Cloud-native optical network automation platforms

van Quan Pham

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAS005

Thèse de doctorat

Cloud-Native Optical Network Automation Platforms

(Plateformes d'automatisation natives en nuage (Cloud) des réseaux optiques)

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Telecom Sudparis

École doctorale n°626 : École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat : informatique

Thèse présentée et soutenue à Palaiseau, le 31/05/2022, par

VAN-QUAN PHAM

Composition du Jury :

Thomas Clausen Professor, Laboratoire d'informatique de l'École polytechnique	Président
Stefano Secci Professor, CNAM	Rapporteur
Marco Ruffini Associate Professor, Trinity College Dublin	Rapporteur
Maurice Gagnaire Professor, Télécom Paris	Examinateur
Djamal ZEGHLACHE Professor, Télécom SudParis	Directeur de thèse
Dominique Verchere Senior Research Engineer, NOKIA Bell Labs France	Co-directeur de thèse
Esther Le Rouzic Cadre Scientifique, Orange Lab	Examinateur

INSTITUT POLYTECHNIQUE DE PARIS

DOCTORAL THESIS

Cloud-Native Optical Network Automation Platforms

Author:
Quan PHAM VAN

Supervisor:
Prof. Djamal ZEGHLACHE
Dr. Dominique VERCHERE

June 20, 2022

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisors: Professor Djamal Zeghlache and Dr. Dominique Verchere, for their continuous guidance and insightful assistance. Without their precious support it would not be possible to conduct this research.

I would like to thank each and every one of Bell Labs, specially their encouragement but also for promoting me to widen my research from various perspectives.

I would like also to thank my colleagues in Telecom sudparis, Orange Lab, Telefonica, ONF, SENDATE project for the fruitful discussions and collaborations.

A special thanks goes to my family.

Contents

List of Figures

List of Tables

List of Abbreviations

SDN Software **D**efined **N**etworking

API Application Programming Interface.

ODTN Open Disaggregated Transport Network.

EON Elastic Optical Network.

GMPLS Generalize Multi-Protocol Label Switching.

IETF Internet Engineering Task Force.

ILP Integer Linear Programming.

BLP Binary Linear Programming.

NETCONF Network Configuration Protocol.

OAM Operation, Administration and Maintenance.

ONF Open Networking Foundation.

ONOS Open Network Operating System.

OSA Optical Spectrum Analyzer. OSNR Optical Signal to Noise Ratio.

PCE Path Computation Element.

PMD Polarization Mode Dispersion.

pre-FEC BER pre-Forward Error Correction Bit Error Rate.

QoT Quality of Transmission.

REST REpresentational State Transfer.

RF Radio Frequency.
ROADM Reconfigurable Optical Add&Drop Multiplexer.

RPC Remote Procedure Call.

RSA Routing, and Spectrum Assignment.

WDM Wavelength Division Multiplexing.

WSS Wavelength Selective Switch.

XML eXtensible Markup Language. YANG Yet Another Next Generation.

SNMP Simple Network Management Protocol.

SBVT Sliceable Bandwidth Variable Transponder.

OT Optical Terminal.

MIF Monitoring Intent Framework.

ODL OpenDaylight.

MOSI Max Occupied Slice Index.

BER Bit Error Ratio.

NMC Network Media Channel.

MC Media Channel.

OTSi Optical Tributary Signal.

ODU Optical Data Unit.

SNR Signal Noise Ratio.

DSR Digital Signal Rate.

Chapter 1

Introduction

Optical communications industry development and architecture transformation are driven by the enormous growth of traffic volume in emerging services such as Data Center (DC) cloud interconnection, ultra-bandwidth video services, and 5G mobile network services. Demand for cloud and video services is driving increasingly diverse traffic to the metro network. Traffic stays in the metro networks as operators, cloud providers, and co-location providers add data centers and keep content closer to end-users. Moreover, a new deployment of optical fibers and network elements is costly and requires long-term planning. Service providers are interested in high-speed rates optical interface (e.g., 1Tb/s) and the efficient use of the optical resource through an optical network that can dynamically provision and reconfigure network resources. In the cloud era, networking is DC-centric. Therefore, networks must support quick service provisioning, full automation, and open collaboration, and network security mechanisms must be enhanced to improve network reliability. These are the typical requirements and challenges of the next-generation optical communication networks

1.1 GMPLS & Software-Defined Networking (SDN)

In the past, the rapid development of optical transport system has increased usage of optical networks and optical devices. The diversity and complexity in managing these optical devices are the main motivations in enhancing MPLS to a more generalized version of MPLS known as GMPLS (Generalized MultiProtocol Label Switching). The GMPLS (Generalized MPLS) was introduced in 1999-2000 and was designed as a superset of the MPLS control plane protocols to "avoid reinventing the wheel"; by using and extending existing protocols for the unique characteristics of the circuit-world. It was thought that the use of the same protocols could lead to an intelligent, automated, unified control plane (UCP) for a variety of technologies packet, time-slots, wavelengths, and fibers [2]. However, GMPLS has failed in actual deployment in wide area networks. After a decade, GMPLS hasn't been commercial deployment as a unified control plane for all packet, time-slots, wavelengths, and fibers networks. Even there have been many interoperability demonstrations and standardized at the ITU, IETF and OIF. Most transport equipment vendors have implemented it in their switches with a proprietary implementation of the GMPLS control plane. GMPLS is ultimately just a bunch of control protocols and applications such as GMPLS RSVP-TE [3] (signaling), LMP [4] (link management), OSPF-TE [5][6], and ISIS-TE [7] (routing), PCEP [8] (path computation), path computation [9], multi-layer [10], network failure recovery [11] (protection, restoration) make GMPLS becomes quite complicated. There are two different control plane scenarios are considered to deploy a GMPLS network:

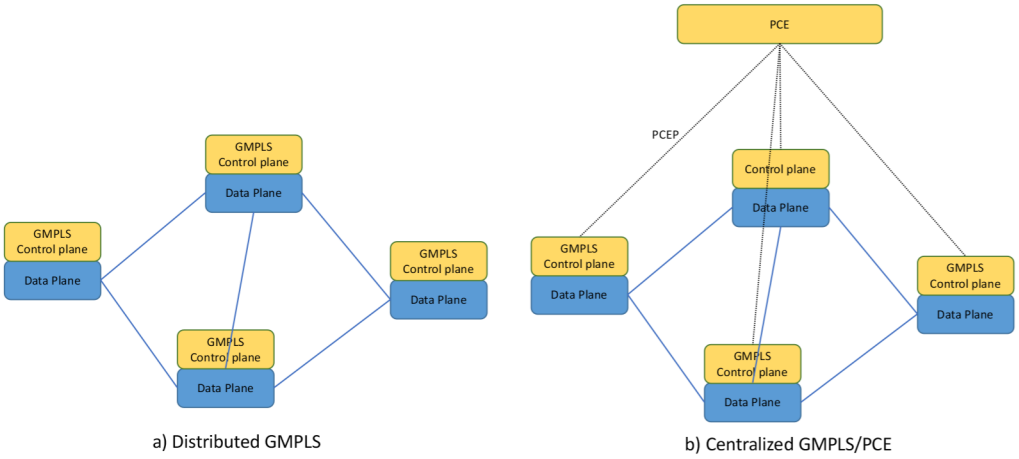

FIGURE 1.1: GMPLS Architectures

- The distributed GMPLS is shown as 1.1-a where path computation during both provisioning and restoration are locally performed at the control plane of network nodes. Each network node stores its own Traffic Engineering Database (TED) in the distributed scenario, including network topology and spectrum availability information. The TED is updated through Link State Advertisement (LSA) information exchanged through OSPF-TE protocol [5]. During provisioning, upon lightpath request, the source node computes a path and, using RSVP-TE [3], triggers the establishment of a Label Switched Path (LSP) (also called lightpath) for serving the specific request. During restoration, upon failure, the detecting node sends an RSVP-TE Notify [5] to the source node of each disrupted LSP and generates an OSPF-TE Router LSA identifying the failed link which is flooded on the network [3]. A source node receiving the RSVP-TE Notify performs the following actions. First, it sends an RSVP-TE Tear message along the working path of the disrupted LSP to release the resources utilized; then, it computes a new path. Once path computation is completed, the source node triggers RSVP-TE signaling to activate the computed backup path.

