POLITECNICO DI TORINO

Master's Degree in COMPUTER ENGINEERING

Master's Degree Thesis

Autonomous and Softwarized Management of Disaggregated Open Optical Networks

Supervisors

Prof. Vittorio CURRI

Giacomo BORRACCINI

Candidate

Renato AMBROSONE

Company Supervisors
Consortium GARR

Ing. Paolo BOLLETTA

October 2022

Abstract

Given the continuous growth of data traffic demand, the optical network management of current infrastructures has become a key factor: Internet pervasiveness has enlarging fast with fiber-to-the-home (FTTH), 5G and future 6G technologies. In the last decade, software-defined network (SDN) concepts have been applied within the optical communication world, making optical networks dynamic and programmable. This has been achieved thanks to the adoption of a single network operative system and the virtualization of the network elements (NEs) forming the physical layer (PHY). The latter can be achieved by means of device disaggregation, vendor-neutral control and multi-vendor inter-operability: Internet service providers and network operators are now interested in these solutions, avoiding vendor lock-in and cutting down on capital expenses. Network virtualization and disaggregation lay the foundation for the creation and the manipulation of a virtual object called digital twin (DT) of the network: a data structure modelling and emulating the behavior of the real system. In the wake of the Yet Another Next Generation (YANG) data model, other implementations were born aiming to unlock vendor-neutral control and independent device virtualization. Cooperating with the standardization bodies (e.g. IETF, ITU-T, IEEE, OIF), the largest part of Internet Service Providers (ISP) promoted various activities in order to achieve agreements for network disaggregation and automation. In particular, we have recently seen the birth of various consortia specialized in improving the efficiency of various aspects of the management of an optical network.

In this work, firstly, a SDN architecture is defined in a context of open and disaggregated optical networks. Consequently, an implementation for a centralized network controller managing open and disaggegated optical networks has been developed. The main goal is to enable the independent management of control and data planes, engaging dynamic operations and expanding the possibilities of optimization degrees. The conceived SDN architecture has three main actors: the optical network controller (ONC), each optical line controllers (OLCs) and the PHY-DT. The communication among the actors is performed defining a software orchestrator called open optical transport network controller (OOTNC), operating as dispatcher and integrating additional functionalities in terms of network management and control. Different data models, frameworks and protocols are investigated: Network Configuration Protocol (NETCONF) as a network management protocol, from IETF, YANG model for NE virtualization, from OpenConfig, reconfigurable optical add & drop multiplexers (ROADM) models, from OpenROADM, open network operating system (ONOS) as a SDN controller, from Open Networking Foundation (ONF), and GNPy for quality of transmission (QoT) estimation, from Telecom Infra Project (TIP). Using a custom representational state transfer (REST) application, ONOS is able to retrieve information regarding the network topology

and the network status. Furthermore, additional endpoints allow to set the estimated LP modulation format. GNPy is integrated within the PHY-DT as QoT estimator. The developed software framework has been conceived also to promptly react in case of link or node failures, firstly evaluating the amount of lost traffic and then recovering the issue.

"Stay hungry, stay foolish."
Steve Jobs

Acknowledgements

Having reached the end of my study path, I feel the need to thank all those who have been by my side over the years, all those who have dedicated a minute to me, all those who have a word towards me.

First of all I have to thank all the PLANET team for the support they have provided me over the last few months. In particular, I want to thank my supervisor Giacomo: thank you for the patience of these months, for your dedication and consistency in your explanations. Thanks to prof. Vittorio Curri for the trust he has placed in me and for the great opportunities you have given me in recent months.

Thanks to my academic tutors, Gloria Vuagnin, Matteo Colantonio and Paolo Bolletta for their esteem in me granting me carte blanche during my project but always dedicated to showing me the route.

Thanks to all my friends, to those who had a word for me at any time of the day or night, to those who have been behind my paranoia and my anxieties.

Thanks to my family, my uncles, my cousins, and especially my parents for being the constant guide throughout my life.

Thanks to my grandparents, who illuminate my path every day.

Renato

Table of Contents

List of Tables

List of Figures

Acronyms

AM Amplitude Modulation

API Application Programming Interface

ASE Amplified Spontaneous Emission

ASK Amplitude Shift Keying

AWGN Additive White Gaussian Noise

BER Bit Error Rate

DT Digital Twin

FDM Frequency Division Multiplexing

FM Frequency Modulation

FSK Frequency Shift Keying

FTTC Fiber To The Cabinet

FTTH Fiber To The Home

GNPy Gaussian Noise simulation in Python

gRPC gRPC Remote Procedure Calls

GSNR Generalized Signal-to-Noise Ratio

HTTP HyperText Transfer Protocol

IETF Internet Engineering Task Force

ILA In Line Amplifier

IMDD Intensity-Modulation Direct Detection

IP Internet Protocol

LOGO Local Optimization Global Optimization

L-PCE Lightpath Computation Engine

MEMS Microelectromechanical system

NBI NorthBound Interface

NLI Non Linear Interference

NSI Network Slice Instance

OLC Optical Line Controller

OLS Optical Line System

ONC Optical Network Controller

ONF Open Networking Foundation

ONOS Open Network Operating System

OOK On Off Keying

OOPT PSE Open Optical and Packet Transport Physical Simulation Environment

OOTNC Open Optical Transport Network Controller

OSaaS Optical Spectrum as a Service

OSGi Open Service Gateway initiative

OSI Open Systems Interconnection

OTDR Optical Time Domain Reflectometer

PM Phase Modulation

PSK Phase Shift Keying

QoT Quality of Transmission

QoT-E Quality of Transmission Estimator

QPSK Quadrature PSK

ROADM Reconfigurable Optical Add-Drop Multiplexer

RPC Remote Procedure Call

RSWA Routing and Spectrum Wavelengths Allocation

SDN Software Defined Network

SNR Signal to Noise Ratio

SSH Secure SHell

TAPI Transport API

TDM Time Division Multiplexing

TIP Telecom Infra Project

WDM Wavelength Division Multiplexing

YANG Yet Another Next Generation

Chapter 1

Motivation and Goals

The continuous growth of internet traffic demand forced optical networks to move through new communication and management technologies. Due this shift towards open and disaggregated optical networks, many researchers and telcos were drawn to the potential benefits of this approach.

However, these technologies have no more then 10 yeas, so, despite the born of many consortia aiming at standardization, there are still many areas of development.

In the view of the above, this work aims both to provide a global knowledge of open and disaggregated optical networks and to demonstrate their potential. With this purpose, a custom orchestrator framework has been developed. Within this orchestrator, a QoT-E software will be integrated. QoT-e potential will be tested within an hard failure recovery use chase demonstrating how automatic traffic recovery is be feasible thanks to the disaggregation and standard interfaces.

The manuscript is divided into three main parts:

1. Firstly, an introduction is provided to the world of optical networks and signals, describing the main components and the SDN approach. This part includes "Chapter 2 - Fundamental Concepts" chapter.

2. Then, the SDN approach is applied to optical networks from a theoretical point of view in order to achieve optical network virtualization. This part includes "Chapter 3 - Network Architecture" chapter.

3. Finally, in this perspective, the structure of the developed software framework is presented and obtained results are discussed providing also possible future developments. This part includes Chapter 4 - Open Optical Transport Network Controller", "Chapter 5 - Results" and "Chapter 6 - Future applications and developments" chapters.

Chapter 2

Fundamental Concepts

According to an ancient Greek myth, Pheidippides – a Greek messenger – had to run from Marathon to Athens to deliver news of the victory of the battle of Marathon, and then collapsed and died. If he had had a modern smartphone connected to a 5G network, it would have taken a few milliseconds to send the same message, and it would have made much less effort. Unluckily, this myth is set in 490 BC and the era of electrical communication began in 1830 through the advent of telegraphy [1].

In this chapter, a brief introduction is outlined regarding the evolution of the networking. Then, the world of optical networking is explored, conducting a complete overview of the main components and their role within an optical network.

2.1 Historical Overview

Communication has always been crucial in human life, since earliest times the human beings felt the need to communicate remotely, before with rudimentary tools, as smoke signals, and then with increasingly complex instruments. So long as human progress is based on the exchange of information, data networks are the basis of today's society.

In the 1870s, the electric telephone was developed as a result of previous work with harmonic telegraphs. As the first industrial services were introduced, telephone technology advanced swiftly, and by the middle of the 1880s, telephone exchanges had been established in all of the country's main cities. The largest network based on electrical communication technology was the switching network developed everywhere in the world starting from the end of the XIX century [2]. This network allowed every telephone connected to the network to be directly communicate with the others by means of dedicated lines manually set using switching plugs. This infrastructure is still exploited to exchange data information.

During the last decades of the XX century, communication moved towards switching data networks. Thanks to the introduction of the network switches (Fig. 2.1), communication is no more performed through a dedicated channel, but, data can go from any source to any destination without a dedicated channel. A computer network utilizing exclusively network switches is known as a completely **switched network** [3]. For this purpose, switching nodes are network elements capable to route the traffic to the next node until they reach the end device without concerning with content of data.

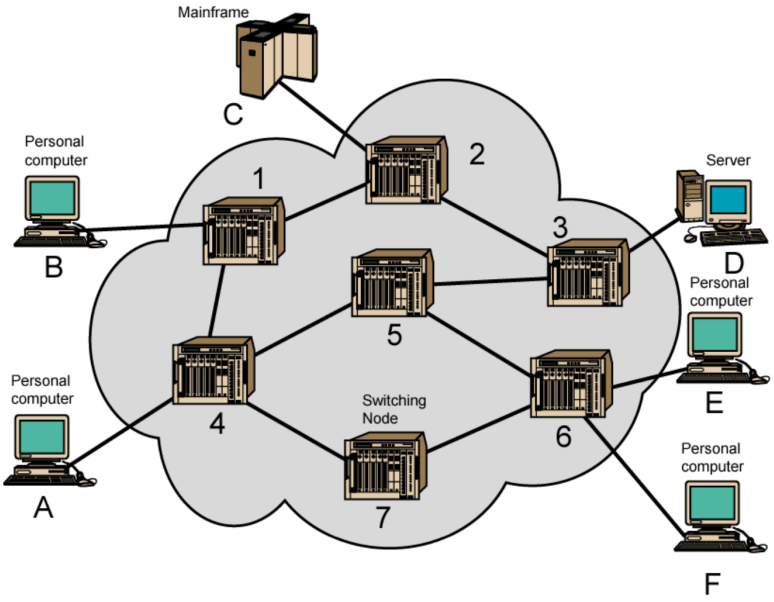


Figure 2.1: Network topology with switching nodes.

Connections can be established through two main techniques:

- **Circuit switching**, implements a dedicated connection between two network nodes. This solution is rarely used at upper layer [4] due to several drawbacks:

  1. **Inefficiency**: once the connection is established between two end points, all capacity is dedicated to that single connection. If no data are transferred, all the capacity can be wasted.
  2. **Delay**: the amount of time required to establish a single connection would be too high compared to the expected quality of service.

  Although this option is inefficient, it remains a common method to implement point-to-point connections at the physical level, so long as dedicated connection is needed to transport information in electromagnetic field.

- **Packet switching**, implemented thanks to the introduction of analog-to-digital information transport switches. Commonly used above the physical layer, the original transmitted data can be re-built at the receiver since the information can be divided in different parts, and each one can be independently routed inside the network. The role of each switching node is to store and forward each received packet. The main advantage is that the single line connecting two nodes can be used to transport packets related to different couple of source-destination nodes (Fig. 2.2).

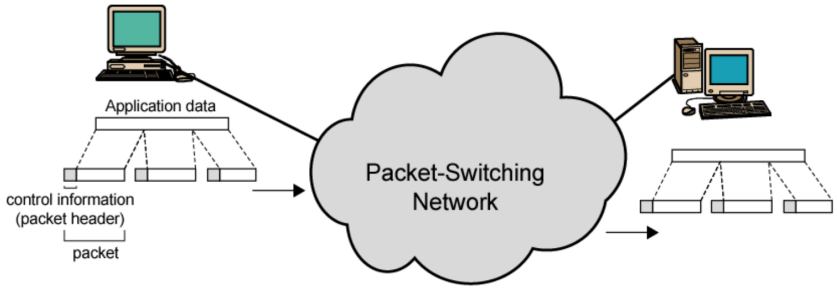


Figure 2.2: Information are divided into packet and devices are not aware of the network topology.

Since its adoption, optical networks has always provided sufficient bandwidth to satisfy all the requests.
So, up until the early 2000s, optical networks were static, with no need to achieve greater capacity.

