Enhancing Whitebox Optical SDN Control Using OpenROADM Models

Enhancing Whitebox Optical SDN Control Using OpenROADM Models

1st Riccardo Schips
Politecnico di Torino
Turin, Italy
s303321@studenti.polito.it

2nd Roberto Morro
Telecom Italia
Turin, Italy
roberto.morro@telecomitalia.it

3rd Renato Ambrosone
Politecnico di Torino
Turin, Italy
renato.ambrosone@polito.it

4th Stefano Straullu
Links foundation
Turin, Italy
stefano.straullu@linksfoundation.com

5th Francesco Aquilino
Links foundation
Turin, Italy
francesco.aquilino@linksfoundation.com

6th Antonino Nespola
Links foundation
Turin, Italy
antonino.nespola@linksfoundation.com

7th Anna Chiadò Piat
Telecom Italia
Turin, Italy
anna.chiadopiat@telecomitalia.it

8th Vittorio Curri
Politecnico di Torino
Turin, Italy
vittorio.curri@polito.it

***Abstract*—In this paper, the authors exploit the OpenROADM standard model for the management and control of commercial ROADMs via an open optical SDN controller. Since each ROADM model has its own operating system, characterized by specific proprietary languages and interfaces, an additional middleware software layer must be introduced between the controller and the ROADM devices. This middleware aims to expose commercial devices through the OpenROADM model. Furthermore, the controller can send requests via the OpenROADM standard to the middleware.**

I. Introduction

The growth of data traffic on the internet necessitates an enhancement of network architecture. This can be achieved by replacing static devices with software-based devices that run their own operating systems. These entities form the foundation of an open optical Software Defined Network (SDN)[13]. Within an SDN, software-based devices such as transceivers, which inject signals into the network, and Reconfigurable Optical Add-Drop Multiplexers (ROADMs), which manage signal forwarding, play crucial roles.

Focusing on the ROADM entity, the transition from a static device to a dynamic one allows for optimized signal handling. Utilizing a ROADM enables the direct management and forwarding of incoming optical signals while maintaining them in the optical domain. In contrast, a static add-drop multiplexer cannot process an optical signal directly; it requires the signal to be converted to the electrical domain for processing and then reconverted to the optical domain before re-transmission. These conversion steps introduce significant latency, thus highlighting the necessity of replacing static devices with ROADMs. ROADMs are more efficient in signal processing, as they reduce latency in forwarding operations. This reduction in latency is critical to preventing network congestion, especially as data traffic increases.

Additionally, the expansion of network devices is crucial to accommodate growing traffic demands. Each transceiver can only transmit data at a single frequency; thus, an increase in traffic necessitates the deployment of more transceivers.

Typically, networks are oriented towards single-vendor solutions, but transitioning to multi-vendor networks offers significant advantages. Multi-vendor networks enable cost control through diverse market options, allowing for the selection of various device models. Different vendors produce devices and software with unique proprietary languages, necessitating standard models to ensure interoperability within a multi-vendor network.

Another reason to have standard models is to enhance the ability of the optical digital twin (DT) [5] to facilitate innovation and collaboration across industries. Standard models enable the consistent representation and interpretation of data, which is crucial for integrating digital twins into complex, multi-stakeholder ecosystems. This uniformity allows different organizations to collaborate more effectively, sharing and utilizing data from digital twins without the need for extensive customization or translation. It also accelerates the development and deployment of digital twin solutions by providing a common framework that developers can build upon, reducing the time and cost associated with creating bespoke models. Additionally, standard models support the scalability of digital

