Overview of South-Bound Interfaces for Software-Defined Optical Networks

Miquel Garrich Alabarce, *Member, IEEE,* Anderson Bravalheri*

Universidad Politécnica de Cartagena, Cuartel de Antiguones, Plaza del Hospital 1, 30202 Cartagena, Spain

** High Performance Networks Group, Faculty of Engineering, University of Bristol, BS8 1TH, UK*

Tel: (34) 968 338 946, e-mail: miquel.garrich@upct.es

ABSTRACT

In SDN-enabled networks, the control plane and data plane interaction relies on open SouthBound Interfaces (SBIs) so that the SDN controller exercises direct control over the data plane elements. In this paper, we review current initiatives of SBI to control optical components which include ad-hoc extensions of OpenFlow and YANG modelling proposals combined with the NETCONF / RESTCONF protocols. Then we overview different tools and frameworks available for quick prototyping and deployment of software services that are compliant with such interfaces. Finally, we discuss the advantages and drawbacks of the reviewed initiatives considered key enablers for standardized end-to-end network programmability.

Keywords: Software defined networking, optical networks, programmatic interfaces.

1. INTRODUCTION

Software Defined Networking (SDN) enables advanced network programmability because it decouples the forwarding data plane actions from the control plane decisions [1], thus potentially overcoming the limitations of current network infrastructure and enabling many new functionalities [2]. In SDN, SouthBound Interfaces (SBI) play a key role because are used from the SDN controller to directly control data plane elements. OpenFlow [3] is a remarkable example of SBI, which used to update packet handling rules in the flow table which governs the behavior of the switches. Undeniably, OpenFlow has become the industrial standard in electronic packet networks.

In optical networks, SDN is attracting notable interest due to its applicability to control and manage the specific optical (photonic) transmission and switching characteristics of the optical domain [4]. Software-Defined Optical Networks (SDONs) aim to exploit the flexibility of SDN control to support networking applications with an underlying optical network infrastructure [5]. A comprehensive survey of techniques applying SDN to optical networks can be found in [6], which includes virtualization and orchestration aspects for SDONs. Although [6] remarks the importance of simplified management strategies, it provides a limited view of the models that would be required for a centralized optical network management. Indeed, common abstractions and interfaces are essential in SDONs to enable open and vendor agnostic management of optical equipment. This aspect is deeply covered in [7], which surveys and compares the most important models and proposes an intent interface to create virtual topologies based on the existing models. In this context, a clear driver that motivates the existence of several proposals are different views of network operators on their requirements, operational needs and particular use cases may lead to a variety of optical network models, covering various aspects ranging from device-oriented up to generic descriptions of optical network elements. Consequently, initiatives regarding white-box and openness of optical networks are attracting the interest of telecom operators aiming to define an open unified vendor-independent network [8].

In this paper, we review current initiatives of SBI to control optical components which include ad-hoc extensions of OpenFlow and YANG modelling proposals combined with NETCONF / RESTCONF protocols. Then we overview different tools and frameworks available for quick prototyping and deployment of software services that are compliant with such interfaces. Finally, we discuss the advantages and drawbacks of the reviewed initiatives.

2. SOUTH-BOUND INTERFACES AND PROTOCOLS

2.1 OpenFlow

OpenFlow (OF) was proposed as a standard protocol to enable the separation of the data and control plane in packet networks. Originally based on convergent principles of operation shared among several electronic switch vendors [3], the OF protocol is intended to be simple and assumes that network switches can handle received packets by following rules contained in a table. Each rule describes a condition (e.g. input port, header values) that triggers specific actions (e.g. forward the packet to an output port, modify/discard the packet) when matched. Therefore, the network programmability is achieved by changing (and chaining) different flow tables.

The same model cannot be directly applied to a circuit switching equipment and in particular to ROADMs, due to the lack of packet structure. However, information flows inside optical circuits can be matched according to physical parameters (e.g. central frequency of the carrier wave, bandwidth), in the same way packets are matched according to header values. Optical channels can be “forwarded” between two different fibers (similarly to packets steered between two switch ports), or even “modified” (if you have a colorless ROADM). As a result, OF extensions were proposed to enable the SDN control of optical networks [4], which led to an architectural proposal for a unified control and management of circuit-based (optical) and packet-based electronic networks [5].