2.1.1 Software Defined Networking

Starting from 2010, the performance growth of optical networks could not keep up with the demand for internet data traffic (fig. 2.3). Switching to the use of coherent optical technologies at the expense of IMDD [6], a better exploitation of the network was possible. However, the optical network infrastructure is still underutilized (Fig. 2.4). To increase the exploitation of the infrastructure, the solution that has been pursued was in the application of a software defined networking (SDN) [7] approach at the physical layer, transforming the transport in a virtualized function controlling and adapting the physical layer to the request coming from the application layer. Thus, referring to the optical connection transparent matrix in Fig. 2.5, the scenario has been completely modified, reaching a full colored matrix.

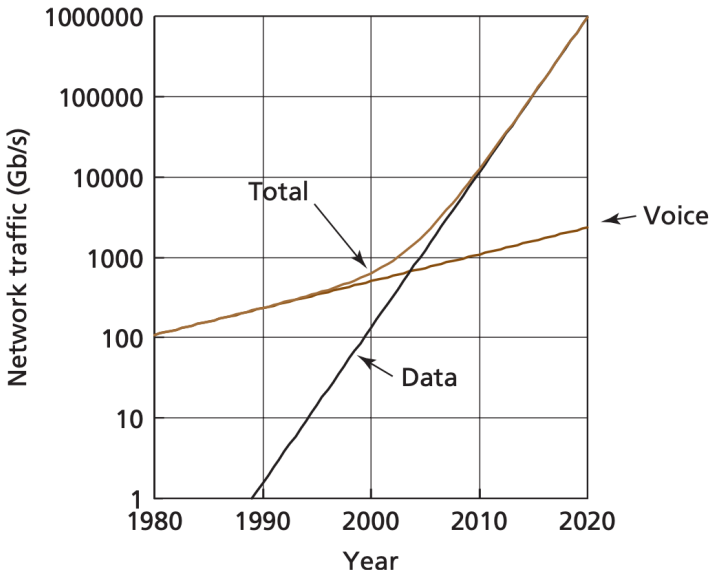


Figure 2.3: Increase of network traffic in north America during the last decades [5].

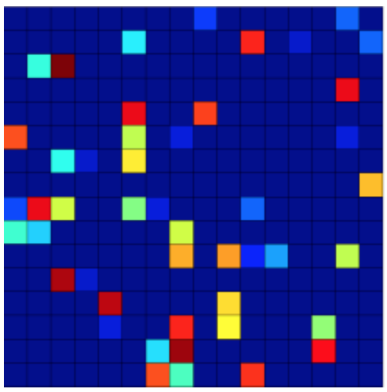


Figure 2.4: Optical connection transparent matrix before 2010. The blue squares show how the network was very underutilized.

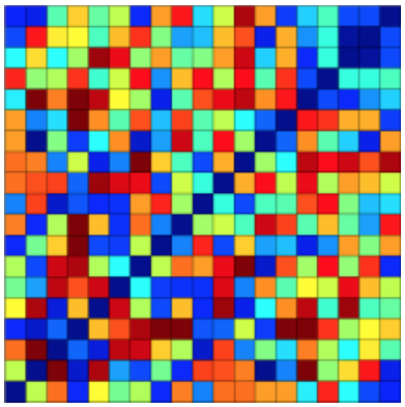


Figure 2.5: Optical connection transparent matrix after SDN introduction. The colors are the potential capacity deployable in a source-destination couple transparent connection, changes depending on how we deploy the traffic in the physical layer.

This means that the orchestration of the optical network can bring to the maximization of its capacity, avoiding the operators to install additional new optical fiber cables. However, the actual networks are made-up of devices whose

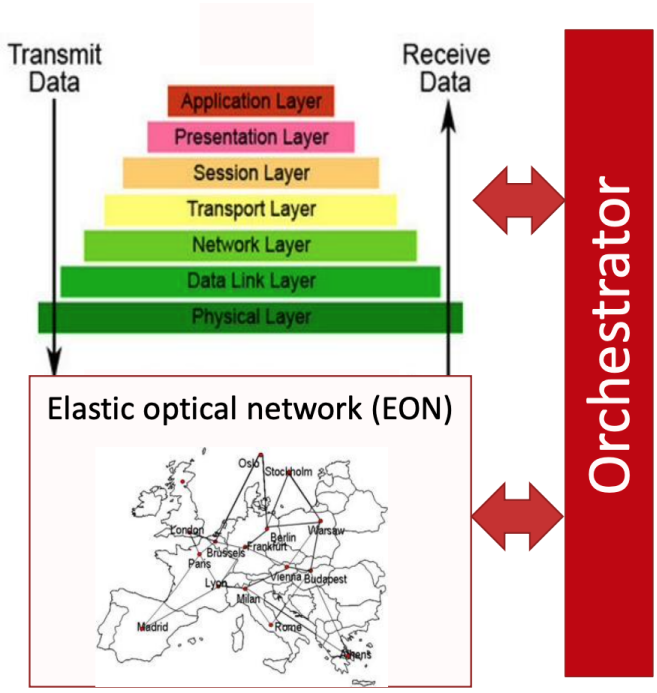


Figure 2.6: Orchestration techniques applied to European network.

implementations are closed software solutions, with all the problems that this approach brings [8, 9, 10].

2.2 Open Optical Networks

Nowadays, mostly on the Internet infrastructure is covered by transparent optical infrastructure. Upon these, many degree of openness are possible [11]:

- **Aggregated network**: closest solution where all the network is managed as a single entity. All the software and hardware components are closed, proprietary and from a single vendor. This solution does not provide space for openness(Fig.2.7).

- **Partly disaggregated network**: different segment of the network are still managed in a closed way, but the different pieces of the network are build upon different operator devices.

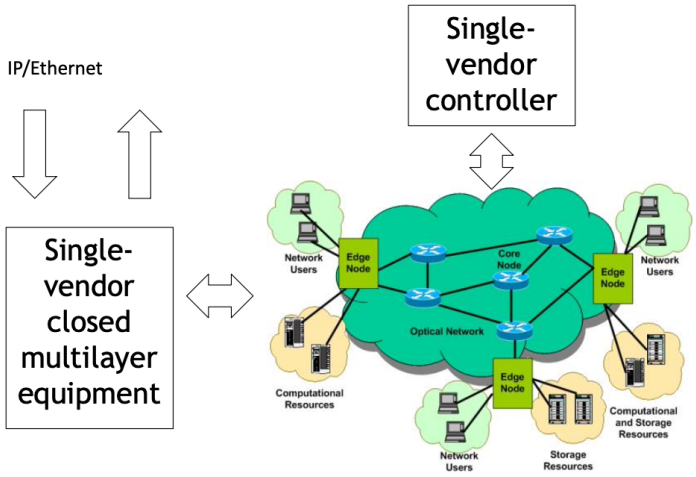


Figure 2.7: Aggregated network topology with closed software interaction.

- **Fully disaggregated network**: in this scenario, each element of the network can be from different vendors and each network element can be abstracted inside the SDN controller using standard model and controller through standard protocols (Fig.2.8).

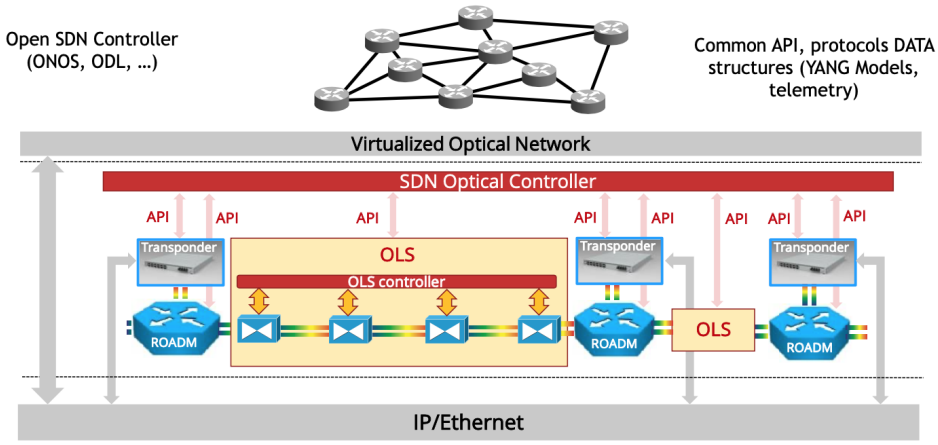


Figure 2.8: Disaggregated network example, showing ROADMs, Transponders and OLSs interfaces.

The implementation of open and disaggregated solutions allows to establish a vendor-neutral communication between the control system and the network devices, implying easier management of the infrastructure, also in terms of modernization and updating, and with a wider margin of maneuver.

An optical signal spreads across all the network infrastructure, but, depending on **network transparency**, many configurations are possible [12]:

- **Transparent Optical Network**: optical network where the signal is propagated without any regeneration from any source to any destination. In this scenario, only transparent optical circuit are deployed (Fig. 2.9). The largest part of the core networks are transparent.

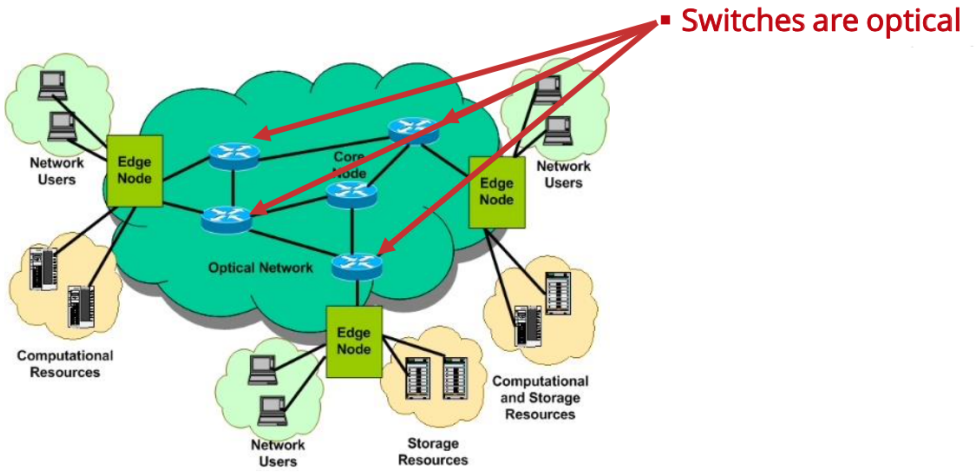


Figure 2.9: Optical nodes within an optical network are in charge to rout traffic. Connections between different nodes are transparent if there is no electrical conversion along the path.

- **Translucent Optical Network**: optical network in which the signal of a connection is regenerated in at least one intermediate node, passing from the optical domain to the electrical one and vice versa to avoid excessive degradation of the signal.
- **Opaque Optical Network**: extreme case in which no transparent switch are present and in each node re-generation of the optical signal is performed. Due the huge regeneration, the power consumption is relevant.

2.3 Optical Network Elements

An optical network is a transparent infrastructure with switching node connected by optical line systems, with the possibility to add and drop optical data traffic at edge nodes.

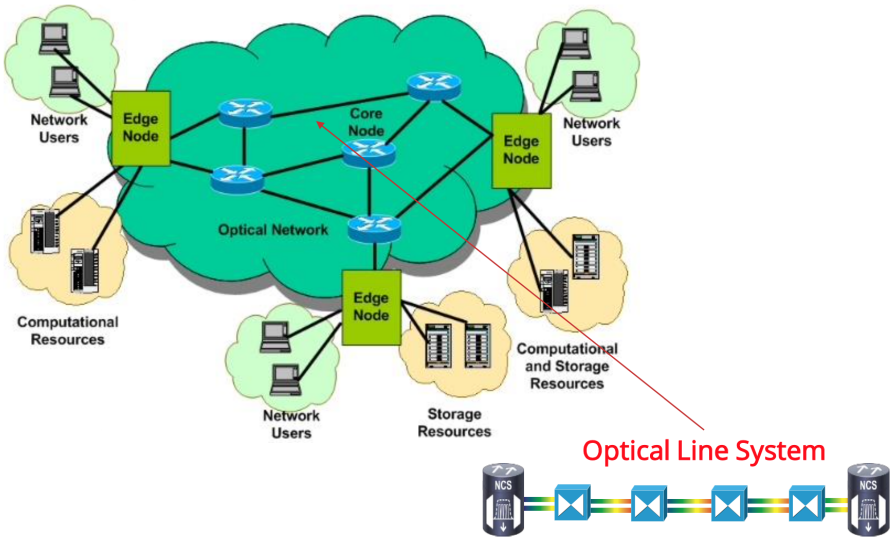


Figure 2.10: Between each optical node, signal amplification is exploited thanks to optical amplifiers. A line is composed from fiber spans and amplifiers, making an Optical Line System.