- The centralized GMPLS/PCE is shown in 1.1-b, where a centralized PCE is deployed to perform the path computations. In the centralized scenario, besides the TED, the considered Stateful PCE also stores an LSP database including information about all the LSPs currently established in the network [8]. The PCE communicates with the nodes through the Path Computation Element Protocol (PCEP) [8]. Upon lightpath request, the source node sends the PCEP PCReq message to ask for a path to the PCE during provisioning. The PCE performs the path computation and replies with a PCEP PCRep message, including the computed path and a suggested label indicating the spectrum slot to reserve. The source node will then trigger RSVP-TE to perform the LSP establishment. Similar to the distributed scenario, when a failure occurs, the detecting node sends an RSVP-TE Notify [5] to the source node of each disrupted LSPs and floods an OSPF-TE Router LSA for advertising the failure. When the source node receives the Notify message, it sends an RSVP-TE Tear message to release the resources used by the disrupted LSPs. Then a PCEP PCReq message is sent to request the computation of a backup path. Additionally, when the

Tear message reaches the destination node, a PCEP PCRpt message is sent to the PCE to report the released resources along the working path of the disrupted LSPs.

Recently, Software-defined networking (SDN) is an umbrella term encompassing several kinds of network technology to make the network as agile and flexible as the virtualized server and storage infrastructure of the modern data center. SDN is defined by three architectural principles: the separation of control plane functions and data plane functions, the logical centralization of control, and the programmability of network functions. Separation of control and data planes means that the decision about how to handle traffic is not made by the switch in the data plane but is taken by a centralized controller. Unlike traditional GMPLS, A logically centralized control plane in SDN (shown in 1.2) includes an SDN controller, and its applications are responsible for performing all functions such as network discovery, path computation, service provisioning, restoration, protection... The idea is that networks can be made easier to be managed (i.e., control and monitor) by going towards centralized solutions. Moreover, experience has shown that some traffic engineering problems, such as shared path protection, QoS, and adaptive load balancing [12], can be better solved with a global view of the network. Many other aspects of networking can also benefit from global optimization. A study in [13] has shown that SDN schemes significantly reduce the lightpath setup and recovery time with respect to GMPLS-based schemes. The programmability of network functions enables software automation that can react and reprogram the network without involving humans in the critical path. Previous interfaces to network devices mainly were designed for human interaction (e.g., the CLI: Command Line Interface) or narrow management functions (e.g., through SNMP: Simple Network Management Protocol).

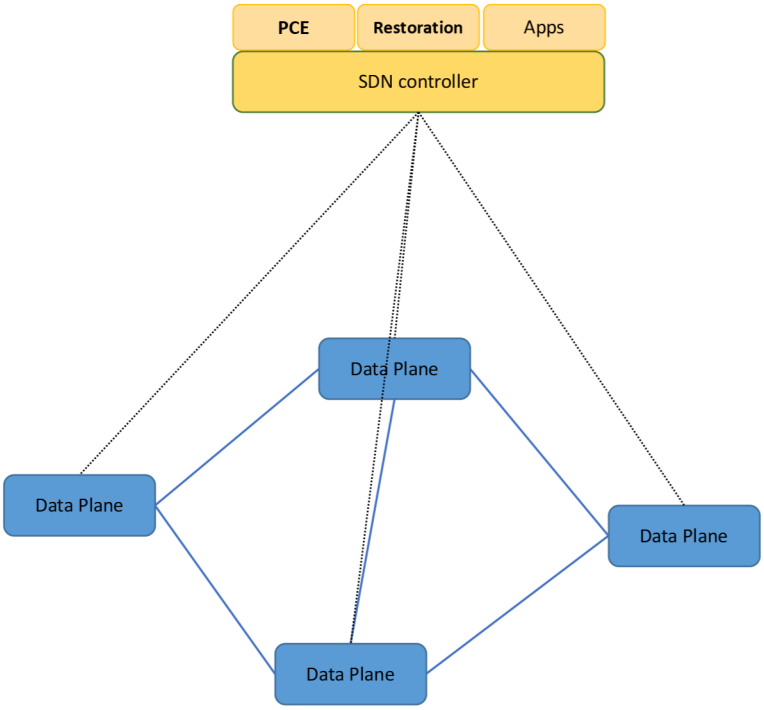

FIGURE 1.2: SDN Architecture

Although SDN was initially born for packet-switched networks, in recent years, SDN gained interest as a control plane solution for EONs. In particular, some studies have been made to enable remote configuration of the components of the optical node, such as the switching matrix and transmission characteristics (e.g., bit rate) [14]. However, we are at the early stage of introducing SDN in EONs, and work still needs to be done on the control plane to fully support the configuration of optical networks, e.g., SBVTs. Moreover, apart from the control plane, also the management plane needs improvements for supporting management mechanisms to reduce deployment and operational complexity and maximize the benefits of EONs capabilities [15].

1.2 Evolution of Optical Networks Control and Management

Optical transport systems and networks have historically been closed systems because optical signal transmission and switching technologies are complex. Because many optical system components such as transponders, amplifiers, wavelength multiplexers/demultiplexers, wavelength selective switches (WSS), gain equalizers are coupled. The control and management software of optical networks have been traditionally bound with these optical systems as well shown in 1.3 - left. Network disaggregation proposes to decouple the closed optical system equipment (OTs, ROADMs, Amplifiers...) according to their functions into independent and software-manageable hardware devices in 1.3 - right. Furthermore, disaggregation aims to separate the operating system from the underlying hardware, enabling the development of vendor-agnostic optical hardware. Although there have been numerous efforts to open up optical transport networks and move away from proprietary systems (e.g., the black link model supporting alien wavelengths), all of these optical systems still have to be designed together and optimized to get at least acceptable system performance, resulting in slow progress toward open disaggregated optical transport networks.