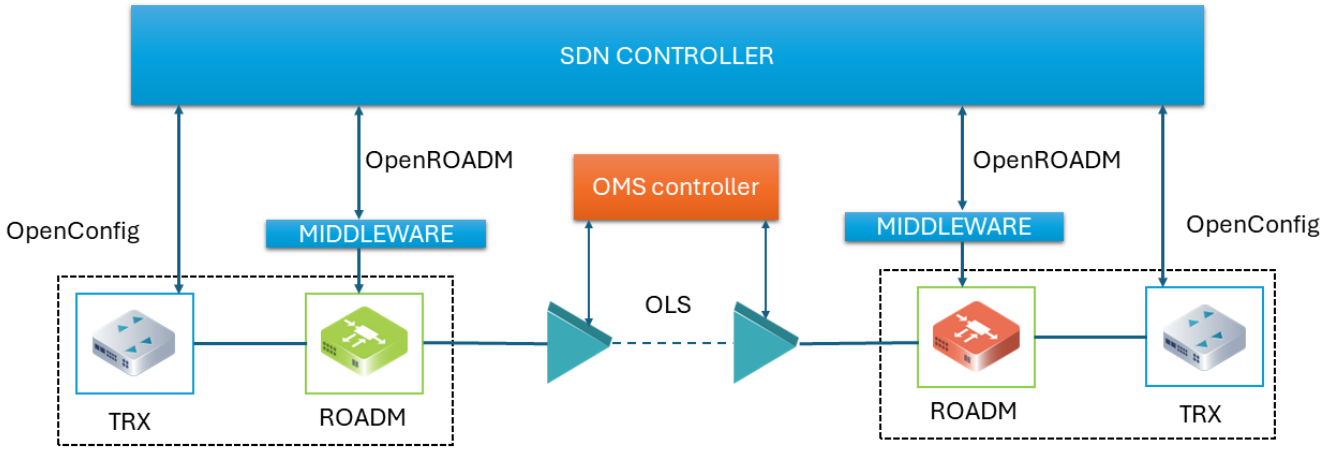

Fig. 1. Multi-vendor scenario in which each ROADM exposes a middleware for the translation of the OpenROADM commands

twin implementations, ensuring that as systems grow and evolve, the digital twins can adapt without losing coherence or functionality. By promoting compatibility and interoperability, standards not only streamline current operations but also pave the way for future advancements, driving the broader adoption and success of digital twin technology in various sectors.

In this context, the interoperability between the controller and a specific ROADM model was tested to demonstrate the feasibility of multi-vendor communication. This test illustrated how communication occurs between the controller, utilizing OpenROADM [8], and a ROADM device with its proprietary language.

II. Optical Network Architecture

This paper follows the partially disaggregated architecture proposed in [4]. An optical SDN architecture is characterized by the disaggregation of the functions: the control plane is managed by the controller, whose aim is to communicate with the entities of the network, providing them the functions that must be executed; the data plane is managed directly by the devices, and it is related to the data transfer inside the network [12]. In this paper, the software architecture was built by embracing the disaggregated line approach: the ROADMs and transponders are controlled by an SDN controller while the Optical Multiplexer Section is controlled by software that handles the management of the amplifiers OMS-Controller [1]. This software architecture respects the growing need for modern optical networks to move toward a multi-vendor approach. In this context, the network controller needs to communicate with different nodes in the network even if they are from different vendors. With this in mind, we have developed a software middleware that aims to abstract the vendor singularities, exposing the device to the controller as if it were a fully compatible device with the chosen standard, OpenROADM. This standard will be described in deep in the following.

A. The OpenROADM device model

OpenROADM [8] is a Multi-Source Agreement initiative, active since 2015 and comprising several network operators and optical system and component vendors, aiming at defining interoperability specifications for ROADMs, transponders and pluggable optics. From the control plane perspective, OpenROADM defines data models for device, network, and service modelling, targeting the fully disaggregated network model. The device model covers detailed configuration information, alarm, and performance monitoring and defines several objects to abstract the implementation of a ROADM device. The most notable are: *circuit-packs* that represent a physical piece of equipment which contains a group of hardware functional blocks; the *ports* container defines the ports associated with a circuit pack; the *internal-links* object reflects the connectivity within each circuit pack; the *physical-links* container reflects the connectivity between ports across different circuit packs; *interfaces* that defines supported interface types and the association with Port objects; *roadm-connections* lists the cross-connections between couples of interfaces. The OpenROADM consortium publishes also whitepapers explaining how to exploit the YANG models and how common operations should be encoded. In particular, the OpenROADM device whitepaper [10] defines how the above mentioned objects should be used to model network devices (ROADMs, transponders, amplifiers and pluggables) and the operations that an external controller can perform on them. The procedure to create a cross-connection between two ROADM ports it's relevant in the scope of this paper. Considering an express-connection, i.e. a cross-connection between two degrees, the procedure requires the creation of Media Channel (MC) interfaces on the involved ports, specifying the lower and upper frequencies of the spectral slot, then the creation of Network Media Channel (NMC) interfaces on the previously created MC interfaces and, at last, the creation of the *roadm-connections* between the two NMCs.