2.3.1 Optical Fiber

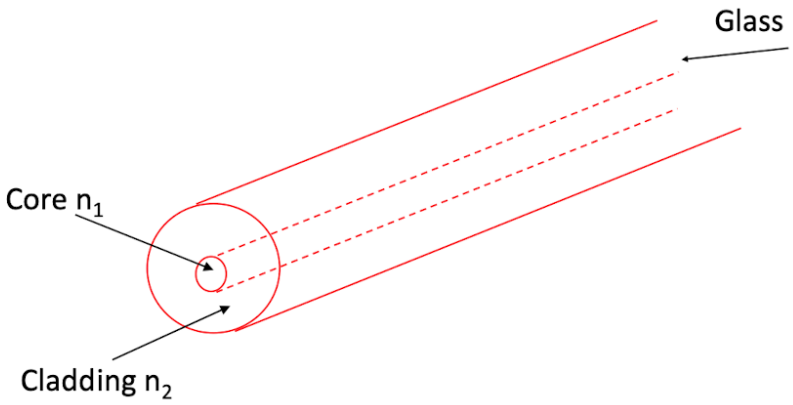


Figure 2.11: The optical fiber inside is made with two components having different refractive index in order to obtain total refraction.

An optical fiber cable is a glass pipe composed by two parts with different refractive index called core and cladding (n_1 and n_2 in Fig. 2.11). The peculiar physical phenomenon in optical fibers exploited in optical communications is the

total internal reflection, which is obtained by making a light beam collimate inside the core with an angle of incidence below a critical threshold, θ_c, described by the Snell's law (acceptance angle of optical fibre, Eq. 2.1), guiding the light through the fiber.

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1} \tag{2.1}$$

$$\theta_c = \arcsin\frac{n_2}{n_2} \tag{2.2}$$

The splicing operation allows to join two pieces of fiber both to repair a cable from a cut or to increase the length of a span avoiding the use of mechanical connectors. During the fiber-to-fiber coupling, the axial alignment represents a fundamental action, that should be realized with a precision higher than the core diameter, of the order of few µm.

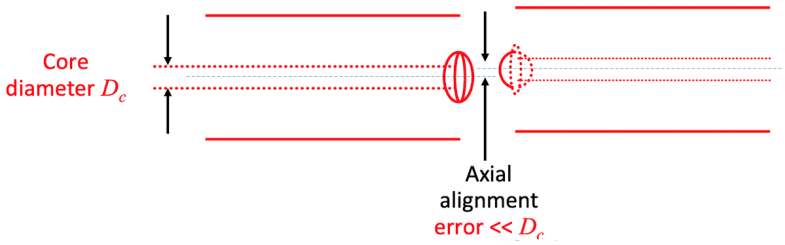


Figure 2.12: The image represents the fundamental quantities that come to attention when performing a splice between two fibers

2.3.2 Optical Amplifier

Optical amplification is a physical quantum mechanism where an energy transfer takes place from a laser pumping optical power to a propagating optical signal [13]. This phenomenon is enhanced properly doping the fiber glass with erbium, from which the erbium-doped fiber amplifiers (EDFAs) take name. Providing the correct wavelength to the EDFA's core containg erbium ions, an amplification of the input signal is achieved. The amount of the gain depends on the pumping scheme and from the other atoms within the core. The main drawback of this solution is the presence of noise generated by spontaneous emission adding noise to the hole signal [14](Fig. 2.13).

An amplifier used for transmission and for networking can be summarized as a **gain** and additive **noise** (Fig. 2.14).

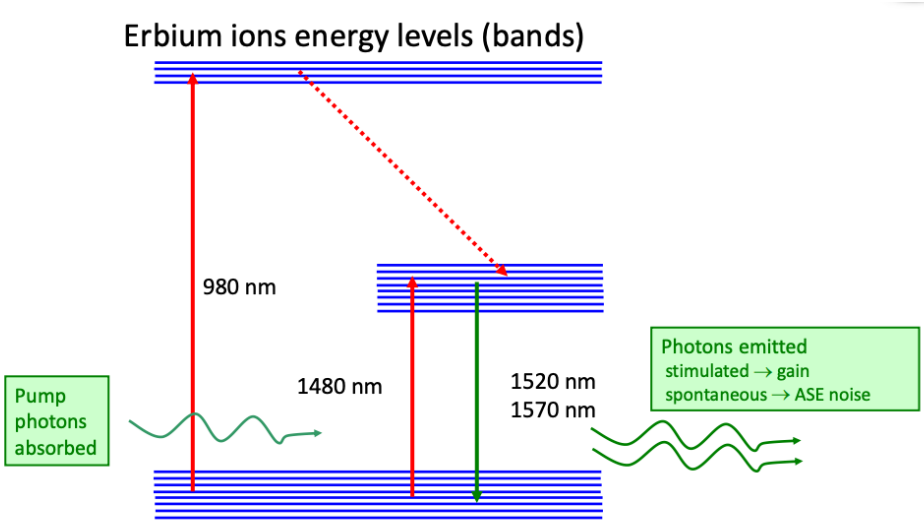


Figure 2.13: EDFA the image represents the fundamental quantities that come to attention when performing a splice between two fibers.

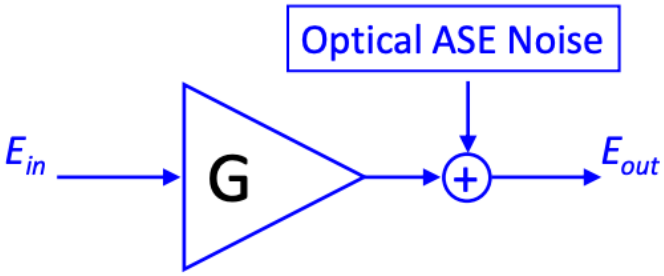


Figure 2.14: An optical amplifier can be modeled through two fundamental quantities, which are the optical gain and the introduced ASE noise.

2.3.3 Optical Line System

An optical line system (OLS) is a optical transparent link connecting two adjacent nodes. It is usually bidirectional, consisting of three main components:

- Fiber spans: with a dedicated fiber for each direction.
- In-line optical amplifiers: each ILA is made of two amplifiers, one for each direction.
- Booster and/or pre-amplifier: amplifiers placed at the output/input of switching nodes. Typically, boosters and pre-amplifiers are integrated within the switching node.

Figure 2.15: An optical line system composed by fiber spans and amplifiers.

2.3.4 ROADM: Reconfigurable Add & Drop Multiplexer

The ROADM is a key component of an optical node and it is in charge of implementing switching operations. In general, each node in an optical network has a ROADM for each degree of direction, able to route the traffic through the node according to a specified path or to add/drop the local traffic. The principle is that any input at any wavelength can be addressed to any output in a transparent way [15]. As shown in Fig. 2.16, each ROADM presents a WSS [16] at both input and output. In order to perform the switching operations, it is necessary to physically connect the ports of the WSSs among the different ROADMs.

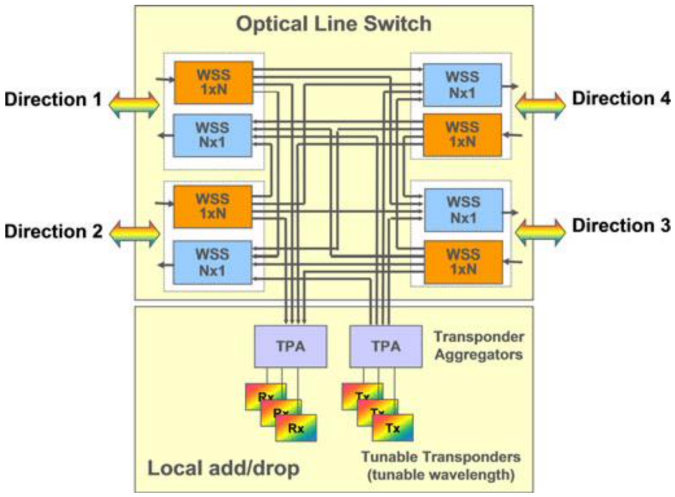


Figure 2.16: ROADM internally rely on N WSS in ingress and N WSS for the egress. Those are connected in full mesh. Then, thanks to local transponders the traffic can be added or dropped.

At the state of the art, the optical switching within the WSS is performed through micro electro-mechanical systems (MEMS) realized with liquid crystal on silicon (LCoS).

2.3.5 Transponder and Transceiver

The key component which allows to transmit and receive optical signals within the optical infrastructure is the **optical transceiver**. Historically, these devices ware a closed solution implemented by different vendors [17]. By the way, in the recent years, integration on chip and standardization brought to the birth of **plugable transceivers** [18] (Fig.2.17). A transponder is a device hosting a certain number

Figure 2.17: Picture showing different sizes of non-standard transceivers.

Figure 2.18: Standard transponder with standard slots, capable to handle a certain number of transceivers.

of pluggable transceiver (Fig. 2.18).

2.4 Optical Signal

As within optical networks signal is transmitted across nodes, signal cannot be propagated using electrical signals. This technique is acceptable only if transmitter are receiver have a direct connection. Another significant example concerns EDFA amplifiers working in the optical domain.

2.4.1 Optical Transmission Techniques

Within all the communication medium many transmission techniques have been studied [19]. In the optical signals, the possible modulation techniques that have been established in the course of history are essentially two.

Intensity Modulation Direct Detection

IMDD is the simplest modulation transmission techniques, used until 2010 but still employed at the edge node. It is based on OOK and 10 Gbps connection

standard. With this technique the light is switched on and off quickly, with a threshold beyond which the signal represents the high bit, the low bit otherwise.

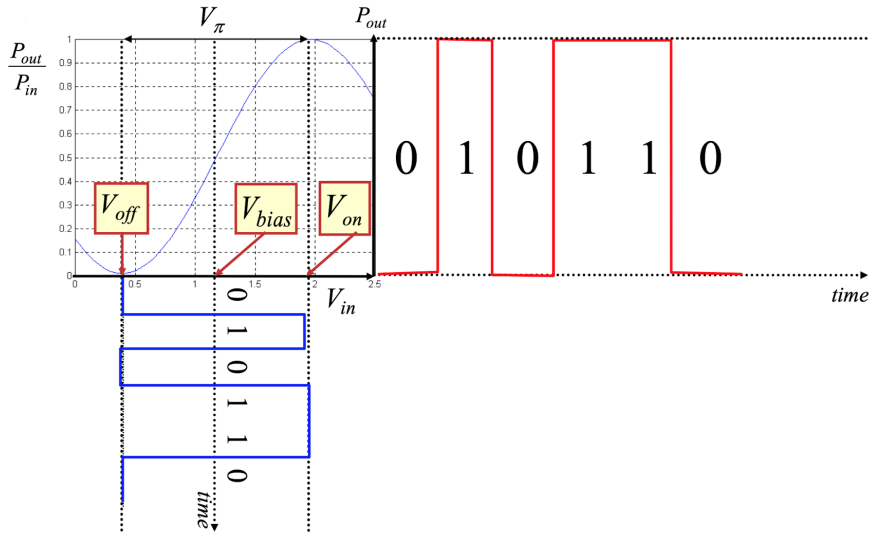


Figure 2.19: Mach Zehnder Intensity Modulation Direct Detection transceiver example. Signal is coded thank to a binary codification. Above a threshold voltage, IMDD codes a high bit, under this voltage low bit is coded.

Coherent Optical Technologies

IMDD was suitable until around 2007, when since the smartphone revolution, the demand for bandwidth to the internet has increased exponentially. Since that moment, investments in the optical network have increased, and today WDM is the standard for the core of the network and is also gaining ground within datacenters. With coherent optical, modulation format is multilevel modulation format. In the IMDD only amplitude is exploited to communication, conversely, coherent uses both amplitude and phase. This gives two degree to design modulation (Fig. 2.20).

2.4.2 Multiplexing Techniques

At the state of the art, a single signal can carry up to 400 Gbps, occupying tens of Giga Hertz in the spectrum. The next step is to use exploit the fiber in a larger spectral region. The main idea is to use the same technique employed in wireless communication: frequency division multiplexing (FDM). Because of historical reasons, the same technique applied to optical fiber communication is called **Wavelength Division Multiplexing** (WDM).