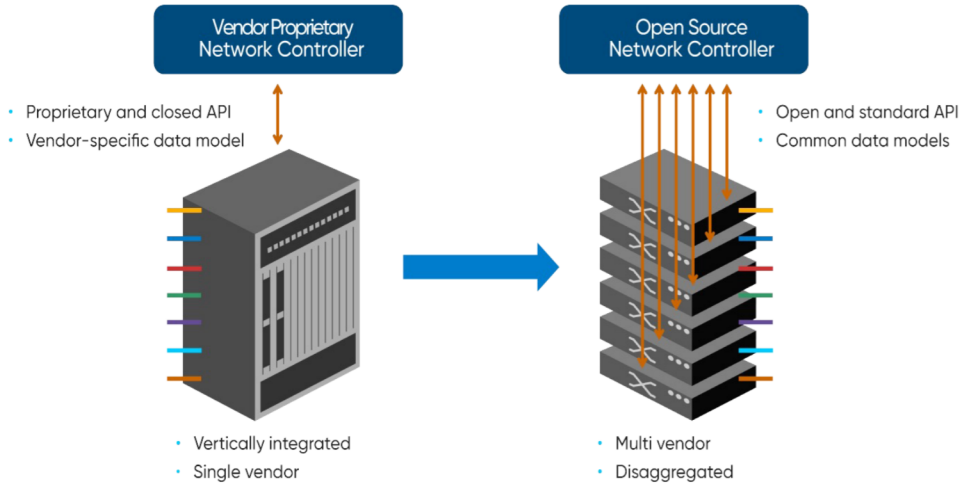

FIGURE 1.3: Disaggregation concept [1]

The traditional approach to controlling and managing optical networks is undergoing a transformation driven by the rapid advances in software-defined networking (SDN) technology. Current deployed optical networks (shown in 1.4-a) use proprietary technology from a vendor which is not directly interoperable with other

vendors and, as such, all devices (transponder, ROADM, amplifier), the network management system (NMS), optical planning, optimization, and monitoring tools are provided by the same vendor. The vendor lock-in is expanded as the network is maintained and evolves. This is holding back flexibility and innovation in the optical network arena.

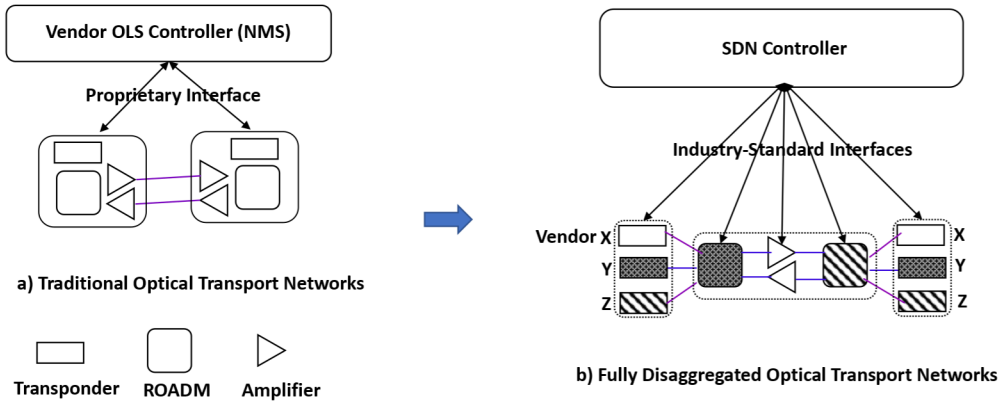

FIGURE 1.4: Open Dis-aggregated Optical Networks

Increased optical networking flexibility, innovation, and reduced vendor lock-in can be achieved by opening the optical network, i.e., disaggregating the network and deploying an open SDN controller that can control and manage a multi-vendor network via open and standard interfaces. The fully dis-aggregated optical network architecture (Fig. 1.4-b) is the ultimate goal of open dis-aggregated optical networks where the SDN controller configures and manages all devices from different vendors directly through open standard interfaces allowing the SDN controller to have direct access to the devices and providing it a complete view of the network topology. This fully disaggregated optical network architecture can be used for greenfield deployments; however, there are still many challenges such as standardization, interoperability testing, and validation challenges to be overcome. Furthermore, the SDN controller in fully disaggregated optical network architecture must be more robust, have high scalability and flexibility, and ease the integration of third-party software modules and applications.

While fully disaggregated optical network architecture remains many challenges need to be resolved, a pragmatic approach to opening an optical network is to evolve to the partially disaggregation architecture [16], [17]. It can be used for short-term brownfield deployment shown in Fig. 1.5) where the optical terminal (OT) (transponder) is the first component being removed from the vendor lock-in. Transponders have a rapid innovation cycle and thus enable operators to offer a new type of connectivity services with incremental speed, reach, and QoT, among other attributes. However, due to the lack of standardization at Layer 0 (L0) and the complexity of optical transmission due to the presence of physical impairments, the vendor's Open Line System (OLS) controller needs to remain in the network and be responsible for controlling the devices in the OLS domain via its vendor-proprietary interface and performing the optical-impairment-aware path computation. The OLS controller interfaces with the SDN controller via a standard northbound interface (NBI). The OLS domain is abstracted as a "network node" in the SDN controller topology. The OTs,

in this case, are controlled directly by the SDN controller via an open-source device data model.

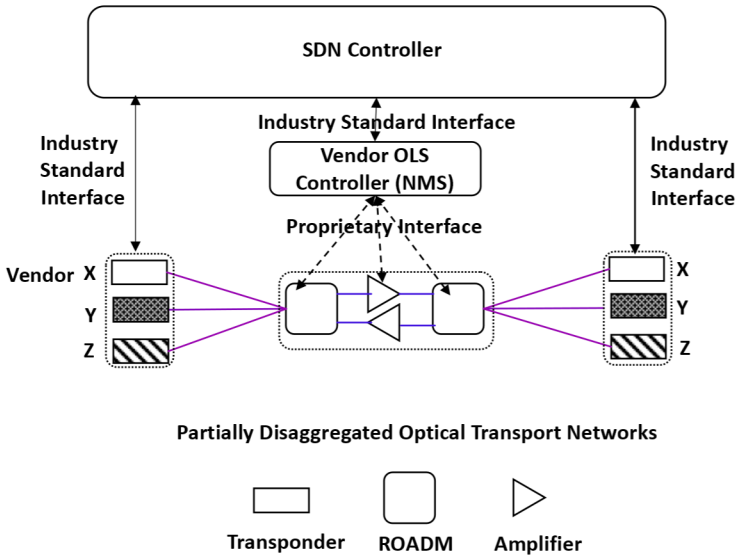

FIGURE 1.5: Open partially disaggregation architecture

1.3 Thesis contribution and organization

This PhD. Thesis investigates the possible solutions to enhance the control and management platform of the open disaggregated optical networks. In particular, Chapter 2 covers the SDN controller platform. Chapters 3 and 5 focus on dynamic resource allocation and optimization. Chapter 4 cover the closed-loop control scenarios. More in detail, the thesis is structured as follows:

- **Chapter 2** discusses the limitations of the monolithic SDN controller platform. Then, a Container-based Microservices SDN controller Platform (µONCP) is proposed, presented, and demonstrated its capabilities in a real hardware multi-vendor network testbed at a network operator labs with different setups such as single vendor fully disaggregated network setup, multi-vendor partially disaggregated setup and the mix of multi-vendor fully and partially disaggregated network setup.

- **Chapter 3** investigates and evaluates different RSA algorithms designed for the µONCP's Path Computation Function Application. BLP model is also proposed and evaluated with RSA algorithms via simulation.

- **Chapter 4** proposes and demonstrates two closed-loop control automation scenarios. in the first scenario, a graph-based solution implemented in the µONCP's Alarm Correlation application is presented. It demonstrates the alarm correlation capability in a real hardware testbed of a partially disaggregated network set up at a network operator lab. The second scenario demonstrates that the SDN-based QoT monitor reacts to performance degradation and dynamically fixes the misalignment between transmitter and receiver central frequencies.

- **Chapter 5** evaluates and demonstrates dynamic optical channel defragmentation showing the benefits of dynamic optical channel defragmentation in increasing the spectrum usage efficiency and reducing connectivity service blocking probability.
- Finally **Chapter 6** summarizes the thesis results and the Ph.D. achievements, including the list of publications and other accomplishments.