B. The OpenROADM agent

A software agent has been developed to implement the procedures defined by the OpenROADM consortium. It exposes a NETCONF interface implemented using Netopeer [9] and the transAPI tool [11], both based on libnetconf. By means of the transAPI is possible to invoke a specific call-back function whenever an *edit-config* Remote Procedure Call (RPC) operation performs changes on a branch of the configuration. The agent has therefore call-back functions that manage the controller requests for the creation of the MC and NMC interfaces and the roadm-connections. To decouple the OpenROADM model processing from the action required by the underlying hardware, the agent can load specific drivers at runtime. Drivers are coded as Linux dynamic libraries associated to a *circuit-pack-type* and loaded by the main module when an *edit-config* RPC creates a new *circuit-pack* of that type. The driver implements a few functions to perform actions on the corresponding circuit-pack. The most important functions are: *i) init* called during agent startup, to set-up the communication session and perform initial circuit-pack setup; *ii) close*: called at agent closing to free all the allocated resources; *iii) make_connection*: for the creation of the cross-connection (spectral window) between circuit-pack ports; *iv) delete_connection*: for cross-connection removal. Moreover, the agent encompasses also the datastore describing the implemented device configuration. To improve agent flexibility and portability, it is implemented as a docker container encompassing simple drivers for emulated circuit-packs and a complete datastore for a 3 degree ROADM. A different device can be supported sharing a directory from the host's filesystem into the container folders containing the drivers and the datastore by means of the mount docker option.

III. Setup

Fig. 2 illustrates the configured laboratory setup, featuring three Adtran ROADMs, each comprising three degrees. Each degree is composed by a twin of multiplexer (MUX) and demultiplexer (DMX): each one is composed by one *network port* used to transmit and receive the signal from the OMS, and 12 *client ports* used both to interconnect the different degree within the ROADM and to connect add/drop modules. Each port is capable of both transmitting and receiving: from the hardware prospective, the presence of two connectors labeled TX and RX allows the handling of the signal's direction.

Connections between degrees are established using optical fibers, which must be correctly inserted into the appropriate client ports according to the transmission direction, i.e., from a transmitting client port to a receiving client port, which must belong to different degrees. The connections controllable by the operating system are those between a network port and a client port. In this context, the SDN controller communicates with optical switches and transponders using standardized protocols such as NETCONF, allowing for real-time adjustments to routing paths, bandwidth allocation, and network topology in response to varying traffic demands and network conditions. Communication between the controller

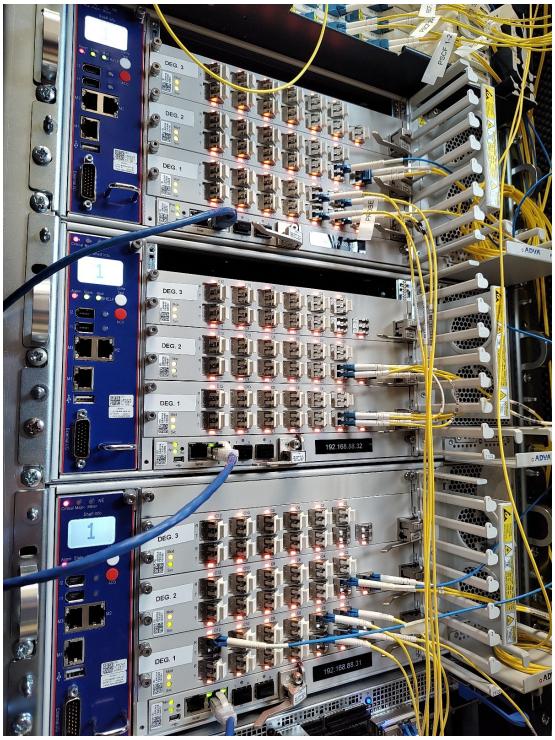

Fig. 2. Laboratory three node network setup picture. This photo focuses on the switching part of optical laboratory setup, each node is connected to optical line system which is not shown in this picture.

and the ROADMs is facilitated by the OpenROADM standard. As described in Section II-A, this standard was developed to create a general model capable of communicating with all ROADMs, regardless of the vendor. In this setup the communication is not direct, as the ROADMs are not supporting the OpenROADM models; hence, a middleware is introduced to translate commands from the OpenROADM standard to the proprietary language. The selected controller is the Open Network Operating System (ONOS) [7], an open-source SDN controller that supports the OpenROADM standard.