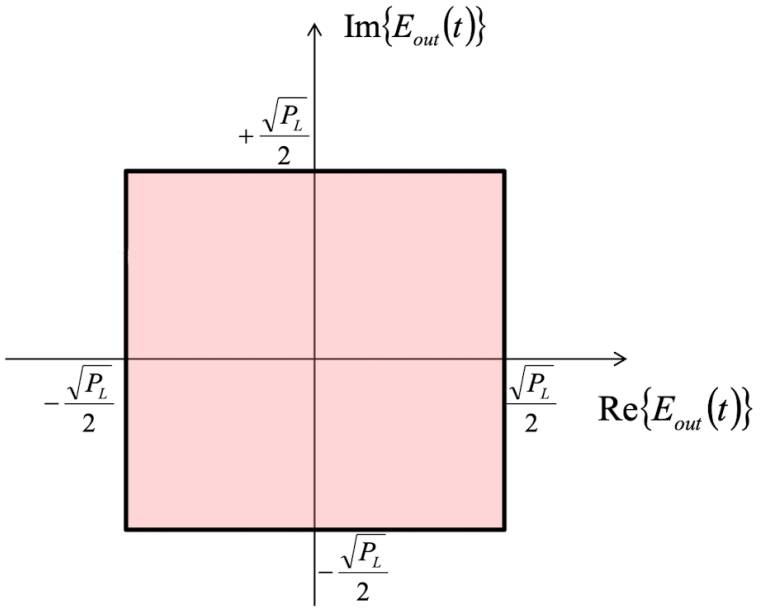


Figure 2.20: Thanks to WDM, a two dimensional modulation space can be exploited.

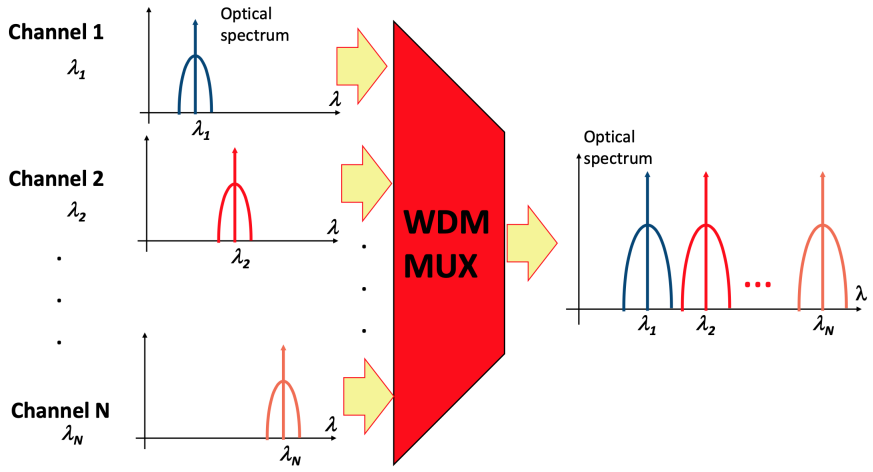


Figure 2.21: WDM gets multiple signal in input and is capable to multiplex all in a single fiber.

Another available technique is **Time Division Multiplexing (TDM)**. This is not engaged in the optical world, but it is exploited in the electrical domain. TDM is used to multiplex different tributary at low bit rate in the electrical domain,

adjusting the bit rate for optical communication.

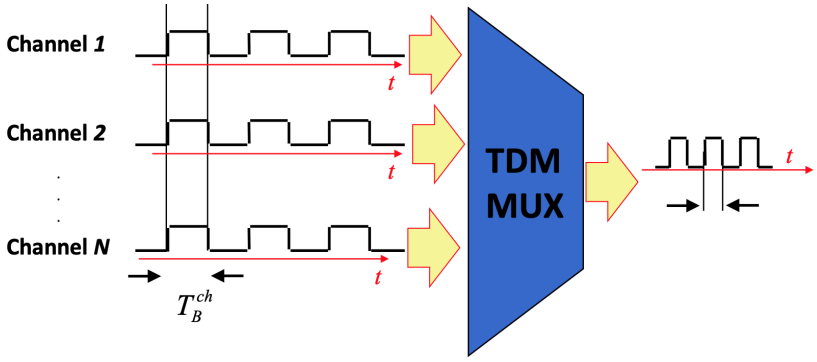


Figure 2.22: TDM is based on alternating the different signals with respect to time.

2.4.3 Modulation Format

Regardless of the medium used to carry the information, these can be coded in analog or digital. Analog solutions have been heavily used for multimedia purposes as they allow to have a faithful representation of the original content. On the contrary, as the digital coding rely on discretization it is more exploited for transmissions [20]. Inside a fiber, an optical analog signal wave can be described as:

$$\mathbf{E}(t) = \hat{\mathbf{e}}\, a \cos\left(\omega_{\mathbf{0}} t - \phi\right). \tag{2.3}$$

Depending on the element, many modulation are possible:

- **Amplitude Modulation (AM)** (a in 2.3);
- **Frequency Modulation (FM)** (ω in 2.3);
- **Phase Modulation (PM)** (ϕ in 2.3).

These techniques can be used even for digital modulation: **Amplitude-shift keying (ASK)**, **Frequency-shift keying (FSK)** and **Phase-shift keying (PSK)**, as shown in 2.23.

Starting from the first years of XXI century, a new approach exploiting both amplitude and phase modulation has been introduced [22]. To better understand this solution, Eq. 2.3 can be re-written as:

$$A = a\, e^{i\phi} \tag{2.4}$$

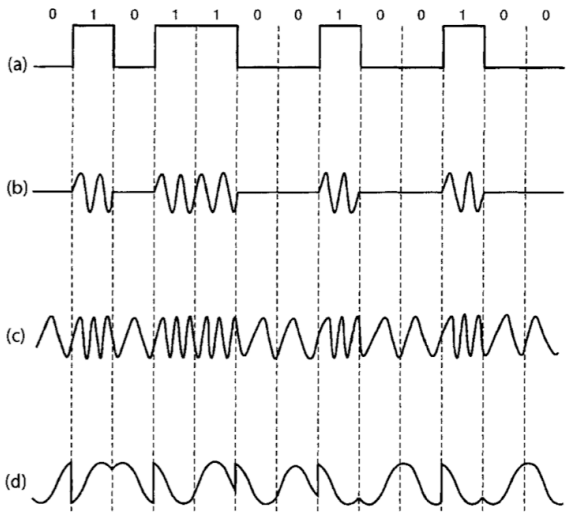


Figure 2.23: Shows amplitude modulation(b), frequency modulation(c), phase modulation(d) starting from a digital signal (a) [21].

Fig. 2.24 sums it up via constellation diagram [23]. The first two subfigures (a and b) show the ASK and the PSK modulations. Fig. 2.24-c shows the Quadrature-PSK (**QPSK**), a phase modulation using 4 possible values transmitting two bits. The Last example is the **multilevel QPSK**, carrying out a larger number of bits.

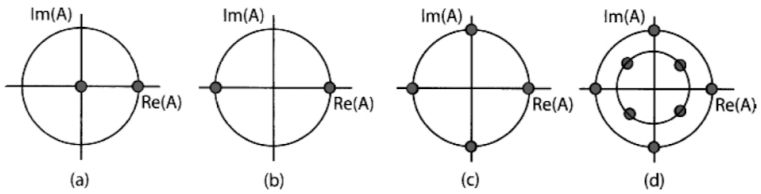


Figure 2.24: Constellation diagram for ASK(a), PSK(b), QPSK(c), multilevel-QPSK(d) [21].

Chapter 3

Network Architecture

Actually, optical networks are handles via a monolithic approach with a single closed controller. Open and disaggregated networks require a control structure. In this scenario, modularity is fundamental feature in order to keep separation of concerns. As in multi-vendor scenario each domain exposes a standard interface, thus, OSaaS [24] is one of the possible targets.

3.1 Software Defined Open Optical Networks

In traditional network approach, each network device is build upon three architectural planes [25]:

- **Control plane**: in charge of computing the local forwarding state. Usually, this operation is performed through distributed protocol implemented in each device. Usually this action happens exchanging some information about device's state (neighbor, connected links cost etc..).
- **Data plane**: processing and delivery of packets with local forwarding state. Data plane aim to compute a forwarding decision according to the packet header and the computed forwarding state.
- **Management plane**: in purpose of providing interfaces towards a human figure who must interact with the device in order to configure it or read telemetry values.

Although the network devices are equipped with a logical separation of the 3 planes, they are however equipped with an operating system with proprietary interfaces(fig. 3.1). This severely limits both a multivendor network and the ability to change the behavior of the devices themselves. In SDN approach the devices did not have to be compatible each other, they must be compliant only with the header

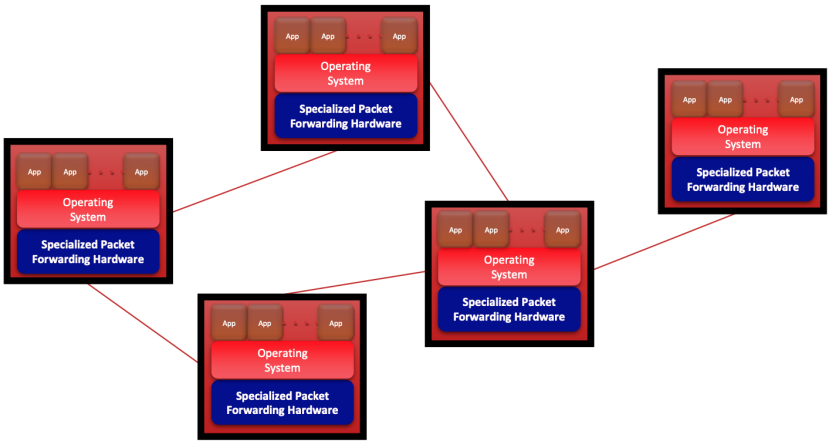


Figure 3.1: With traditional closed approach, each devices exchanges information with its neighbour. In this scenario, each device will make its own forwarding decision.

of the transported packet. Furthermore, a centralized solution aims to simplify the devices' architecture, moving control plane into a centralized controller. In this way, the network elements are only in charge to forward packet following rules pushed from the controller. Thus, to make upgrading and adding features easier, these devices need to be more software defined focused.

The concepts of SDN can be applied to the various levels that make up the OSI [4] stack, allowing its application also to the physical layer [26, 27]. Thanks to the transparent optical packet switching [28], the optical controller is moved from the network node's control plane to the the overall SDN controller. The latter approach is explained in fig. 3.2.
The data plane is kept within network device, who, will follows the configurations pushed from the control plane. The control plane, is moved into a centralized controller running a network operating system relying on network topology abstraction. Above this network abstraction, many behaviour can be implemented through application plane run on the network map abstraction.

Furthermore, SDN can solve several problems inherited from the traditional networking approach:

- **Network management**: troubleshooting a network issue is cumbersome operation and typically no automatically recovery solutions are implemented. An SDN and open approach will simplify failures detection and recovery.

- **Network evolution**: actually, if a network device has to be updated it or a is needed a new feature needs to be added direct connection is required [11]. If the logic is moved out from the device, this operation will be easier. Assuming that a feature needs to be included within the network, only the control plane into the centralized device has to be updated. Furthermore, this is not the main advantage, because, with a closed approach it is not possible to add a new feature in network device due the proprietary software. On the contrary, an open approach unlock the possibility to develop and install all the possible solution. Develop proprietary software is usually a solution feasible for medium-large network [29].

- **Network design**: different networks follow different philosophy in their design, this because no formal principles exist and each network manager choose different solution when a network has to be build.

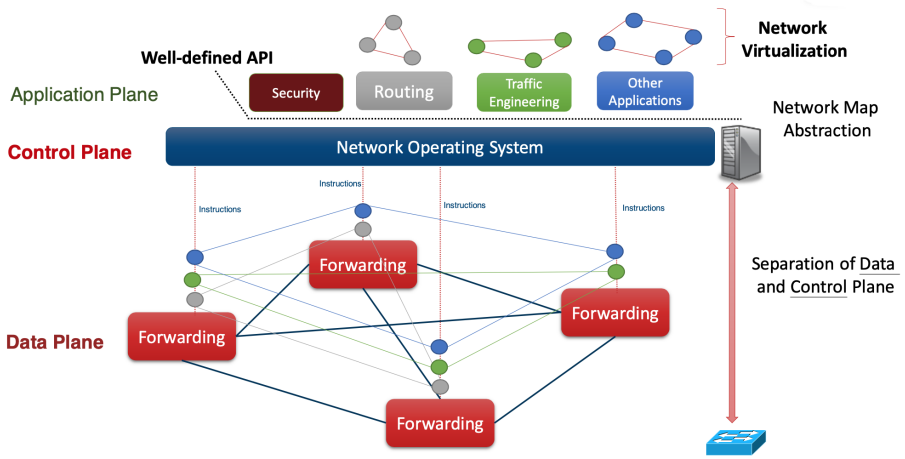


Figure 3.2: Software defined network schema. Devices keep a simple data plane exposing standard interfaces. Control plane is moved into a centralized controller above which it's possible to run multiple applications solutions.

If the control plane needs to be moved into the centralized controller, all the behaviours and the knowledge of the network has to be emulated. Primarily, full knowledge of network topology and installed equipment is needed, then, all the **network operations** must be emulated by providing a standard solution. Actually there aren't standard solution to control the hardware: each vendor has its own language. Also transport operations need to be virtualize, but, a standard solution is provided by OpenFlow [30].