Chapter 2

Container-based Micro-services Optical Network Controller Platform

The new generation of optical transport network architecture has been formed by SDN technology, where a centralized controller controls the entire network. The centralized controller manages network resources (port, capacity, spectrum...) and performs complex functions such as global network planning, monitoring, optimization, resource slicing, root cause analysis, correlation... With recent advances in applying machine learning technology to optical networks and the requirements of open dis-aggregated multi-vendor optical networks, the centralized controller has to provide the capability to support fast and straightforward integration of new network control functions and vendor driver plugins. It also has to be flexible and scalable enough to support the transformation of optical architecture from partially dis-aggregated to fully dis-aggregated deployment.

2.1 Contribution

This research activity investigates and proposes a new software architecture and detailed implementation of the Centralized SDN controller platform that provides a modular network control plane called Container-based Micro-services Optical Network Controller Platform (µONCP). The µONCP provides the capabilities to instantiate, upgrade, and automate and on-demand deployment of the controller platform modules and their applications. The µONCP relies on the micro-services architecture, standardized YANG data model such as TAPI, OpenConfig, and in combination with Docker container [18] technology to allow: Network control platform as a service, automated and on-demand deployment of controller's modules or applications, and on-the-fly upgrades or swap of the software components. In the container-based micro-services, the controller's modules and its applications are built as independent micro-service components running inside the docker containers, as shown in 2.4, the modules communicate with each other over a messaging system (e.g., Rest/Restconf and Kafka). This allows the applications to be developed in any programming language (e.g., C, Java, Python), hence rely on their libraries, and to be independently tested. This µONCP enables automation of Continuous Integration/-Continuous Delivery (CI/CD), which reduces CAPEX and OPEX of SDN software development and operations. The controller and its applications are managed and orchestrated by an orchestrator combined with CI/CD tools such as Git and Jenkins to automate the testing and validation of each element. The orchestrator allocates the IT resources for each software module, such as CPU and memory. Furthermore,

the µONCP allows each of its modules to scale horizontally to adapt to the size of optical network infrastructure and the transformation of optical architecture from partially dis-aggregated to fully dis-aggregated deployment.

2.2 Current SDN controller

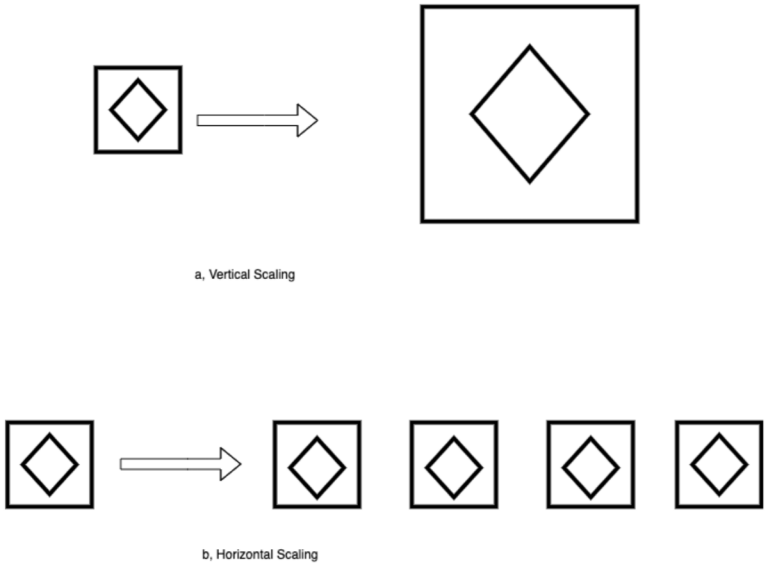

FIGURE 2.1: a) Vertical and b) Horizontal Scaling

The traditional approach to the control and management of the optical networks is shifting along with the rapid advances in software-defined networking (SDN) technology. The separation of control and data plane pushed the development of the logically centralized control plane. Current open-source projects (e.g., ONOS, ODL) are monolithic, as shown in 2.2, and resource-hungry software integrating both SDN controller and its applications, which is simple to test, deploy and scale vertically. However, this leads to non-efficient use of the resources and does not provide scaling mechanisms dealing with high loads of connectivity service requests or many network devices. Moreover, as more and more applications and features are being developed and integrated into the SDN controller platform, the limitations of monolithic software appear: Addition of new features requires to be planned to rebuild and redeploy the controller, vertical scalability shown as 2.1-a of IT resources, hard to maintain the source code... The controller platform also needs the enhanced capability of instantiating, upgrading, automated and on-demand deployment of the controllers and the applications such as path computation, defragmentation, reoptimization, recovery or quality of transmission,... to be adapted to the specifics of the optical network infrastructures [19]. This requires an evolution in the control plane architecture from monolithic to microservices, including the SDN controllers and the SDN applications. Most of the studies listed in [20] have been done to solve the controller placement problem considering a variety of objectives such as minimizing the latency between controllers and their connected switches, enhancing the reliability of the control network, or minimizing deployment cost and energy consumption. These previous studies are at the level of algorithm design and performance evaluation. The issues regarding how first to apply a controller placement

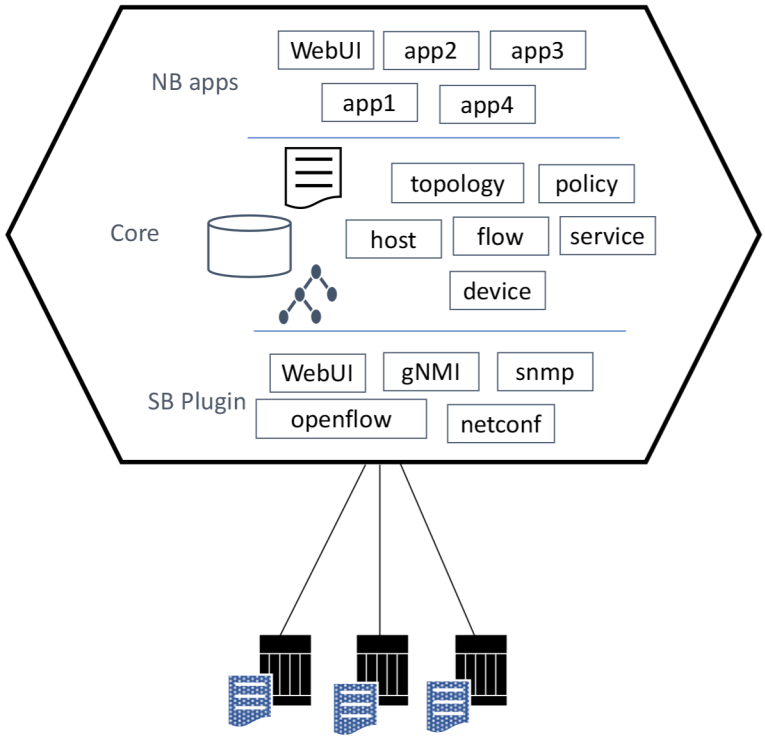

FIGURE 2.2: Monolithic controller

algorithm and then to deploy or migrate the SDN controllers and the SDN applications in a real network control plane platform has not been addressed yet, including with an experimental prototype.

2.3 Proposed Micro-services Controller Architecture

To overcome the monolithic controller's challenges, the microservices-based controller is proposed as shown in 2.3. The micro-services architecture style naturally results in a modular platform with clean interfaces. The big monolithic controller software stack is decoupled into small software modules. The modules communicate with each other over a well-defined and standardized messaging system. Small components make the system horizontally (multiple instances of the same component type) scale up and down quickly. It also provides the controller resiliency where the micro-services are monitored and restarted in case of misbehavior.