There are various methods to interact with the ROADM's operating system: Secure Shell Protocol (SSH), RESTCONF [3], or NETCONF [6] protocols. In this case, the RESTCONF protocol was chosen. RESTCONF commands are converted into software written in C language, which runs inside the OpenROADM agent. The main functions developed, as mentioned in Section II-B, are now described from the ROADM perspective, detailing the necessary commands the ROADM's operating system must execute.

The first function, *init*, is used for the login operation. Commands sent to the ROADM are written according to the RESTCONF protocol, which requires specific headers in the request, including a token necessary for authorizing the execution of all commands. This function retrieves a temporary token to be included in the request headers, allowing the execution of subsequent functions. The *init* and *close* functions are also used for purposes outlined in Section II-B.

Two additional functions, *make_connection* and

delete_connection, manage connections within the ROADM. The former creates a new connection between two ports using the respective MUX or DMX, while the latter deletes an existing connection.

To connect two ports, several steps must be executed: *i)* the creation of a Spectrum Slot Group (SPSLG); *ii)* the creation of an Optical Tributary Signal Assembly (OTSiA); *iii)* the connection of the two ports via the media port; and *iv)* changing the OTSiA status. All steps, except the third, must be repeated for both ports. Detailed explanations of these steps are provided below.

The first step involves creating a spectrum slot group, which consists of one or more spectrum slots. For simplicity, one spectrum slot per SPSLG is typically created. A spectrum slot is a continuous spectrum characterized by a central frequency and a channel width, according to the following rules: the central frequency must fall within the C-Band (191.3 THz to 196.1 THz), and the channel width must range from 37.5 GHz to approximately 5 THz, following the equation $W = K \cdot 6.25 + 37.5$, where K is an integer. In each port, created spectrum slots must not overlap. Another crucial parameter is the direction: a slot can be unidirectional (transmitting or receiving) or bidirectional (transmitting and receiving).

The creation of a spectrum slot facilitates filtering an incoming signal by dividing the entire stream into specific slots characterized by central frequency and width. The divided signal inside each channel can be managed separately according to specified rules, allowing some channels to be dropped and others forwarded.

The second step involves creating the OTSiA, which serves as the carrier of the signal, allocating resources to transport the signal contained within the spectrum slot. This step must be performed for each port the signal crosses, with the OTSiA associated with a specific spectrum slot of a specific port, sharing the same central frequency, width, and direction.

The third step allows the ROADM to connect two spectrum slots physically linked by a MUX or DMX within a degree, creating a connection between a client port and a network port. This command requires specifying both endpoints (the spectrum slots of each port). If the connection is bidirectional, the port order is irrelevant; for unidirectional connections, the receiving spectrum slot must be specified first, followed by the transmitting slot. The connected slots must have the same central frequency and width and belong to a network port and a client port, respectively.

Finally, the OTSiA state for each port must be set to in-service (IS) to initiate signal forwarding through all relevant ports. By default, the OTSiA state is out-of-service (OOS) upon creation. If not switched to IS, no change in the signal spectrum is observed as the channel remains inactive.

These steps are illustrated in Fig. 3, showing the required actions to set up a connection within a MUX or DMX. These processes are encapsulated in the driver controlling the Adtran model and are executed when the OpenROADM agent sends a command to create a connection between two ports of a degree. The necessary parameters include the degree, the two ports to be connected (according to the datastore mentioned in Section II-B), the central frequency, and the channel width (specified in MHz). These parameters must conform to the spectrum slot creation rules, and one port must be the network port while the other is a client port. The port order is significant as the stream is unidirectional.

To create a cross-connection, the function *make_connection* must be invoked two times: at first, the connection is performed inside the first degree, while a second call must be executed with the specification of the second degree.

On the contrary, the *delete_connection* function removes a connection between two ports, requiring the degree and the two ports as parameters.

Finally, it is important to note that the ROADM's operating system can detect if a connection cannot be established, providing error explanations in the response body. For example, it can identify if a spectrum portion is already occupied or if the chosen central frequency is outside the C-band. These features enable parameter adjustments to ensure correct function execution.

IV. Results

Figs. 6 and 7 show NETCONF handshake snapshot where OpenROADM capabilities are retrieved: this is the initial step of the environment creation in which the OpenROADM agent runs. Furthermore, when the OpenROADM agent is launched, the *init* function is invoked. As outlined in Section III, this function not only configures the initial environment, but also provides the token string required for executing commands directed to the ROADMs.