The architecture proposed by Channegowda et al, shown in Fig. 1 (adapted from [5]) is based on an extension of the OF protocol to enable an abstraction that unifies and generalizes the flow concept to both the optical domain (including fixed and flexible grids) and the packet-switched network. The extension of the OF protocol is implemented in the flow switching rules by applying specific technological fields in the flow tables of the "generic" OF switch. Moreover, the proposal in [5] also includes inter-domain rules implemented in the flow tables so that technological constraints can be applied for specific traffic that traverses different technologies.

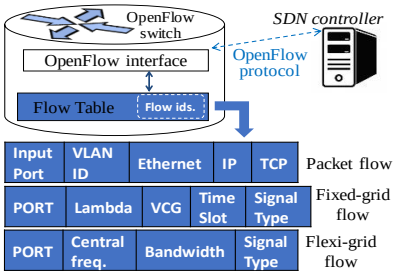

Figure 1. Flow definitions for different technology domains. Adapted from [5].

2.2 NETCONF/RESTCONF protocols and YANG modelling

The ***NETCONF*** protocol was created by IETF as an effort to standardize the access and configuration of network equipment, and to address the weakness of the Simple Network Management Protocol (SNMP) [9][10]. NETCONF is a remote procedure call (RPC) protocol that uses extensible markup language (XML) for data serialization [11], allowing network operators to change static and runtime configurations in a diverse range of devices. This protocol stablishes main operations for configuration and state management aside from allowing extensions by means of defined capabilities [12]. The XML messages exchanged during NETCONF sessions are defined on a per-device basis, however formalized using the ***YANG*** modelling language [13]. This language allows mapping configuration parameters and status information for each device in a tree-data structure.

With the pervasiveness of web services and the increasing advantages in exposing Application Programming Interfaces (APIs) through HTTP connections, a subset of the NETCONF semantics was transferred to a RESTful [14] context, originating the ***RESTCONF*** protocol. In RESTCONF, important pieces of the hierarchical data-structure defined in the YANG modules correspond to REST "resources" that can be accessed/manipulated using `GET`, `POST`, `PUT`, `PATCH` and `DELETE` HTTP methods. In turn, these methods basically provide that same functionality that the main NETCONF configuration and state management operations [13].

OpenROADM [15] is a Multi-Source Agreement (MSA) that defines optical interoperability specifications and YANG models which currently counts on 15 members including world leading vendors and telecom operators. In summary, OpenROADM targets multi-vendor, interchangeable and inter-workable optical functions with standard APIs written in YANG modelling language that can be accessed through an SDN controller using NETCONF. YANG models include pluggable optics with interoperable line-side given that client sides are currently well covered in other standards. Flexible interoperable ROADMs include YANG models to capture colorless and directionless (CD) and contentionless (CDC) characteristics.

OpenConfig [16] is a project that provides a common data model for management interfaces that network operators can use to configure and monitor any equipment regardless of vendor. OpenConfig includes a set of vendor neutral data models that address different technologies and which are derived from operational needs, use cases and requirements from operators. The main objective in OpenConfig is to standardize the management interface or APIs to network elements, regardless of data-plane function. In the optical domain, OpenConfig describes a set of five models for the optical transport systems (e.g., OSC, amplifiers, terminals and ROADMs).

OpenDevice [17] is an adaptation of the proposals done in OpenROADM and OpenConfig with the addition of a declarative part with specific functionalities. In particular, Infinera and Lumentum recently demonstrated [17] an automated service management and automated optical power control based on YANG modelling in which optical signals were expressed between the two vendors' ROADMs and terminals.