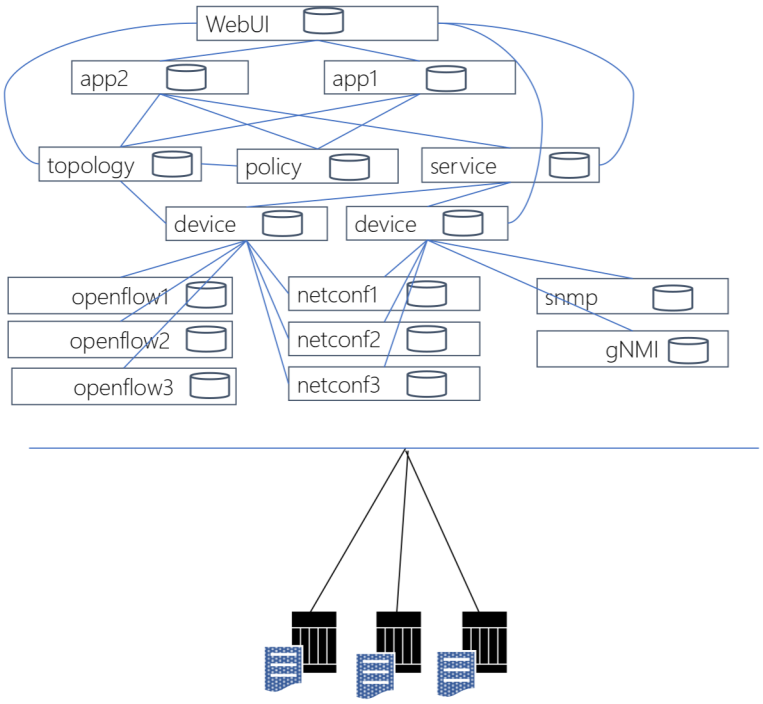

FIGURE 2.3: Micro-services controller

2.4 µONCP System Architecture and Implementation

In this section, the proposed controller software architecture is presented together with the detailed implementation of each module/component of the controller. Then, the use-case scenarios such as topology discovery and service-provisioning are described.

2.4.1 System Architecture

Unlike the SDN controllers currently being developed as a single monolithic composed of a software stack integrating the SDN controller and its applications as shown in 2.2. The µONCP is a collection of small micro-services software modules in containers running on the Kubernetes platform, as shown in 2.4. The micro-service architecture style naturally results in a modular platform with clean interfaces, while Kubernetes orchestration enables µONCP to scale up and down quickly. It also provides the controller resiliency where the microservices are monitored and restarted in case of misbehavior. Furthermore, each software module can quickly scale horizontally in case of overload. The µONCP is composed of 4 main layers: Southbound plugins, Core functions, Northbound applications, and Workflow Manager Orchestration.

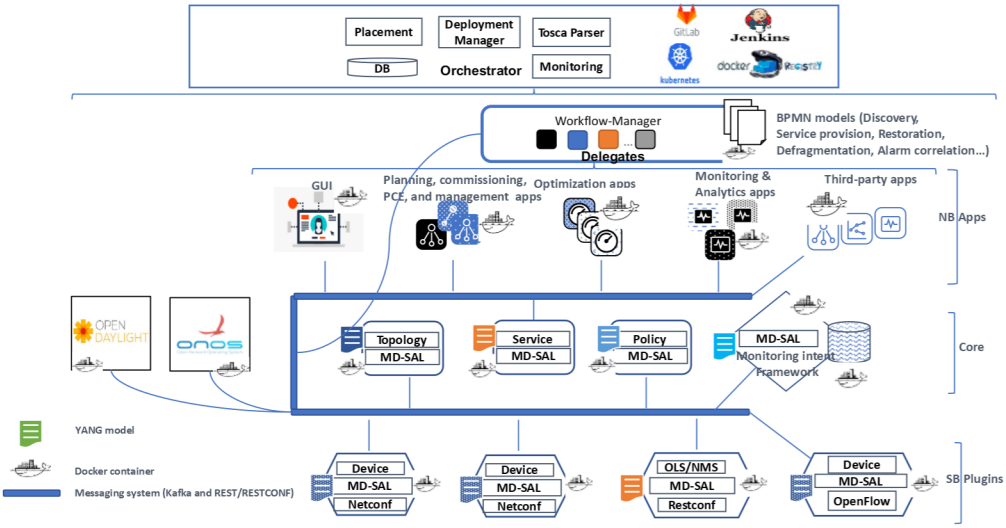

FIGURE 2.4: µONCP System Architecture

A common **Data Model-driven Development Approach** is used to design and develop southbound plugin modules and the core function modules. Each module is developed based on OpenDaylight Model-Driven Service Abstraction Layer (MD-SAL) as the key element and built with its own yang models. MD-SAL is an infrastructure component that provides messaging and data storage functionality based on data and interface models defined by application developers (i.e., user-defined models). MD-SAL has two main objectives: i) Define a standard layer, concepts, data model building blocks, and patterns that provide an infrastructure/framework for applications and inter-application communication. ii) Provide support for user-defined transport and payload formats, including payload serialization and adaptation (e.g., binary, XML or JSON). MD-SAL uses YANG as the modeling language for both interface and data definitions to achieve the above objectives and provides a messaging and data-centric runtime for such services based on YANG modeling. MD-SAL extends the YANGTools project (which provides a YANG parser, data structures, and data supporting functionality) with messaging patterns and the concept of data stores/data brokers.

YANG is a data modeling language standardized by IETF [21]. YANG has been developed and standardized as a language to model data into NETCONF messages. A Yang module can be translated into an XML representation which makes YANG also adaptable to other protocols (e.g., RESTCONF, Kafka) beside NETCONF. In recent years, operators and vendors have been increasingly interested in YANG because of the possibility of standardizing standard models for configuration and management data in a vendor-neutral way. YANG also supports the "deviations" from the standard model to enable a vendor to adapt and augment the original model with advanced, differentiated features. YANG model contains two types of data: configuration and state (operational) data. An external entity explicitly sets configuration data on the system (e.g., Southbound plugin, northbound application). State data cannot be set by the external entity but can be read. There are two ways to interact with the device's models, synchronously or asynchronously. In the case of synchronous communication, each configuration command issued by the controller is then confirmed or rejected. The controller knows the success or failure of the configuration change that it makes (and hence it knows the exact configuration of the

system). In this case, the value of any configuration variable always reflects the configured value. In the case of asynchronous communication, each transaction is written, but the device does not synchronously return success or failure. The value of a configuration variable may not reflect the controller's intent. Thus a field can have different actual vs. intended values. Due to this ambiguity, each configuration data is also replicated as state data representing the device's current state. YANG also supports the definition of "Notification" to model the content of NETCONF Notification messages, which indicate that certain events have been recognized (e.g., a failed link). Moreover, although YANG is mainly considered a data modeling language, it also provides the possibility to define executable functions through Remote Procedure Calls (RPCs) that specify the name, the input, and the output parameters of a specific function, e.g., switching on(off) a device inside a node. Then, some considerations are here reported on the nature of YANG. First, it is a highly readable text language. This significantly simplifies management and troubleshooting operations than protocols relying on bit encoding, which requires ad-hoc software to parse encoded information, such as OpenFlow. Nowadays, handling a text file instead of bit encoding does not represent a challenge. Moreover, in the case of bit encoding, the support of novel parameters at the data plane would imply redesigning the protocol messages' content, such as header and objects. On the contrary, thanks to the nature of YANG, when the model changes, the YANG model can be refined without redesigning the protocol, thus providing a much more practical solution with respect to bit encoding. Such an example must be considered relevant given the continuous evolution of the technology in the data plane.