To verify the interaction between ONOS and the Adtran model of the ROADM, the function tested was *make_connection*. The ROADM output is depicted in Fig. 4. The plot illustrates that six channels were established, each characterized by a constant width of 62.5 GHz, with varying

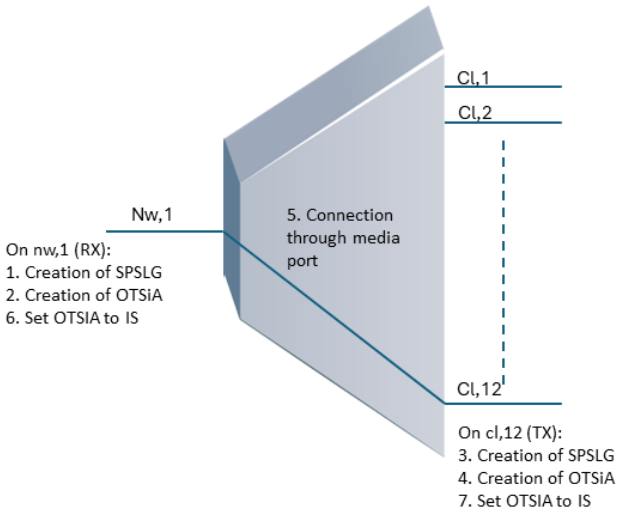

Fig. 3. Connection between a network and a client port

central frequencies. Specifically, six connections were created and forwarded to the network port within the first degree of the first ROADM of the optical line. For this scope, the *make_connection* function was invoked six times, with each call specifying the necessary parameters for connection creation as explained in Section III.

Fig. 4 presents results obtained from both the internal Optical Channel Monitor (OCM)[14] of the ROADM and the Optical Spectrum Analyzer (OSA)[2]. The integrated OCM displays six points, one per channel, while the OSA provides a more precise curve representing the entire power spectrum within the C-Band. The OCM's resolution is equivalent to the channel width, with a minimum resolution of 37.5 GHz, and in this case, 62.5 GHz. The OCM output represents the integral of the power within the channel; hence, increasing the channel width results in a higher power integral, yielding higher values. Each network port is equipped with an OCM capable of providing power values associated with channels in both directions. To verify the reliability of OCM values, the spectrum obtained via the OSA is also plotted. The peaks of the channels in the OSA spectrum correspond to the power values recorded by the OCM.

These results affirm that the OCM integrated with the ROADM is a valuable tool, enabling users to verify if the signal behaves correctly. In general, the OSA is an instrument used in the labs for double checking the results, but it is not present in the optical line. Since it is not possible to access to the OSA, also for his expensive cost, the integrated OCM is a good instrument that allow a first approximation of the power evolution at each ROADM.

The established channels are subsequently propagated to another ROADM at the end of OMS. In this case, as illustrated in Fig. 5, plots obtained with the OCM and the OSA are provided, capturing power values at the exit of the last amplifier, before the fiber enters in the receiving network port of the first degree of the ROADM that terminates the optical line. The plot shows that the six channels created at the line's origin are clearly visible. The primary difference lies in the power values of the channels. Fig. 4 presents measurements at the exit of the first ROADM's multiplexer, before the first amplifier, where channel peaks are low. On the contrary, Fig. 5 shows channel peaks much higher, influenced by the last amplifier's gain. In general, a ROADM hosts an amplifier at the exit of the MUX, at which is correct to measure the power that enters in the optical line. In this context, the first amplifier is outside the ROADM module: the power measured at the MUX is much lower if compared to the exit of the amplifier. To obtain consistent values, power measurement at the transmitter should be performed after the first amplifier.

V. Conclusion and Comments

Fig. 4 and 5 are crucial as they demonstrates the feasibility of establishing a connection between a controller and a device produced by different vendors. This finding is significant as it supports the viability of multi-vendor networks. In this scenario, the device successfully executes a command delivered by the controller, facilitated by middleware capable of translating from a standard to a proprietary model. This additional layer is necessary to establish a communication towards the devices and must be adapted to the examined entity. Additionally, it allows the devices within a network to use the same controller's standard, independently from the model: the introduction of a middleware layer represents a fundamental support for the interoperability in multi-vendor networks, avoiding a redefinition of the SDN controller. This layer may introduce latency since the communication between the controller and the devices is not direct and requires additional resources to handle the interoperability, but it strongly