3. SOFTWARE TOOLS AND FRAMEWORKS FOR CLIENT AND SERVER/AGENT CREATION

Software components in the SDN ecosystem can undertake two distinct roles. The client usually corresponds to the SDN controller or a program used by human operators devoted to perform configurations in the network. The server commonly corresponds to a device, another program, or even a simulation, that must react to the requests sent according to the specific protocol, performing (or pretending to perform) the network configuration itself. The server is also referred as "agent" or "daemon". In this section we explore the available solutions for SBI in SDONs.

3.1 OpenFlow

Since the OpenFlow protocol is historically related to the origin of SDN, the most common way of configuring OF-enabled switches is via SDN controllers. As a result, libraries implementing an OpenFlow client are scarce and usually outdated. However, a few implementations focusing on packet parsing and generation can be used to implement OpenFlow clients and servers, thus, the implementation of optical extensions is not straightforward.

Loxigen is a tool that generates OpenFlow bindings for C, Python and Java, highly updated and maintained by the Floodlight SDN controller team [18]. ***Pio*** is a Ruby library maintained by the Trema OpenFlow Framework [19], that supports version 1.3 of the protocol. ***Ofpmsg-js*** is a JavaScript library maintained by the Flowgrammable group [20], that supports versions up to 1.5. ***Python-openflow*** is an updated Python library, maintained by the members of the Kytos opensource SDN project [21].

OpenVSwitch (OVS) [22], backed by the Linux Foundation, is a virtual switch implementation, that can be controlled by several standard protocols and works as reference implementation due to its broad adoption in virtualized environments. OVS is very well documented, however, since the project focus on commodity hardware implementation, adapting it to custom and prototypical solutions requires deep knowledge/familiarity of the source code and the Linux kernel.

Indigo project [23], maintained by the Floodlight community, is an alternative to OVS, and implements an OpenFlow switch agent focusing in equipment manufacturers. In order to fulfil its vision, the project is based in a modular design that provides all the common functionality, but relies on a separated developer-specific hardware abstraction layer, which may turn the process of implementing prototypes or simulations easier. Unfortunately, Indigo documentation is not as available as OVS.

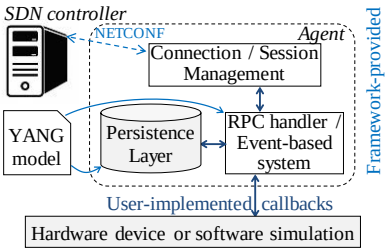

Figure 2: ConfD operating principles.

3.2 NETCONF

NETCONF have been used to directly configure the equipment since its creation, and just more recently it was adopted as the second most promoted standard for SBI in SDN controllers. Therefore a plethora of different client implementations is available in popular programming languages, e.g. [24], [25], [26] and [27], however most of them still require manually composing the XML content for the messages. Conversely, a very limited number of tools is available for implementing NETCONF agents.

ConfD is a proprietary framework created by Tail-f (owned by Cisco), distributed under two different categories of licenses. The basic edition is offered without cost but supports a very limited subset of features, for instance, just the C and Erlang APIs are available, and the documentation is sparse. On the other hand, the premium edition is commercialised with the complete set of features, including Python API and customer support [28]. The framework is able to compile the YANG models provided by the developer, automatically generating the infrastructure necessary to run the NETCONF server. Additionally, it creates a persistence layer which stores all the information modelled as a tree data structure. Associated to the persistence layer, an event-based system creates a communication bus that enables developers to react to configuration requests (executing custom operations in the underlying equipment or simulation), and to operational status changes (publishing them to the storage layer). This architecture is show in Fig. 2. Due to the complexity of the solution, the choice of the programming language (optimized for runtime performance) and the lack of openness (as a closed source software), the learning curve of ConfD is steep, and creating custom agents can be cumbersome for developers inexperienced in the framework.