LISTING 2.1: example.yang.

```
module example {
    namespace "sssup:example";
    prefix example ;

    typedef NEW-TYPE{
        type enumeration {
            enum type-one;
            enum type-two;
            enum type-three;
        }
    }
    leaf data-1 {
        type uint16 ;
    }
    leaf data-2 {
        type decimal64 {
            fraction-digits 18;
        }
    }
    leaf data-3 {
        type NEW-TYPE ;
    }
    list element-of-a-list {
        key "leaf-data-1";
        leaf leaf-data-1 {
            type uint16 ;
        }
        leaf leaf-data-2 {
            type uint16 ;
        }
    }
};
```

LISTING 2.2: example.yang tree.

```
module example
    +--rw data-1? unit16
    +--rw data-1? decimal64
    +--rw data-1? NEW-TYPE
    +--rw element-of-a-list? [leaf-data-1]
        +--rw leaf-data-1 unit16
        +--rw leaf-data-2? unit16
```

YANG can be hierarchically represented in a tree structure with a root and leaves. Listing 2.1 shows a generic YANG model and the resulting tree organization: root and leaves have names, data types, data values, and child leaves. For example, YANG defines data types as 16-bit unsigned integer (as "data-1", line 13 in Listing 2.1) or 64-bit signed decimal (as "data-2", line 17). The new data type can also be defined. In Listing 2.1, "data-3" is of the "NEW-TYPE" type. The new type can assume just three values (lines 5- 11). YANG also includes the definition of lists. The "key" of a list is used to specify one or more leaves in a list that will uniquely identify an element (data instance) of the list. The example in Listing 2.1 shows the model "example" composed of four leaves (each leaf is defined with the syntax "leaf", e.g., line 12): "data-1", "data-2", "data-3", and a list (line 23). Each of these data is associated with a type. A list is initiated with the command "list" (line 23), and the data of a list can have child leaves as "leaf-data-1" and "leaf-data-2". The resulting tree, obtained with the Pyang software, is visualized in Listing 2.2 with: i) "example" as the root; ii) "data-1", "data-2", "data-3", and the list as leaves; and iii) "leaf-data-1" and "leaf-data-2" as the leaves of each element in the list.

Data model-driven development approach addresses the requirements in terms of speed and scale of network automation and monitoring capabilities. The cost of IT operations can be reduced. Motivations aside, organizations are interested, and vendors are implementing these capabilities. However, to do so requires a new way of thinking, especially if someone has been in networking for a long time.

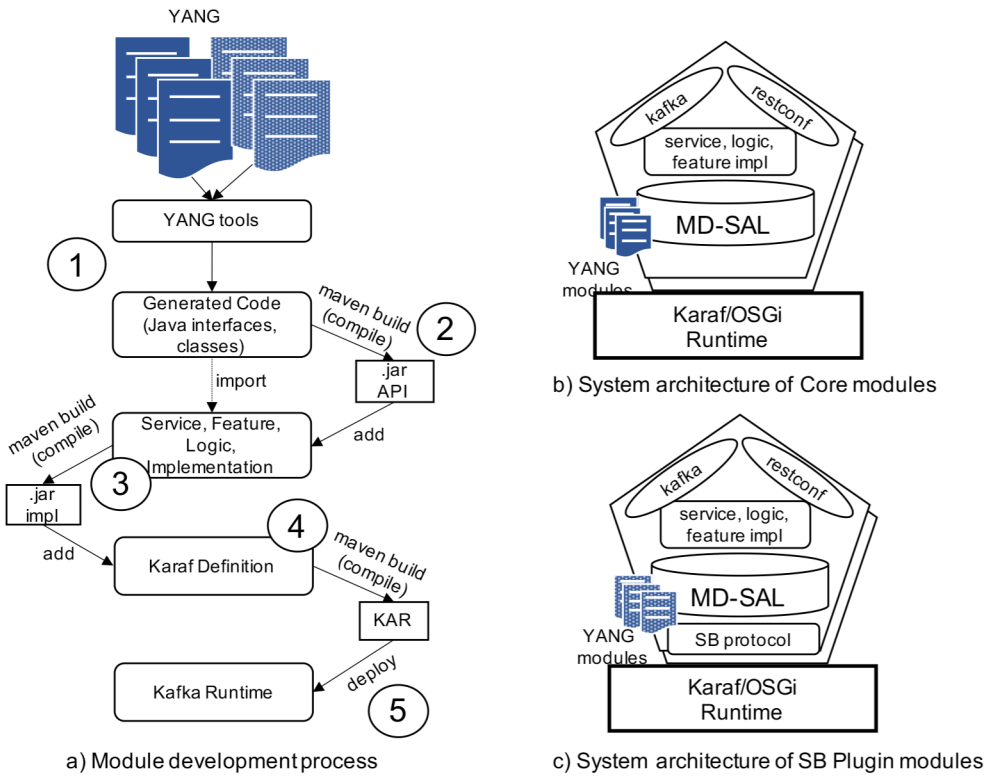

FIGURE 2.5: Module architecture and development process

As depicted in 2.5-b and c, Karaf/OSGi is used as the run-time environment of the SB plugin and core modules. Each module has its own YANG data models and interfaces. However, all modules have a common development process of 5 steps, as shown in 2.5-a. Starting from a set of YANG data models, the MD-SAL YANGtools compile and generate a jar file containing JAVA APIs (interface and classes). Using the generated JAVA APIs implements the service, logic inside each module becomes simple. After logic and service implementation, the module is compiled and ready to be deployed as run-time.

2.4.2 South-Bound Plugin Modules

The southbound plugin is an SDN enabler that provides a communication protocol between the control plane and the data plane. It is used for device discovery and also pushes configuration information to devices. It also provides an abstraction of the network device's functionality to the control plane. Major challenges of southbound interfaces are heterogeneity, vendor-specific network elements, and language specifications. There is a wide range of traditional network protocols and models, which creates heterogeneity issues. Every vendor has its own architecture for switching fabric and supports different configurations and languages. Southbound plugins resolve these issues in this architecture by providing an open and standardized protocol and model. The proposed architecture contains three southbound plugin modules: NETCONF/OpenConfig, OpenFlow, and RESCONF/TAPI.

OpenFlow module

OpenFlow is the first standard communications interface defined between an SDN architecture's control and forwarding layers. OpenFlow allows direct access and manipulation of the forwarding plane of network devices such as switches and routers, both physical and virtual (hypervisor-based) [22]. This provides us a way to control the behavior of switches throughout our network dynamically and programmatically. OpenFlow is a key protocol in many SDN solutions. To turn the concept of SDN into practical implementation, two requirements must be met. First, a common logical architecture must be in all switches, routers, and other network devices to be managed by an SDN controller. This logical architecture may be implemented in different ways on different vendor equipment and in different types of network devices, so long as the SDN controller sees a uniform logical switch function. Second, a standard, secure protocol is needed between the SDN controller and the network device [23]. Both of these requirements are addressed by OpenFlow. Indeed, OpenFlow is composed of OpenFlow switch and OpenFlow protocol shown in Figure 2.6. OpenFlow switch meets the first requirement, and the OpenFlow protocol meets the second requirement. Figure 2.6 illustrates the basic structure of the OpenFlow environment.