Fig. 3.10 summarizes the modular approach spreading out within open optical networks.

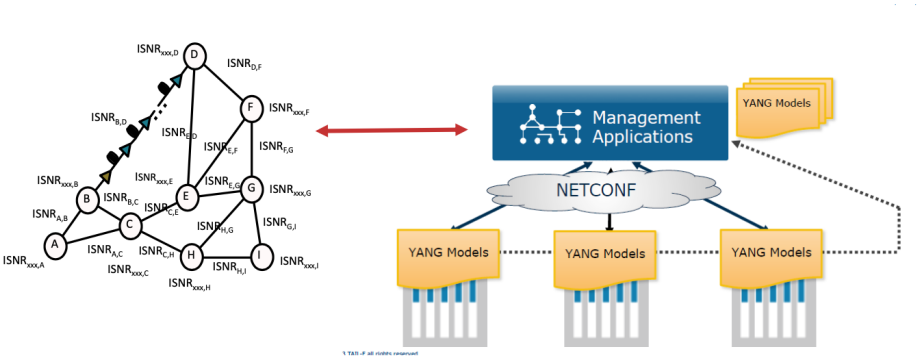


Figure 3.3: Network emulation is usually based on standard YANG models and NETCONF interfaces.

3.2 Optical Network Virtualization and Slicing

According to [31, 32] the demand for internet traffic is growing more and more under the pressure of smart working, cloud services, high resolution streaming etc. Thus, deploying dynamic optical infrastructures at high data rates that can serve these various application types, each with their unique access and network resource use patterns, is a major problem for network operators.
Basing on [31]: "A virtual optical network is a set of virtual optical nodes interconnected together that share a common administrative framework. Optical node virtualization is the creation of a virtual representation of an optical network node, based on an abstract model that is often achieved by partitioning or aggregation. Within a virtual optical network, virtual connectivity (virtual link) is defined as a connection between one port of a virtual network element to a port of another virtual network element."

Optical Network Virtualization is the killer feature for most current and future technologies:

- **Network Slicing**: is a basic requirement for modern FTTx and 5G connectivity and this is ongoing request from the operator to improve networking. Slicing is based on a shared infrastructure and a network slice instance (NSI) is a fully instantiated logical network that satisfies specified network requirements for a given service. It is made up of a collection of network functions and the resources necessary to deploy those (fig. 3.4).

- **Quality of Transmission Estimation**: cannot be achieved without network virtualization in order to build a network's digital twin and compute LP's GSNR estimation.

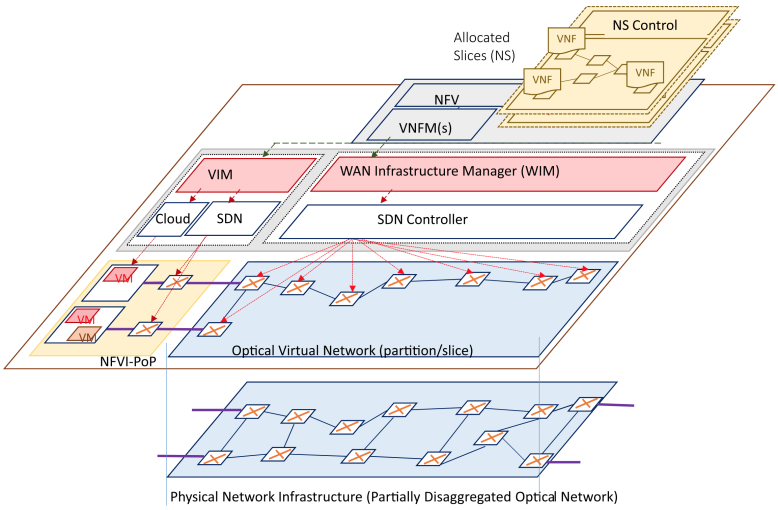


Figure 3.4: Network slicing over virtualized disaggregated optical network [31].

3.2.1 Terminal Device Emulation

One of the main emulation project is the OpenConfig project which goal is to define vendor-independent YANG data models [33]. This project covers multiple functionalities, and according to their official guide [34], Open Config can deal with:

- **Common data models**: representations of data that are consistent and coherent, created by users for vendor-independent administration in a wide range of networking use cases.
- **Streaming telemetry**: all devices based on OpenConfig principles, streaming telemetry is a subscription-based strategy for effectively and precisely monitoring network devices [35, 36]. This technique is much more secure, efficient and reliable than SNMP [37].
- **Management protocols**: based on gRPC, a contemporary, secure RPC framework designed for distributed services, device management and control protocols.
- **Testing and compliance**: automated testing of OpenConfig implementations for compliance that is independent of vendor.

3.2.2 ROADM Device Emulation

ROADM Device Emulation is achieved thanks to OpenROADM project [38]. One of the main objectives, is to be able to configure ROADM through centralized SDN

controlled, instead of needing human intervention. The YANG device model itself

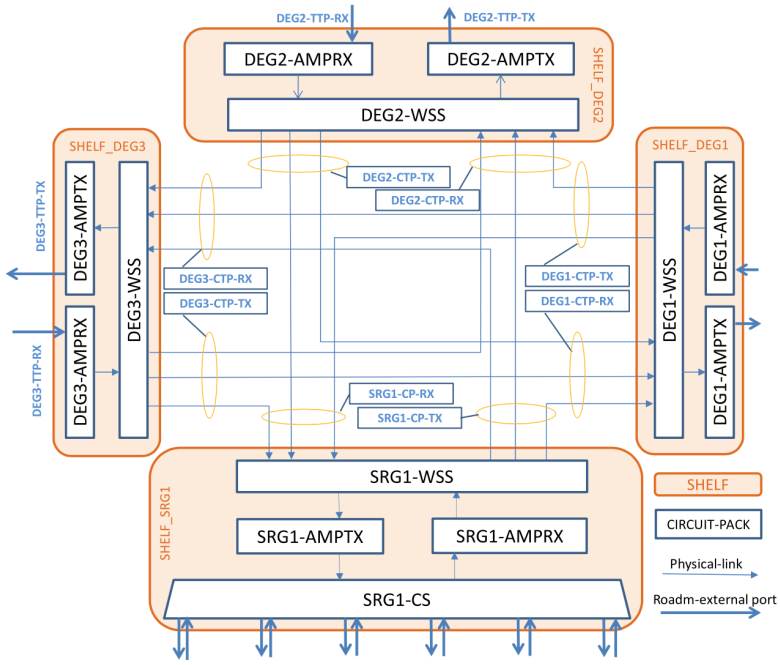


Figure 3.5: Logical view of YANG model OpenROADM v2.2 device [39].

is fairly detailed. On a macro level, it identifies a first section pertaining to the device information (shared language,node id, vendor's model,GPS location, etc.) then it is followed by a section that contains a list of circuit packs defining the physical design, such as ports and current racks and shelves
The use of openROADM and openConfig has been widely exploited [40].

3.2.3 Northbound Interface

The variety of controller interfaces necessitates the usage of specialized plugins, which makes extension challenging and expensive. A standard interface that can be used by many vendors and domains and has similar models is clearly needed to serve as a controller NBI (fig. 3.6).

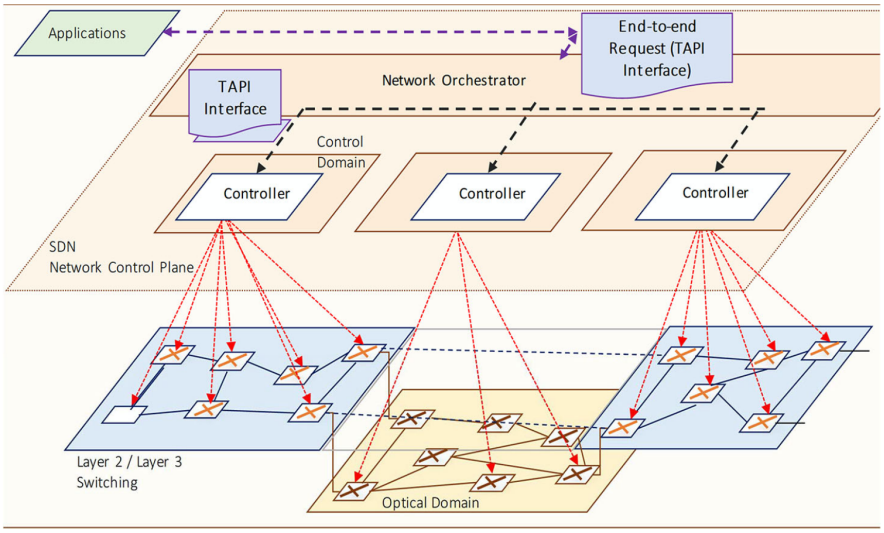


Figure 3.6: Example of NBI [39]. A standard interface is required both for a known external endpoint, and for a standard communication between controller and devices.

3.3 SNR-Based Network Abstraction

Transceivers used in dual polarization coherent optical transmission enable a fully flexible management if, the channels they propagate on is a dual polarization Additive white Gaussian noise [32].
A transceiver can be fully described in terms of the BER vs SNR curve (Fig.3.7) or by analytical expressions [41]:

$$\text{PM-BPSK:} \qquad BER = \frac{1}{2} erfc \sqrt{\frac{OSNR}{K_{pen}}} \tag{3.1}$$

$$\text{PM-QPSK:} \qquad BER = \frac{1}{2} erfc \sqrt{\frac{\frac{OSNR}{K_{pen}}}{2}} \tag{3.2}$$

$$\text{PM-8QAM:} \qquad BER = \frac{2}{3} erfc \sqrt{\frac{3}{14} \frac{OSNR}{K_{pen}}} \tag{3.3}$$

$$\text{PM-16QAM:} \qquad BER = \frac{3}{8} erfc \sqrt{\frac{1}{10} \frac{OSNR}{K_{pen}}} \tag{3.4}$$

Moreover, if the OSNR over a lightpath can be calculated, also the feasible modulation format (therefore the bit rate) over the same lightpath (Fig.3.8) can be

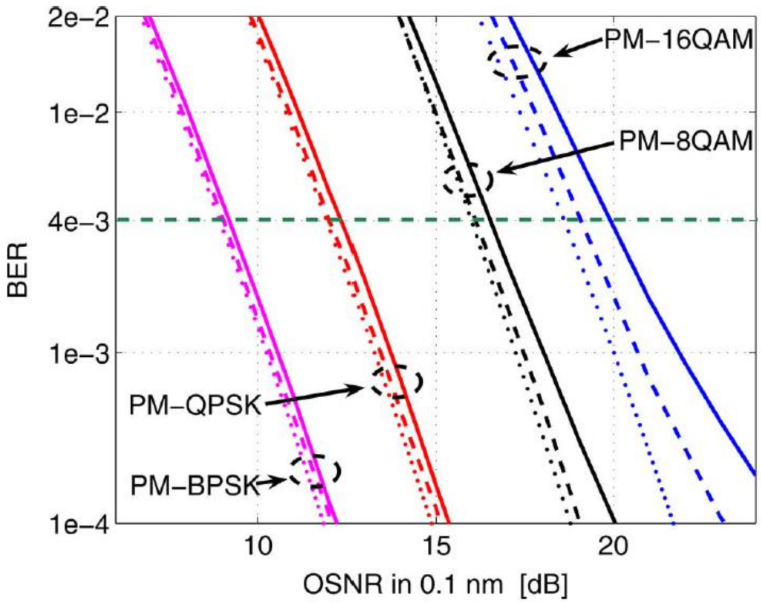


Figure 3.7: A transceiver BER vs SNR curve is capable to fully describe devices behaviour.

exploited. In order to compute the OSNR over a certain lightpath, a centralized controller owning all the topology is needed. Thus, network element should be equipped with standard northbound interface and standard model [32].

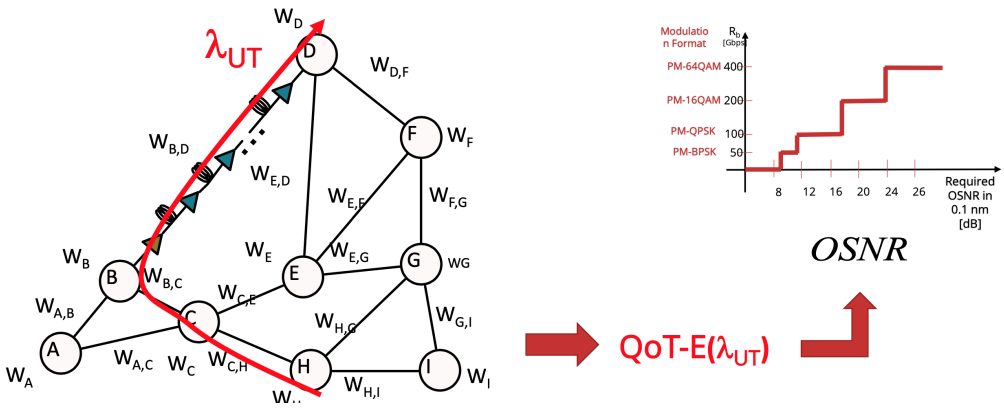


Figure 3.8: If in a network emulation it's possible to run a simulation capable to retrieved the overall GNSR from source to destination, relying on this simulation, it possible to identify to most suitable modulation format.