Netopeer2 is an opensource alternative to ConfD developed and maintained by CESNET [29]. The toolset is implemented using a modular approach and follows similar operating principles to ConfD (Fig. 2). Within Netopeer2 architecture, `libnetconf2` [30] is used to compose the NETCONF interface, implementing the protocol internals, while `sysrepo` [31] creates a database responsible for storing configuration and operational data. `libyang` [32] is widely used to compile and validate the YANG modules provided by the user to create the data schema input for the other software components. Basic documentation is provided including simple examples. Netopeer2 can provide NETCONF APIs to custom devices or simulations, while custom behaviour can be implemented dynamically in Python, Java, Go and Lua, in addition to the default C static language, through `sysrepo` callbacks (although the available language bindings still resemble the low-level C APIs).

YDK is a development kit created by Cisco to support network programmability based on YANG modules [33]. YDK generates object classes for C++ and Python (with expectations to add support for at least Ruby, Go and C#) based on the provided models, being able to convert between native data structures, XML and JSON, which is extremely useful for both NETCONF and RESTCONF server implementations. Similar to Netopeer2, YDK also makes use of `libyang` to provide run-time YANG model analysis. In spite of targeting NETCONF client scenario, YDK can also be used to create agents. Nonetheless, it only supports data mapping and (de)serialization, which means that all the code necessary to handle NETCONF connections, session management and RPC need to be created manually by the developer. An alternative approach would be integrating YDK in a ConfD or Netopeer2 deployment, however, this is a complex task that requires strong knowledge about the selected framework and multi-programming language development.

3.3 RESTCONF

Implementing a RESTCONF client is extremely easy compared with the previous protocols, thanks to the popularity of HTTP (and especially RESTful) APIs. Almost every programming language provides HTTP libraries that can be used to perform calls to the RESTCONF API. The same can also be done using popular Unix command line tools, such as curl [34]. On the other hand, complete RESTCONF server implementations are essentially restricted to commercial solutions, such as ConfD Premium. Fortunately, web frameworks are extremely popular in most of the programming languages and, therefore, can be used in conjunction with toolsets such as YDK, to create RESCONF servers.

PyangBind [35], as an alternative to YDK, is a plugin for the popular YANG parser and generator `pyang` [36], that is able to convert YANG models into entire object class hierarchies in Python. The resulting code can be then used to (un)serialize data from/into the JSON format, which, in turn, can power HTTP requests/responses.

Ygot, created in the context of OpenConfig, aims to achieve similar objectives but targeting the Go programming language [37]. Ygot, is able to validate if Go structures match a defined YANG schema and also generate bootstrapping code (types/data structure definitions and helper methods). Despite focusing on gRPC systems (and therefore protobuffers serialisation), the toolset is able to generate/parse JSON messages and thus can also be used to power classical JSON+HTTP requests/responses.

4. DISCUSSION AND SUMMARY

Although OpenFlow proposals obtained major relevance in the early stages of SDONs, current trends on YANG modelling proposals combined with the NETCONF / RESTCONF protocols are attracting the interest of the industry. For instance, OpenROADM and OpenConfig are initiatives aiming to become de-facto SBI standards. Upon those proposals, different communities propose frameworks that cover a wide range of aspects in SDONs. In this context, the availability of opensource tools to implement protocol agents during product development or research programs is fundamental. Indeed, a framework that abstracts away redundant implementations is essential to provide a common ground that allows the user to focus on the domain specific details. By doing so, the developer should be able to use high-level dynamic programming languages to iterate quickly and achieve fully functional proofs of concept which can then be refined, optimized and turned into the production-ready solution. Inappropriately, the current state of the SDON SBI development does not match these expectations.

OpenFlow agents, regardless of being conceptually simpler and easier to implement in theory, cannot be easily deployed since the available tools (and the protocol itself) were not created to handle the requirements of an optical equipment. At the same time, NETCONF and RESTCONF, that can perfectly support the optical domain specificities, imply in very complex agent implementations for which, no good enough and mature prototyping toolkit is available. This lack of resources jeopardizes the standardization movement in the SDON community. In the absence of programming frameworks, divergent implementations start to appear, as already demonstrated by the early experiments of the OpenConfig initiative in using gRPC as replacement for NETCONF. The SDON community needs to address these issues, so the efforts in research and development of new technologies and techniques can be delivered quicker and without requiring needless effort.