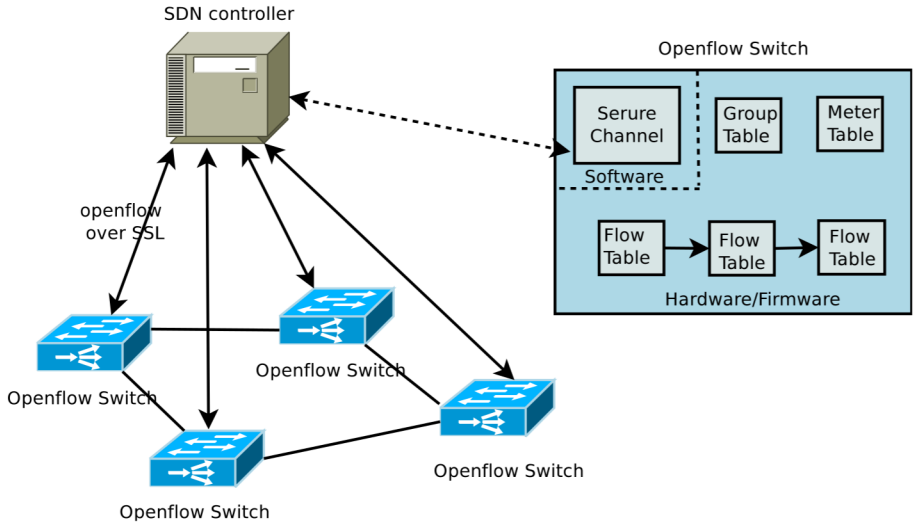

FIGURE 2.6: OpenFlow Switch

OpenFlow protocol describes message exchanges between a controller and an OpenFlow switch. Typically, the protocol is implemented on top of SSL or Transport Layer Security (TLS), providing a secure OpenFlow channel. The OpenFlow protocol enables the controller to add, update, and delete actions to the flow entries in the flow tables. It supports three types of messages:

- **Controller-to-Switch:** These messages are initiated by the controller and, in some cases, require a response from the switch. This class of messages enables the controller to manage the logical state of the switch, including its configuration and details of flow- and group-table entries. Also included in this class is the Packet-out message. This message is used when a switch sends a packet to the controller and the controller decides not to drop the packet but to direct it to a switch output port.

Message	Description
Features	Request the capabilities of a switch. Switch responds with a features reply that specifies its capabilities.
Configuration	Set and query configuration parameters. Switch responds with parameter settings.
FlowMod	Add, delete, and modify flow/group entries and set switch port properties.
Multipart	Collect information from switch, such as current configuration, statistics, and capabilities.
Packet-out	Direct packet to a specified port on the switch.
Barrier	Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.
Role-Request	Set or query role of the OpenFlow channel. Useful when switch connects to multiple controllers.
Asynchronous-Configuration	Set filter on asynchronous messages or query that filter. Useful when switch connects to multiple controllers.

TABLE 2.1: Controller-to-Switch OpenFlow Messages

- **Asynchronous:** These types of messages are sent without solicitation from the controller. This class includes various status messages to the controller. Also included is the Packet-in message, which may be used by the switch to send a packet to the controller when there is no flow-table match.

Message	Description
Packet-in	Transfer packet to controller.
Flow-Removed	Inform the controller about the removal of a flow entry from a flow table.
Port-Status	Inform the controller of a change on a port.
Error	Notify controller of error or problem condition.

TABLE 2.2: Asynchronous OpenFlow Messages

- **Symmetric:** These messages are sent without solicitation from either the controller or the switch. They are simple yet helpful. Hello messages are typically sent back and forth between the controller and switch when the connection is first established. Echo request and reply messages can be used by either the switch or controller to measure the latency or bandwidth of a controller-switch connection or just verify that the device is operating. The Experimenter message is used to stage features to be built into future versions of OpenFlow.

The OpenFlow protocol enables the controller to manage the logical structure of a switch without regard to how the switch implements the OpenFlow logical architecture. OpenFlow only defines the minimal set of hardware capabilities (via the switch specifications) and how to access them (via the OpenFlow channel), but not what to do with these switches, which allows the network operators to define their behaviors in their networks. Although OpenFlow was initially conceived for packet switching

Message	Description
Hello	Transfer packet to controller.
Echo	Echo request/reply messages can be sent from either the switch or the controller, and they must return an echo reply.
Experimenter	For additional functions.

TABLE 2.3: Symmetric OpenFlow Messages

(L2/L3), from release 1.4 [24], OpenFlow has been extended to support also circuit switching (L1/L0). In particular, for L0 switching, new optical port description, modification, and statistics properties were introduced for describing, configuring, and monitoring optical ports. However, these extensions are still incomplete. An extension of OpenFlow was proposed to enrich the optical port statistic properties to include a complete list of parameters for L0 flow monitoring, and we implement a monitoring function as an SDN application.

This section presents the proposed OpenFlow extensions to support the monitoring of the transponders. The Request and Reply OF messages have been extended to offer the new possibility of retrieving only a portion of the parameters. Listing 2.3 shows the parameters of an optical port that are currently supported in 1.5 (the latest release). It can be seen that only a few parameters are supported, in particular, the central frequency, the transmitted and received power, the temperature, and the bias current of the laser. Listing 2.4 shows the proposed extensions (in both) to support the monitoring of transponders. In particular, the Optical Port Statistic structure has been enriched with new parameters about the transmitter and receiver associated with a Line-port of transponders, such as the pre-Forwarding Error Correction Bit Error Rate (pre-FEC-BER), the Polarization Mode Dispersion (PMD), and the others. In addition, for each parameter in the model, these new extensions provide instants values and statistics, i.e., average, variance, minimum and maximum values, to enable the device to compute statistics locally and offload the control channel. Finally, a 32-big filter flag was added to both Optical Port Statistic Request and Reply messages to indicate which parameters are requested by the controller and supported/returned by the device.

LISTING 2.3: Standard OpenFlow 1.5 Optical Port Statistics.

```
struct ofp_port_stats_prop_optical {
    uint16_t   type;          /* OFPPSPT_OPTICAL. */
    uint16_t   length;        /* Length in bytes of this property. */
    uint8_t   pad[4];         /* Align to 64 bits. */
    uint32_t   flags;         /* Features enabled by the port. */
    uint32_t   tx_freq_lmda;  /* Current TX Frequency/Wavelength */
    uint32_t   tx_offset;     /* TX Offset */
    uint32_t   tx_grid_span;  /* TX Grid Spacing */
    uint32_t   rx_freq_lmda;  /* Current RX Frequency/Wavelength */
    uint32_t   rx_offset;     /* RX Offset */
    uint32_t   rx_grid_span;  /* RX Grid Spacing */
    uint16_t   tx_pwr;        /* Current TX power */
    uint16_t   rx_pwr;        /* Current RX power */
    uint16_t   bias_current;  /* TX Bias Current */
    uint16_t   temperature;   /* TX Laser Temperature */
};
```