3.3.1 The Generalized SNR

In transparent optical network, each transparent connection from a source to a destination is called lightpath. In order to be considered *transparent* a lightpath requires wavelength continuity from source to destination. Each lightpath is a dual polarization AWGN, and each noise affecting the channel is composed by amplified spontaneous emission (ASE) and non liner interference (NLI). Because of this multiple source, the SNR metric for transparent lightpaths in an optical network is defined GSNR:

$$GSNR = \frac{P_{CUT}}{P_{ASE} + P_{NLI}} \cdot FP \tag{3.5}$$

where P_{CUT} is the power of the channel, P_{ASE} and P_{NLI} are respectively the accumulated ASE and NLI. FP is a factor ≤ 1 for the accumulated filtering penalty [42]. Noise is *accumulated* because a lightpath is a transparent connection from any source to any destination, so, if a crossed node can be represented as a gain A and a noise n, a connection from a transmitter and receiver is the cascade of the effect for each element (Fig. 3.9).

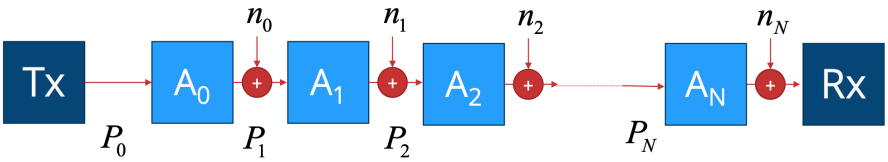


Figure 3.9: A lightpath crosses multiple device. If each devices can be described in terms of gain and noises, a connection suffers of all the amplification and noise introduced by devices going through.

If all the gain and noise are known, at the receiver side the GSNR can be written as:

$$GSNR = \frac{P_0 A_0 .. A_N}{P_{n0} A_1 .. A_N + P_{n1} A_2 .. A_N + .. + P_{nN}} \tag{3.6}$$

To accumulate the GSNR introduced with different cascade, it's convenient to use the inverse of GNSR:

$$ISNR = \frac{1}{GSNR} = \frac{P_{n0} A_1 .. A_N + P_{n1} A_2 .. A_N + .. + P_{nN}}{P_0 A_0 .. A_N} \tag{3.7}$$

$$P_{i+1} = P_i A_i \tag{3.8}$$

$$ISNR = \frac{1}{GSNR} = \frac{P_{n0}}{P_0 A_0} + \frac{P_{n1}}{P_1 A_1} + .. + \frac{P_{nN}}{P_N A_N} = \sum_{i=0}^{N} \frac{1}{GSNR_i} \tag{3.9}$$

Thus, if a model for every element crossed in the lightpath can be written, total GSNR can be summarized as single network element contribution:

$$GSNR = \frac{1}{\sum_{i=0}^{N} \frac{1}{GSNRi}} \tag{3.10}$$

So, in a optical layer if each network element can be abstracted with its GSNR impairment, the overall GSNR can the calculated over every path in the network. Thus, if in the virtual representation of the network (fig. 3.8), all the lightpaths' GNSR can be obtained, then, using a software capable to sum the single GSNR contribution, the overall GSNR can be transformed into feasible modulation format fully automatizing the physical layer. These kind of software are the so called **quality of transmission estimator**.

3.4 Open Optical Networks Components

An open and disaggregated optical network requires different optical data and control plane controllers. Unlike unique-vendor network, which management is handled through a single monolithic closed controller, openness and disaggregation requires new control solutions in line with this philosophy [43].

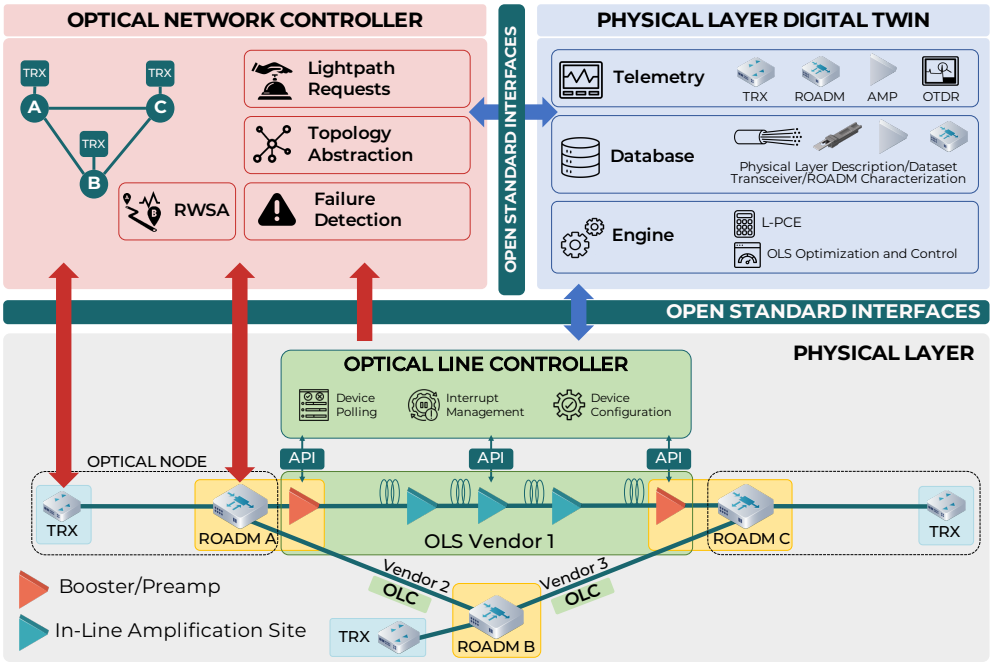


Figure 3.10: Open and disaggregated optical network architecture is composed by different controller for different components, connected through standard interfaces.

3.4.1 Optical Network Controller

In SDN optical network, advantages coming from the use of ONC are widely known [44]. ONC is the only component capable to handle the various network element because of its view across the whole network. Usually, ONC initiates and keeps a connection with all the network elements (transceivers, ROADMs, etc..). Thus, ONC is capable to build a topology abstraction. Furthermore, ONC will be in charge to provide this abstraction to those which request it.

As direct device communication offers to set devices behaviours, the optical controller will be in charge of routing and spectrum wavelengths allocation, to serve lightpath requests. This kind of request usually arrives from the upper layers (IP, ethernet) specifying source and destination nodes. Being the only one with a complete view of the topology, it can apply a series of routing algorithms to find the best path through the nodes of the network, selecting the same wavelength among all the node ensuring wavelength continuity. Relying on topology abstraction and direct connection with OLCs and devices, ONC can receive through open and standard interfaces, notifications about topology updates. This, allows to understand new possible nodes within the network, but also detect failures.

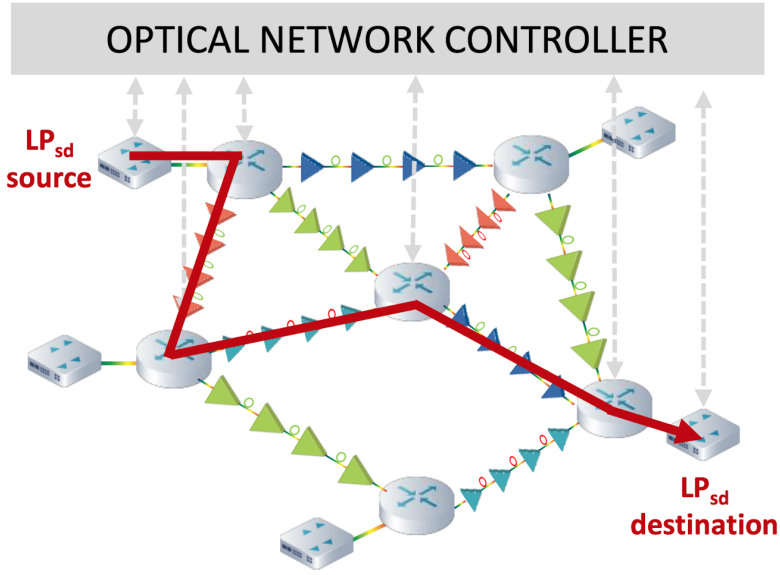


Figure 3.11: Optical network controller lightpath establishment [32].

3.4.2 Physical Layer Digital Twin

As introduced in the latter chapter, in order to estimate a the GSNR across a lightpath a QoT-E software in needed. The total amount of GSNR across a LP is closely related to ASE noise and NLI noise. However, it was demonstrated that the

GSNR's value it's more affected by ASE than from NLI [45]. In turn, ASE noise is strictly related to EDFAs working points crossed within the OLS [46]. Various machine learning (ML) approach has been tested in order to solve this problem [47, 48, 49]. Thus, in addition to QoT-e, DT is also in charge to compute the correct working point for the amplifiers in the network.

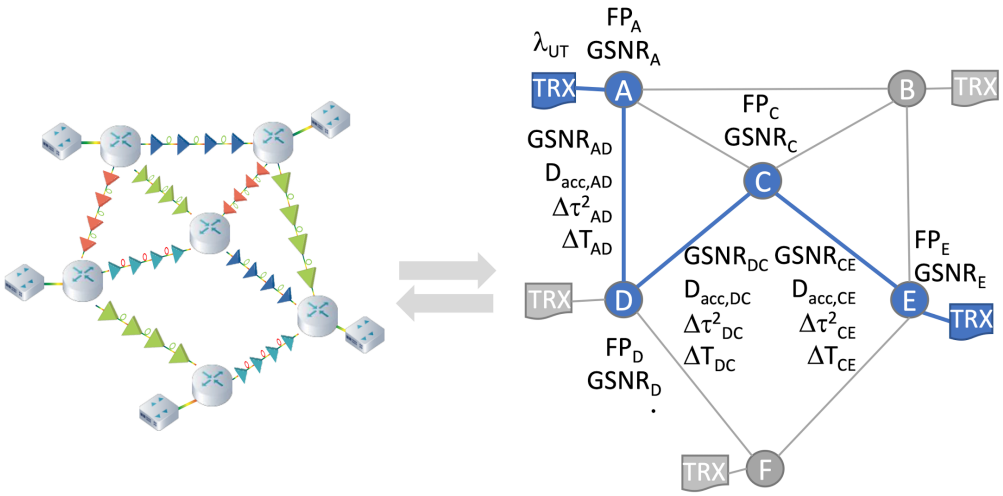


Figure 3.12: QoT-e's view on a network topology. A lightpath is connecting two end points [32]. The DT can characterize all the lines crossed by a LP and keeps its values.

3.4.3 Optical Line Controller

The OLC is a part of the control plane and it is in charge to set the working point for all the amplifiers. OLC is also the only component capable to intercept the interrupt from the amplifiers and forward these to the centralized controller. An optical line controller is usually a proprietary software handling an OLS. Since within an optical network more OLS are present, often more than OLC are present, even from different vendors. Because of this, between proprietary controller software and others control plane components, a standard interfaces are needed. An OLC communicates with ILAs, booster (BST) and pre-amplifier(PRE) through APIs. At boot phase, OLC has to start an OLS probing, composed of three steps:

- ILAs, BST and PRE have to be set with known settings.
- All necessary metrics must be measured.
- Devices in OLSs need to be watched through polling.