ACKNOWLEDGEMENTS: The research leading to these results has received funding from the Eur. Comm. for the H2020-ICT-2016-2 METRO-HAUL project (G.A. 761727) and the H2020-MSCA-IF-2016 INSPIRING-SNI project (G.A. 750611).

REFERENCES

[1] D. Kreutz, et al., “Software-Defined Networking: A Comprehensive Survey,” *Proceedings of the IEEE*, 103-1, Jan. 2015.
[2] S. Das, G. Parulkar, and N. McKeown, “Why OpenFlow/SDN can succeed where GMPLS failed,” *ECOC*, 2012.
[3] N. McKeown, et al., “OpenFlow: Enabling innovation in campus networks,” *ACM SIGCOMM*, 2008.
[4] S. Gringeri, et al., “Extending software defined network principles to include optical transport,” *Com. Mag.*,51-3, 2013.
[5] M. Channegowda, R. Nejabati, and D. Simeonidou, “Software-defined optical networks technology and infrastructure: Enabling software-defined optical network operations [invited],” *JOCN*, v. 5, n. 10, pp. A274-A282, 2013.
[6] A. S. Thyagaturu, et al., “Software Defined Optical Networks (SDONs): A Comprehensive Survey,” *IEEE Comm. Surveys and Tutorials*, vol. 18, no. 4, pp. 2738–2786, 2016.
[7] T. Szyrkowiec, et al., “Optical Network Models and Their Application to Software-Defined Network Management,” *International Journal of Optics*, 2017.
[8] E. Riccardi, et al., “An Operator's view on introduction of White Boxes in Optical Networks,” *JLT*, 2018.
[9] J. Schoenwaelder, “Overview of the 2002 IAB Network Management Workshop,” IETF RFC 3535, 2003.
[10] M. Bjorklund, “YANG—A data modeling language for the network config. proto. (NETCONF),” IETF RFC 6020, 2010.
[11] XML-RPC official web site, http://xmlrpc.scripting.com/
[12] R. Enns, et al., “Network configuration protocol (NETCONF),” IETF RFC 6241, June 2011.
[13] A. Bierman, M. Bjorklund and K. Watsen, “RESTCONF Protocol,” IETF RFC 8040, January 2017.
[14] F. Belqasmi et al., "RESTful web services for service provisioning in next-generation networks: a survey", *IEEE Commun. Mag.*, vol. 49, no. 12, pp. 66-73, December 2011.
[15] OpenROADM web site http://www.openroadm.org/
[16] OpenConfig web site http://www.openconfig.net/
[17] O. F. Yilmaz, et al., “Automated Management and Control of a Multi-Vendor Disaggregated Network at the L0 Layer,” *OFC*, Tu3D.9, San Diego, USA, 2018.
[18] https://github.com/floodlight/loxigen
[19] Parser and gen. Ruby https://github.com/trema/pio
[20] https://github.com/flowgrammable/ofpmsg-js
[21] Python OF lib, https://docs.kytos.io/python-openflow/
[22] OpenVSwitch project, https://www.openvswitch.org/
[23] Indigo project, http://www.projectfloodlight.org/indigo/
[24] Python library https://github.com/ncclient/ncclientt
[25] Ruby library https://github.com/Juniper/net-netconft
[26] Perl library https://github.com/Juniper/netconf-perl
[27] Go library https://github.com/Juniper/go-netconf
[28] Tail-f ConfD, http://www.tail-f.com/confd-basic/
[29] Netopeer2 https://github.com/CESNET/Netopeer2
[30] https://github.com/CESNET/libnetconf2
[31] YANG-based datastore Linux, http://www.sysrepo.org/
[32] https://github.com/CESNET/libyang
[33] Yang Dev. Kit, https://developer.cisco.com/site/ydk/
[34] Curl, com. line tool and library, https://curl.haxx.se/
[35] PyangBind for pyang, http://pynms.io/pyangbind/
[36] https://github.com/mbj4668/pyang/
[37] YANG Go Tools, https://github.com/openconfig/ygot/