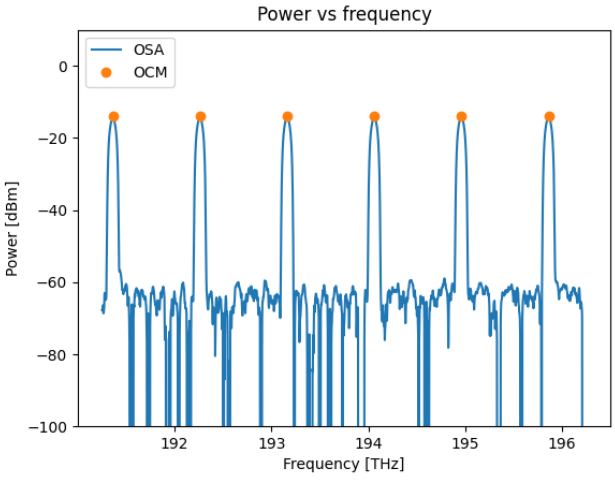

Fig. 4. This is a plot obtained retrieving the data at the beginning of the optical line, from the OCM inside the ROADM and the OSA connected at the exit of the ROADM

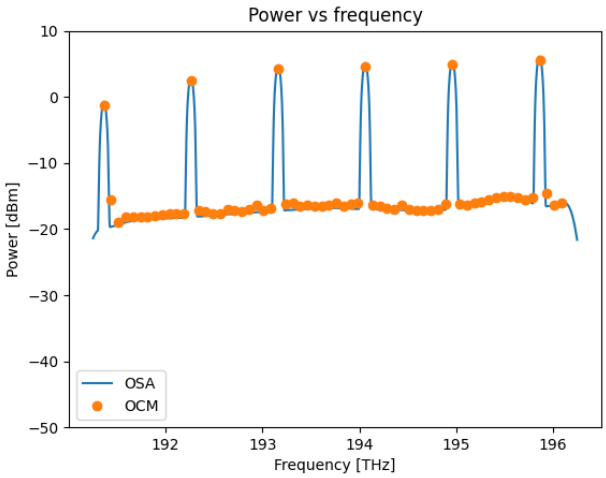

Fig. 5. This is a plot obtained retrieving the data from the OCM inside the ROADM and the OSA at the exit of the last amplifier of thee optical line

```
(openroadm@172.17.0.2) Password:
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:startup:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explic
    <capability>urn:ietf:params:netconf:capability:url:1.0?scheme=scp,file</capability
    <capability>http://openconfig.net/yang/telemetry?module=openconfig-telemetry&amp;r
    <capability>http://openconfig.net/yang/telemetry-types?module=openconfig-telemetry
    <capability>http://openconfig.net/yang/openconfig-ext?module=openconfig-extensions
    <capability>http://openconfig.net/yang/types/inet?module=openconfig-inet-types&amp
    <capability>http://org/openroadm/ethernet-interfaces?module=org-openroadm-ethernet
    <capability>http://org/openroadm/lldp?module=org-openroadm-lldp&amp;revision=2022-
    <capability>http://org/openroadm/network-media-channel-interfaces?module=org-openr
    <capability>http://org/openroadm/media-channel-interfaces?module=org-openroadm-med
    <capability>http://org/openroadm/optical-channel-interfaces?module=org-openroadm-o
    <capability>http://org/openroadm/optical-transport-interfaces?module=org-openroadm
```

Fig. 6. Client NETCONF capability request. In the snapshot subset of the capabilities are shown.

```
netopeer-server[1]: User 'openroadm' authenticated.
netopeer-server[1]: Received an SSH message "request-channel-open" of subtype "session".
netopeer-server[1]: Received an SSH message "request-channel" of subtype "subsystem".
netopeer-server[1]: Writing message (session 2): <?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <capabilities>
    <capability>urn:ietf:params:netconf:base:1.0</capability>
    <capability>urn:ietf:params:netconf:base:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:writable-running:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:startup:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:rollback-on-error:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:interleave:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:notification:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.0</capability>
    <capability>urn:ietf:params:netconf:capability:validate:1.1</capability>
    <capability>urn:ietf:params:netconf:capability:with-defaults:1.0?basic-mode=explicit&amp;
    <capability>urn:ietf:params:netconf:capability:url:1.0?scheme=scp,file</capability>
    <capability>http://openconfig.net/yang/telemetry?module=openconfig-telemetry&amp;revision=
    <capability>http://openconfig.net/yang/telemetry-types?module=openconfig-telemetry-types&
    <capability>http://openconfig.net/yang/openconfig-ext?module=openconfig-extensions&amp;re
    <capability>http://openconfig.net/yang/types/inet?module=openconfig-inet-types&amp;revisi
    <capability>http://org/openroadm/ethernet-interfaces?module=org-openroadm-ethernet-interf
    <capability>http://org/openroadm/lldp?module=org-openroadm-lldp&amp;revision=2022-12-09</
    <capability>http://org/openroadm/network-media-channel-interfaces?module=org-openroadm-ne
    <capability>http://org/openroadm/media-channel-interfaces?module=org-openroadm-media-chan
    <capability>http://org/openroadm/optical-channel-interfaces?module=org-openroadm-optical-
    <capability>http://org/openroadm/optical-transport-interfaces?module=org-openroadm-optica
```