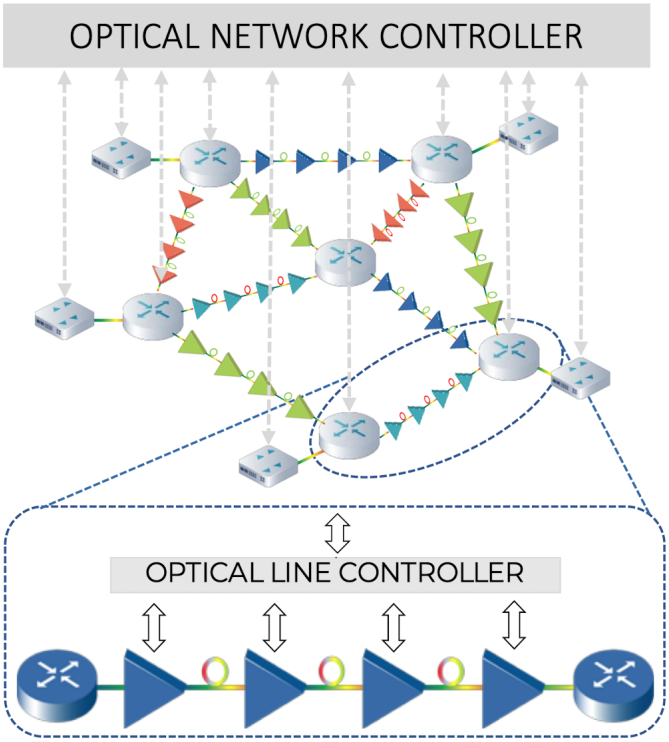


Figure 3.13: Each OLS is completely emulated as a single entity and exposed to the optical network controller though standard interface [32].

3.4.4 Interfaces

Interfaces are a fundamental component into the open optical architecture. With standard interfaces, each component needs to be compatible only with predefined interfaces. This also simplify the development, allowing to change component without needing to deal with new interfaces.

Transport-API (TAPI)

The Open Networking Foundation (ONF) created the standard API known as T-API (Transport API). A TAPI server have to:

- Obtain hardware and topology details from the network.
- Control connectivity services.

T-API has been designed to allow network operations across a multi-layer, multi-domain, multi-vendor transport infrastructure within a SDN context. By serving as an interface between controllers at various levels, T-API can be used to manage

network resources at various degrees of abstraction. An interface between a number of network domain controllers and an upper-level network orchestrator that serves as a multi-domain or hierarchical controller would be an illustration of a typical T-API configuration. The IETF RESTCONF protocol [50] specification is used

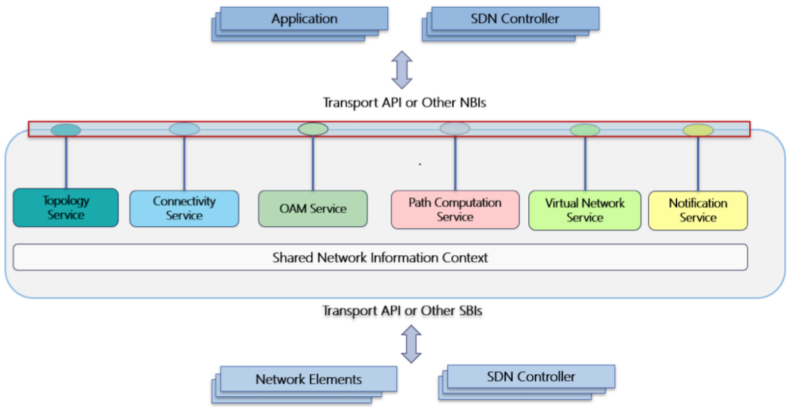


Figure 3.14: T-API structure offer multiple services that can be exploited from other actor in the network.

by the TAPI RESTCONF NBI RESTful web services interface when creating its interfaces. It rely use of the TAPI YANG data models [51], which are outlined in the YANG specification. The following IETF specifications are employed:

- RESTCONF Protocol RFC 8040.
- YANG Specification RFC 6020 [52].
- Hypertext Transfer Protocol HTTP 1.1 - RFCs 7230-7237 [53].

The following resources are part of the RESTCONF specification:

- **restconf/data (Data API)**: Create/Retrieve/Update/Delete (CRUD) based API for the entire data tree defined in the TAPI information model YANG models.
- **restconf/operations (Operations API)**: small number of operations defined as RPCs in the TAPI information model and YANG models make up an RPC-based API.
- **restconf/data/ietf-restconf-monitorinestconf-state/streams (Notifications API)**: RESTCONF notification protocol.

- **restconf/yang-library-version)**: The "ietf-yang-library" YANG module that this server has implemented is identified by this required leaf as having a revision date.
- **restconf/data/ietf-restconf-monitorin:restconf-state/capabilities**: leaf to notify the server's ability to support query parameters.

3.5 Local Optimization Global Optimization

The GSNR is a crucial metric for modelling optical fiber propagation performance, and is given by:

$$GSNR = \frac{P_{CUT}}{P_{ASE} + P_{NLI}} = \frac{P_{CUT}}{P_{ASE} + \eta P_{ch^3}} \tag{3.11}$$

Equation 3.11 has been plotted in fig. 3.15. As GSNR depend on the channel power, the amplifier gains have to the set carefully. It is possible to notice how the equation has a maximum, which can be calculated. Furthermore, the point is very robust because it is located in a curve that is not very sensitive to uncertainties. Thus, by choosing carefully the input power, GSNR can be maximized. Amplifiers are the elements in charge of the power, because the output power is the result of the input power multiplied for the gain. OLCs are in duty to set the correct working point.

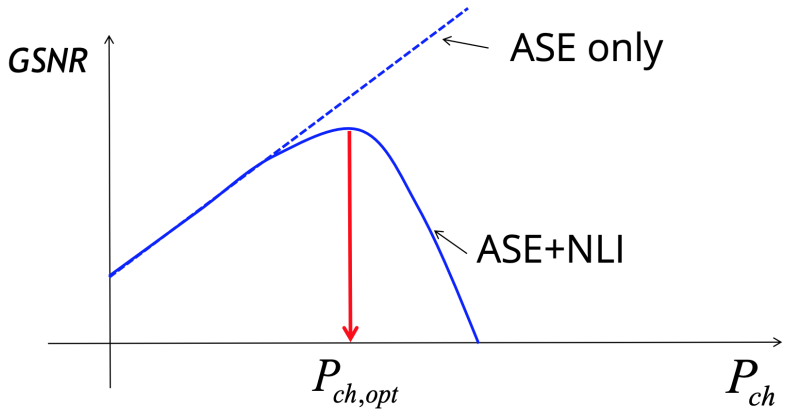


Figure 3.15: Graphical representation of the GSNR behavior. The chart describes the difference between a linear environment, where power can be as be as possible without effect on the GSNR, and the non linear environment where the NLI noise will decreases the GSNR proportionally with channel power.

In order to consider a generic OLS, in fig. 3.16 an OLS with N amplifier is shown.

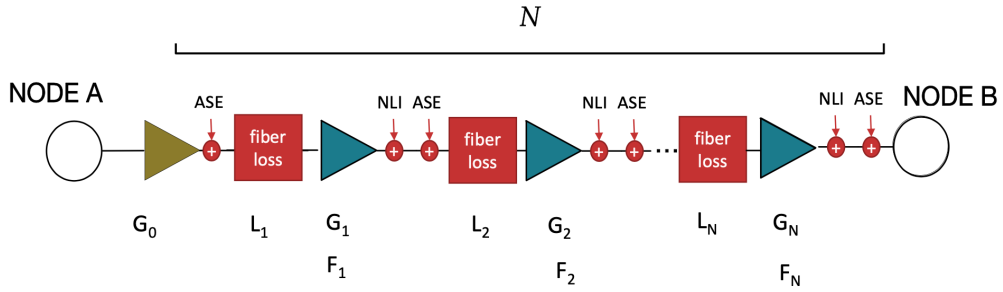


Figure 3.16: Optical line system with ILAs, BST and PRE. Each component is fully characterized in terms of gain, ASE and NLI.

Regarding a transmission between node A and node B, GSNR degradation can be written as:

$$GSNR_{AB} = \frac{P_{out,A} G_o L_1 G_1 \dots L_N G_N}{P_{ASE,0} L_1 G_1 \dots L_N G_N + (P_{ASE,1} + P_{NLI,1}) L_2 G_2 \dots L_N G_N + \dots + (P_{ASE,N} + P_{NLI,N})}$$

Maximize a GSNR means to minimize the opposite of GSNR received ad node B, who can be written as:

$$\frac{1}{GSNR_{AB}} = \frac{P_{ASE,0} L_1 G_1 \dots L_N G_N + (P_{ASE,1} + P_{NLI,1}) L_2 G_2 \dots L_N G_N}{P_{out,A} G_o L_1 G_1 \dots L_N G_N} + \dots + \frac{P_{ASE,N} + P_{NLI,N}}{P_{out,A} G_o L_1 G_1 \dots L_N G_N}$$

The different contribution can be separated:

$$\frac{1}{GSNR_{AB}} = \frac{P_{ASE,0} L_1 G_1 \dots L_N G_N}{P_{\text{out },A} G_o L_1 G_1 \dots L_N G_N} + \frac{(P_{ASE,1} + P_{NL,1}) L_2 G_2 \dots L_N G_N}{P_{\text{out },A} G_o L_1 G_1 \dots L_N G_N} + \dots + \frac{(P_{ASE,N} + P_{NLL,N})}{P_{\text{out },A} G_o L_1 G_1 \dots L_N G_N} \tag{3.12}$$

Which can be simplified as:

$$\frac{1}{GSNR_{AB}} = \frac{P_{ASE,0}}{P_{\text{out },A} G_o} + \frac{(P_{ASE,1} + P_{NLL,1})}{P_{\text{out },A} G_o L_1 G_1} + .. + \dots + \frac{(P_{ASE,N} + P_{NLL,N})}{P_{\text{out },A} G_o L_1 G_1 \dots L_N G_N} \tag{3.13}$$

Now, the inverse of GSNR is divided into the contributions of amplifier and fiber span.

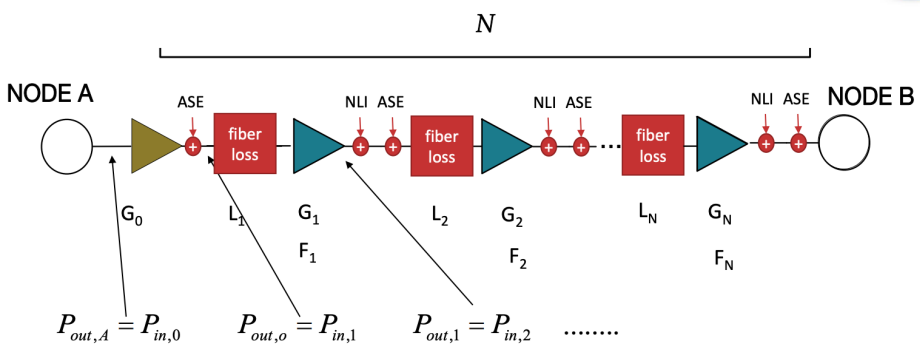


Figure 3.17: OLS portraying each amplifier's output power as the input of following one.

OLS is capable to set the gain of amplifiers, so the equation can be rewritten as the input power of each fiber span:

$$\frac{1}{G_{SNR_{AB}}} = \frac{P_{ASE,0}}{P_{\text{out },A} G_o} + \frac{(P_{ASE,1} + P_{NLI,1})}{P_{in,1} L_1 G_1} + \ldots . + \frac{(P_{ASE,N} + P_{NLI,N})}{P_{in,N} L_N G_N} \tag{3.14}$$

The model of ASE noise is described by:

$$P_{ASE,i} = P_{base} \left(G_i - 1\right) F_i \simeq P_{base} G_i F_i \quad P_{base} = h f_0 B_n \tag{3.15}$$

Furthermore the NLI noise is described by:

$$P_{NLI,i} = \eta_{NLL,i} P_{in,i}^3 L_i G_i B_n \tag{3.16}$$

These noise power description can be substituted within the inverse of GSNR:

$$\frac{1}{GSNR_{AB}} = \frac{G_o P_{\text{bas}}\, F}{P_{\text{out}}\, G_o} + \frac{G_1 \left(P_{\text{base}}\, F_1 + L_1 \eta_{NLL,1} B_n P_{in,1}^3\right)}{P_{inn},\, L_1 G_1} + \ldots + \frac{G_N \left(P_{\text{base}}\, F_N + L_N \eta_{NL,N} B_n P_{n,N}^3\right)}{P_{in,N} L_N G_N}$$

The gain of each amplifier can be simplified:

$$\frac{1}{GSNR_{AB}} = \frac{P_{\text{base}}\, F_0}{P_{\text{out }, A}} + \frac{P_{\text{base}}\, F_1 + L_1 \eta_{NLL,1} B_n P_{in,1}^3}{P_{in,1} L_1} + \ldots + \frac{P_{\text{base}}\, F_1 + L_N \eta_{NLL,N} B_n P_{in,N}^3}{P_{in,N} L_N}$$

The gain as been simplified, so the problem has to be solver in terms of output power. Each term only depend on a single amplifier contribution, thus, each term can be written as:

$$GSNR_0 = \frac{P_{\text{out},A}}{P_{\text{base}}\, F_0} \quad GSNR_i = \frac{P_{in,i} L_i}{P_{\text{base}}\, F_i + L_i \eta_{NLL,i} B_n P_{in,i}^3} \tag{3.17}$$

So, the generic expression can be simplified into:

$$\frac{1}{GSNR_{AB}} = \frac{1}{GSNR_0} + \sum_{i=1}^{N} \frac{1}{GSNR_i} \tag{3.18}$$

Thus, the total inverse GSNR is given by the sum of the booster amplifier plus every amplifier contribution. This means that the optimization of the line is a problem of optimizing every component. So, this method is called **local optimization global optimization**.

$$max\left\{GSNR_{AB}\right\} \Leftrightarrow max\left\{GSNR_i\right\} \forall i = 1, \ldots, N \tag{3.19}$$

The optimization problem can be solved equating the first derivative to zero according to fig. 3.18.