LISTING 2.4: Proposed OpenFlow Extensions.

```
struct ofp_port_stats_prop_optical {
  uint16_t   type;          /* OFPPSPT_OPTICAL. */
  uint16_t   length;        /* Length in bytes of this property. */
  uint8_t   pad[4];         /* Align to 64 bits. */
  uint32_t   flags;         /* Features enabled by the port. */
  of_transmitter_t   transmitter;
  of_receiver_t   receiver;
};


struct of_transmitter {
  uint16_t length;
  uint16_t transmitter_id;
  uint32_t filter_flags;
  uint16_t bit_rate; /* Gbps */
  uint16_t baud_rate; /* GSps */
  uint16_t bandwidth; /* GHz */
  uint8_t modulation;
  of_fec_t  fec;
  of_inst_avg_var_max_min_t central_frequency; /* GHz */
  of_inst_avg_var_max_min_t output_power; /* dBm */
  of_inst_avg_var_max_min_t temperature; /* C*10 */
  of_inst_avg_var_max_min_t bias_current;  /* mA*10 *
};


struct of_receiver {
  uint16_t length;
  uint32_t flags;
  uint16_t bitRate;
  uint16_t baudRate;
  uint16_t bandwidth;
  uint8_t modulation;
  of_fec_t fec;
  of_inst_avg_var_max_min_t central_frequency;
  of_inst_avg_var_max_min_t input_power;
  of_inst_avg_var_max_min_t temperature;
  of_inst_avg_var_max_min_t bias_current;
  of_inst_avg_var_max_min_t sampling_rate;
  of_inst_avg_var_max_min_t pre_fec_ber;
  of_inst_avg_var_max_min_t pmd;
  of_inst_avg_var_max_min_t cd;
  of_inst_avg_var_max_min_t qFactor;
}
;

struct of_inst_avg_var_max_min {
  uint8_t flags;
  uint16_t instant;
  uint16_t average;
  uint16_t variance;
  uint16_t max;
  uint16_t min;
}:

struct of_fec {
  uint8_t fec_type;
  uint8_t message_length;
  uint8_t block_length;
};
```

In summary, this section presented a contribution to implementing an OpenFlow extension for optical networks. SDNs implemented using OpenFlow provides a powerful, vendor-independent approach to managing complex networks with dynamic demands. However, It was designed to operate packet switching networks. Adopt OpenFlow for optical networks would require many works in terms of design and implementation. OpenFlow for Optical remains as experimental research. Besides the OpenFlow protocol, network vendors and operators have also considered other existing protocols (different from OpenFlow) to enable the SDN infrastructure. One of these protocols is Network Configuration Protocol (NETCONF).

OPENCONFIG/NETCONF Module

NETCONF [25] is emerging as an SDN protocol standardized by IETF, which provides both control (e.g., data plane device configuration) and management functionalities (e.g., access to monitoring information). In particular, such protocol standardization is an answer to specific requirements of the IETF [26]:

- Developing standards for network configuration and management
- Use eXtensible Markup Language (XML) for data encoding.

NETCONF protocol provides mechanisms to install, manipulate, and delete management states and information of network devices. NETCONF also has the prospects to be particularly indicated for monitoring purposes through the use of specific messages such as notifications that are not available in other protocols such as OpenFlow. In particular, through NETCONF, a controller can subscribe to a notification when a specific monitored parameter exceeds a given threshold. In this case, the monitoring system triggers the NETCONF notification (i.e., an alarm), which is sent to the controller subscribed to such notification. NETCONF relies on the Yet Another Next Generation (YANG) [21, 27], a modeling language representable in XML. YANG enables the definition of the configurable parameters and state information of the different network devices in a standard way. Thus, YANG and NETCONF provide a standard way to control and manage network elements, independently from the vendor [26]. For these reasons, NETCONF and YANG gather interest from network operators.

NETCONF messages are illustrated in 2.7. Initially, the NETCONF transport protocol session is opened between two peers, named client (i.e., the centralized controller) and server (device), respectively. Thus, <*hello*> messages (1,2) are exchanged between the two peers to list the peer's capabilities (i.e. the capability identifiers), as a discovery phase. A capability describes a supported data model (e.g. the transponder model) or supported operations (e.g. support for notifications), and it is uniquely identified through an Uniform Resource Identifier (URI). A capability is described through YANG modeling language and the URI is the unique identifier for the specific YANG data modeling describing that capability. In this phase, the centralized controller discovers which type of data plane device has to control/manage. Once the session is opened, the NETCONF peers exchange <*rpc*> and <*rpc-reply*> messages. The <*rpc*> message is used to enclose a NETCONF command sent from the controller (client) to the device (server). The main commands encapsulated in the <*rpc*> message are the following: <*get*>, <*get-config*>, <*edit-config*>, and <*delete-config*>. The <*get*> command is used to retrieve the running configuration and state information of the device (3). The <get-config> command is used to retrieve only the configuration data exclusively (the state information is not returned). The <*edit-config*> command is used to write a specific configuration on the device (5) (e.g. to set

the maximum transmission unit of a specific Ethernet interface). Finally, the <*delete-config*> command is used to delete a specific configuration. The <*rpc-reply*> message is sent from the device to the controller in response to an <*rpc*> message. The response data for the given method invoked is encoded as one or more child elements enclosed in the <*rpc-reply*> message. Three are the possible responses encapsulated in the <*rpc-reply*>: <*ok*>, <*rpc-error*>, or the data requested by the operation. The <*ok*> response is sent if no errors or warnings occurred during the processing of the <*rpc*> request and no data was returned from the operation (6,8). The <*rpc-error*> response is sent if an error occurs during the processing of the <*rpc*> request. Otherwise, if some data was expected from the <*rpc*> request, the data is returned (4).

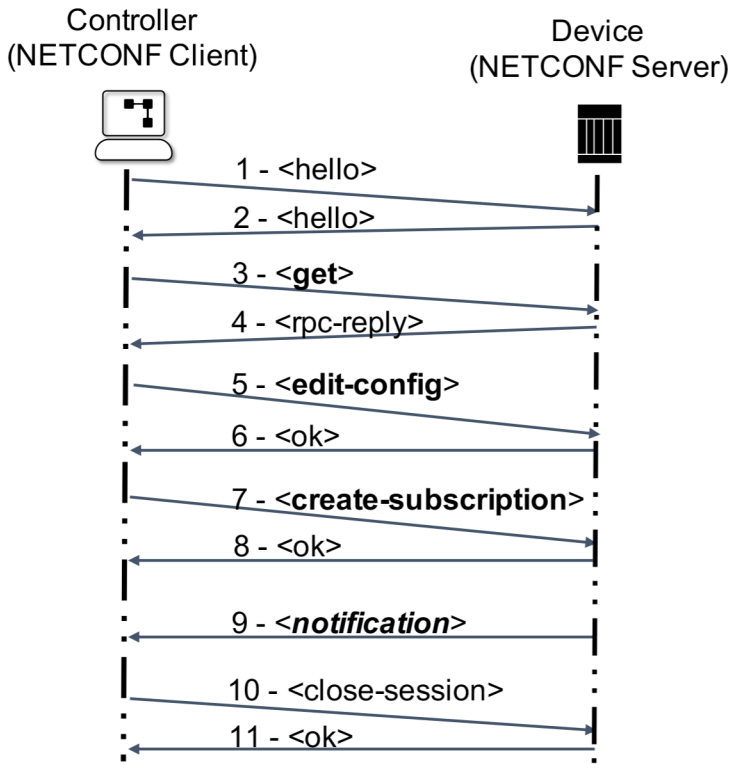

FIGURE 2.7: NETCONF message exchange.

NETCONF and YANG have been introduced in several networking scenarios. For example, in [28], the authors propose a NETCONF agent for link-state monitoring compared with other management technologies such as Simple Network Management Protocol (SNMP). In [29], the authors highlight the benefits provided by NETCONF in terms of security and scalability, compared with SNMP and REpresentational State Transfer (REST). In [30], the authors use YANG and NETCONF to manage a 10G-PON. In [31] and [32], some YANG network models are provided, including a preliminary version for the transponder.

In the context of optical networks, several standardization bodies and working groups (e.g., IETF, OpenConfig [31], OpenROADM [32]) have released YANG models. As an example, the IETF draft in [33] defines a YANG model for representing, retrieving, and manipulating traffic engineering (TE) topologies supporting optical-switching nodes. OpenROADM has recently defined YANG models focused on ROADM disaggregation. These models describe how different pluggable devices