Fig. 7. Middleware NETCONF capabilities subset reply logs.

simplifies the overall communication among the diverse entities that populate the network.

The development of a multi-vendor network necessitates a comprehensive understanding of various device models, that comprehends both hardware and software perspectives. To create code that effectively manages the device under examination, it is essential to study the supported protocols and determine the commands, written in the syntax of the chosen protocol, that the device must execute. Knowledge of the hardware is also crucial, as it provides insight into the level at which the developed software operates. For instance, understanding the physical structure of the Adtran ROADMs used in the setup is necessary for configuring commands appropriately. Additionally, familiarity with the selected controller and the chosen standard is fundamental.

Thus, the establishment of a multi-vendor network is the culmination of developing a standard that can communicate with all devices and thoroughly studying the specific devices within the network. In this context, the development of code for translating OpenROADM commands to the proprietary model necessitates an in-depth understanding of managing the Adtran ROADM model, particularly regarding the essential commands required to fulfill OpenROADM requests delivered by the ONOS controller.

Acknowledgment

Authors wish to thank Open Optical & Packet Transport (OOPT) group in TIP for the support. This work has been partially supported by the EU Horizon Europe research and innovation program, ALLEGRO Project, GA No. 101092766. Authors also wish to thanks IP Infusion.

References

[1] Renato Ambrosone et al. “Open Line Controller Architecture in Partially Disaggregated Optical Networks”. In: *2023 International Conference on Photonics in Switching and Computing (PSC)*. IEEE. 2023, pp. 1–3.

[2] Douglas M Baney, Bogdan Szafraniec, and Ali Motamedi. “Coherent optical spectrum analyzer”. In: *Ieee Photonics Technology Letters* 14.3 (2002), pp. 355–357.

[3] Andy Bierman, Martin Bjorklund, and Kent Watsen. *RESTCONF protocol*. Tech. rep. 2017.

[4] Giacomo Borraccini et al. “Experimental demonstration of partially disaggregated optical network control using the physical layer digital twin”. In: *IEEE Transactions on Network and Service Management* (2023).

[5] Vittorio Curri. “Digital-twin of physical-layer as enabler for open and disaggregated optical networks”. In: *2023 ONDM*. IEEE. 2023, pp. 1–6.

[6] Rob Enns. *NETCONF configuration protocol*. Tech. rep. 2006.

[7] Alessio Giorgetti et al. “Control of open and disaggregated transport networks using the Open Network Operating System (ONOS)”. In: *Journal of Optical Communications and Networking* 12.2 (2020), A171–A181.

[8] *http://www.openroadm.org*.

[9] *https://github.com/CESNET/netopeer*.

[10] *https://raw.githubusercontent.com/wiki/OpenROADM/OpenROADM_MSA_Public/files/ OpenROADM_MSA_Device_Model _WP_13.1.pdf*.

[11] *https://rawgit.com/CESNET/libnetconf/master/doc /doxygen/html/d9/d25/transapi.html*.

[12] Andrea Sgambelluri et al. “OpenConfig and OpenROADM automation of operational modes in disaggregated optical networks”. In: *IEEE Access* 8 (2020), pp. 190094–190107.

[13] Akhilesh S Thyagaturu et al. “Software defined optical networks (SDONs): A comprehensive survey”. In: *IEEE Communications Surveys & Tutorials* 18.4 (2016), pp. 2738–2786.

[14] Valery I Tolstikhin et al. “Monolithically integrated optical channel monitor for DWDM transmission systems”. In: *Journal of lightwave technology* 22.1 (2004), p. 146.