J. Schönwälder. NETCONF interoperability testing. In *Scalability of Networks and Services*, pages 83–94. Springer, 2009.
15. G. Munz, A. Antony, et al. Using NETCONF for configuring monitoring probes. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 1–4. IEEE, 2006.
16. H. Xu, C. Wang, et al. NETCONF-based integrated management for internet of things using RESTful web services. *International Journal of Future Generation Communication & Networking*, 5(3), 2012.
17. Zhu W., Liu N., et al. Design of the next generation military network management system based on NETCONF. In *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on*, pages 1216–1219, April 2008.
18. H. Lu, N. Arora, et al. Hybnet: network manager for a hybrid network infrastructure. In *Proceedings of the Industrial Track of the 13th ACM/IFIP/USENIX International Middleware Conference*, page 6. ACM, 2013.
19. B. Sonkoly, A. Gulyás, et al. Openflow virtualization framework with advanced capabilities. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 18–23. IEEE, 2012.
20. J. M. Smith and S. M. Nettles. Active networking: one view of the past, present, and future. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(1):4–18, 2004.
21. D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse. Ants: A toolkit for building and dynamically deploying network protocols. In *Open Architectures and Network Programming, 1998 IEEE*, pages 117–129. IEEE, 1998.
22. B. Samrat, Kenneth L. Calvert, and Ellen W. Zegura. An Architecture for Active Networking. In *IEEE Communications Magazine*, pages 72–78, 1997.
23. J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat. xOMB: Extensible Open Middleboxes with Commodity Servers. In *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '12, pages 49–60, New York, NY, USA, 2012. ACM.
24. Z. Ayyub Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 27–38, New York, NY, USA, 2013. ACM.
25. N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe. The Case for Separating Routing from Routers. In *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, FDNA '04, pages 5–12, New York, NY, USA, 2004. ACM.
26. L. Yang, R. Dantu, et al. Forwarding and control element separation (ForCES) framework, 2004.
27. N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.
28. M. Wang, B. Li, and Z. Li. sflow: towards resource-efficient and agile service federation in service overlay networks. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 628–635, 2004.
29. A. Zaalouk, R. Khondoker, et al. OrchSec: An Orchestrator-Based Architecture For Enhancing Network-Security Using Network Monitoring And SDN Control Functions. In *IEEE SDNMO, Krakow, Poland*, pages 1–8, 2014.