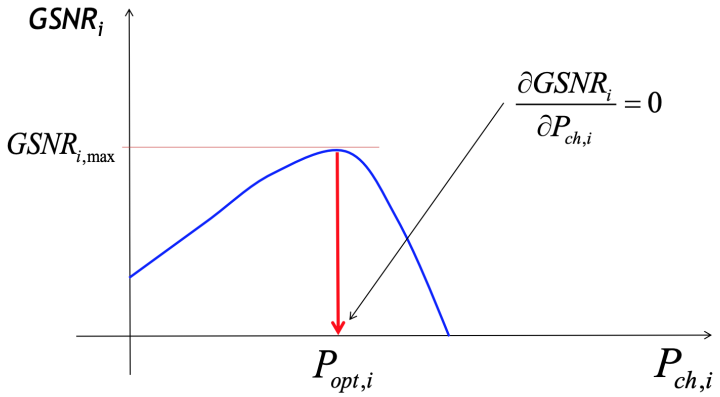


Figure 3.18: Single amplifier GSNR behaviour. The chart describes that the maximum GSNR value can be obtained when the derivative is equal to zero

Single GSNR's value is described by 3.17, so optimal input power and the relative GSNR can be computed as:

$$P_{\text{opt},i} = \sqrt[3]{\frac{F_i L_i P_{\text{base}}}{2 B_n \eta_{NLl,i}}} \Leftrightarrow GSNR_{i,\,\text{max}} = \frac{2}{3} \sqrt[3]{\frac{1}{2\eta_{NLL,i} B_n \left(F_i L_i P_{\text{base}}\right)^2}} \tag{3.20}$$

As introduced, from 3.20 input power is directly proportional to the ASE and inversely to the NLI. By adding all the contributions, it is possible to calculate the GSNR across the OLS.

$$\frac{1}{GSNR_{AB}} = ISNR_{AB} = \frac{P_{\text{base}}\, F_0}{P_{\text{out},A}} + \frac{2}{3}\sum_{i=1}^{N}\sqrt[3]{2\eta_{NLL,i}B_n\left(F_i L_i P_{\text{base}}\right)^2} \tag{3.21}$$

$$\eta_{NLL} \cong \frac{16}{27\pi}\log\left\{\frac{\pi^2}{2}\frac{|\beta_2|\, R_s^2}{\alpha}N_{\text{ch}}^{2\frac{R_s}{\Delta f}}\right\}\frac{\alpha}{|\beta_2|}\frac{\gamma^2 L_{eff}^2}{R_s^3} \tag{3.22}$$

In 3.22 worst chase with N_{ch} is considered, otherwise adding a new channel could cause a disservice.

Chapter 4

Open Optical Transport Network Controller

In accordance with the previous chapters, typically software within an open optical networks are divided into modules. [54]. However, so long as multiple projects are ongoing, no global solution is available. One of the purposes of this project, is the exchange of information between ONC and DT, in order to deploy lightpaths with the best modulation format. In order to perform this communication, there are two main possibilities:

- Integrate the code inside the ONC in order to contact the digital twin before starting the RWSA procedure.
- Create an external request handler acting as a dispatcher between all the components on the network.

The first approach has been discarded due to too many drawbacks: massive changes to a specific ONC would make the project too tied to the single implementation and adaptation to a new ONC would force to restart all the coding. In this chapter, a tool capable to handle a traffic request has been developed. This tool aims to harmonize multiple software modules through existing standard model and interfaces.

4.1 Software Interaction and Structures

As all the logic as been moved out from the ONC a new component has been defined: Open Optical Transport Network Controller(OOTNC). The purpose of this software is to be able to provide a higher level interface, capable of accepting traffic requests and managing them automatically. Furthermore, OOTNC does

not limit itself to accept connections requests implementing the lightpath, but through a QoT-E software , it verifies the best modulation format feasible across the path. Hence, this framework manages to satisfy a traffic demand by guaranteeing continuity of wavelength, transparency and the best possible modulation format. All this takes place in times that can be considered acceptable when compared with current scientific literature.

As shown in 4.1, OOTNC, rely on multiple data structure build upon data incoming from the different network controllers.

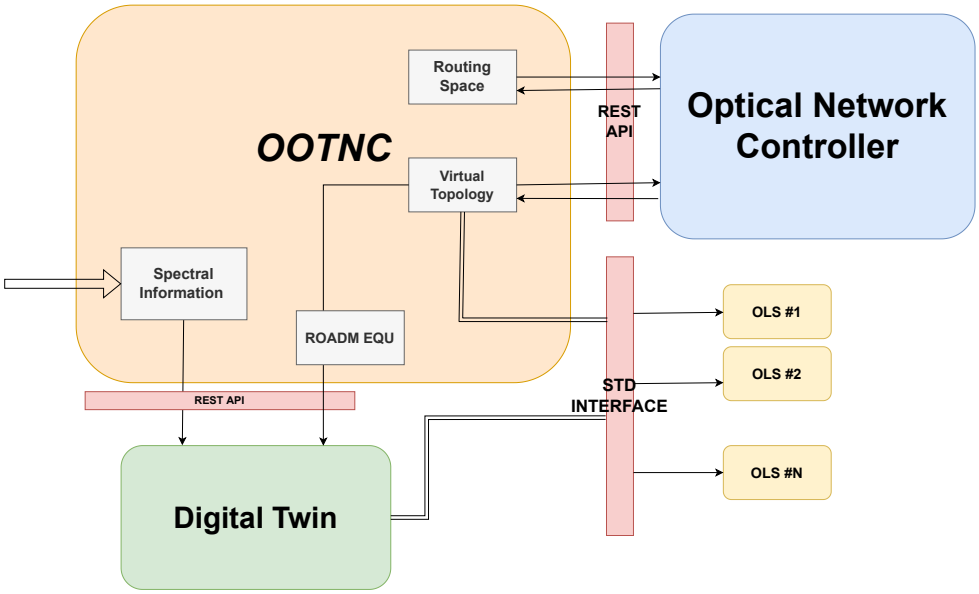


Figure 4.1: General structure of the whole framework. Open Optical Transport Network Controller relies on various data structures and rest interface to build a complete network controller.

4.1.1 Optical Network Controller

The ONC chosen for this experiment is ONOS [55]. ONOS is an open-source project, moslty developed in java [56], with Apache Maven [57] as build system. Moreover, Karaf [58] is used as OSGi [59] module.

Although ONOS has been developed to work as a SDN controller for TCP/IP relying on OpenFlow, a lot of work has been done to exploit the same software to control physical layer [60, 61]. ONOS is an intent based software [62].

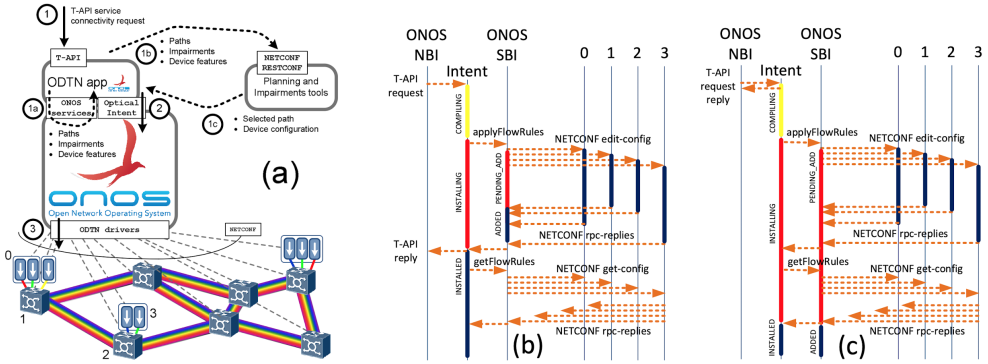


Figure 4.2: ONOS' process for requesting a traffic request [60]. (a) The request is received through a TAPI interface, internally ONOS checks its validity and applies its own RWSA algorithms. Then, through NBI and SBI the configurations are sent to the devices.

4.1.2 Quality of Transmission Estimator

Gaussian Noise in Python (GNPy) is an open-source, vendor-neutral QoT-e software [63]. GNPy is developed by Open Optical Packet Transport Physical Simulation Environment (OOPT–PSE) within Telecom Infra Project (TIP) [64]. GNPy architecture is described in fig. 4.3.

GNPy requires a set of input data for each of the network components along the path in order to get useful GSNR results. These parameters are offered in stable versions either as a series of Microsoft Excel files that are automatically translated into an analogous JavaScript object notation (JSON) format. The ASE noise and the NLI disturbance caused by the nonlinear fiber propagation are calculated using the parameters as inputs in combination.

In order to retrieve the total GSNR along the path, amplifiers description is needed. According to [66], three possible model for EDFA are possible:

- Complete description of the white-box, which allows us to access all the network functions of the device.
- Model provided by the manufacturer that allows access to only some features of the device decided by the vendor.
- Black box model where nothing can be accessed.

GNPy aims to guarantee a multi-vendor support, however, many scenarios do not offer a complete description of the network. Because of this, GNPy proposes a probing tool capable to understand amplifiers configuration. Within the probing technique, some data like fiber spans needs to be characterized.

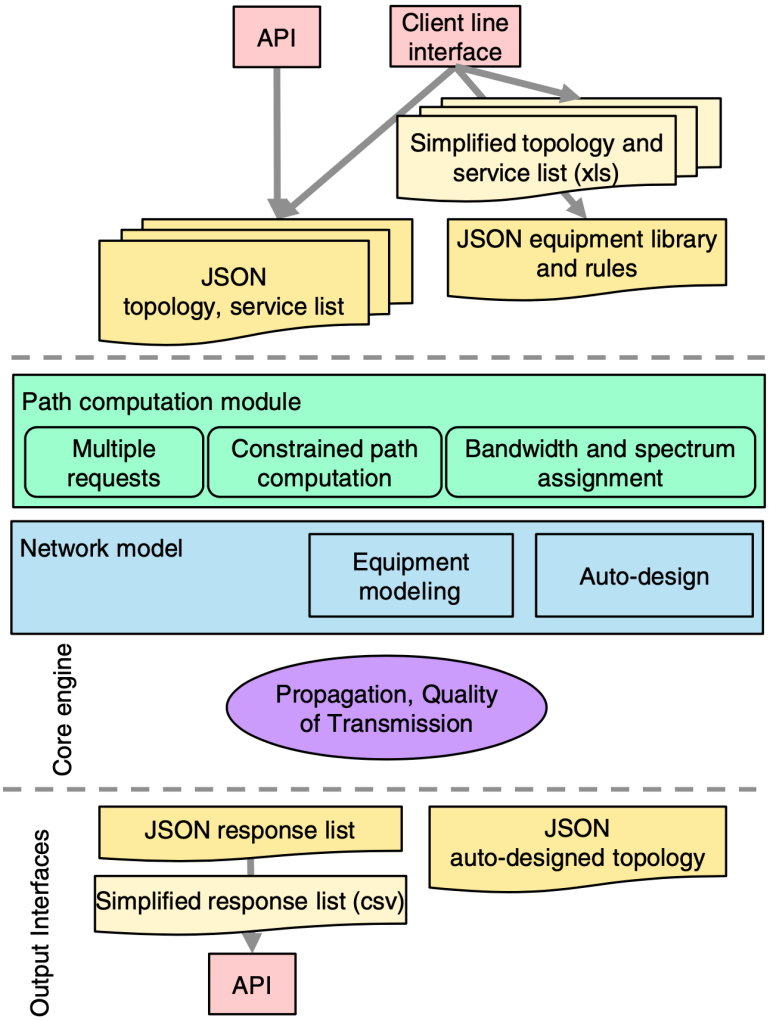


Figure 4.3: GNPy has a general structure [65]. GNPy rely on a core engine within all the propagation is performed and the GSNR is evaluated. Then, it proposed input data interface and output data interface.

The physical layer characterization starts with optical time domain reflectometer (OTDR) analysis in order to measure each fiber span length. To virtualize a single fiber span we need to estimate:

- $\alpha(f)$: loss coefficient function
- $l(0)$: input connector losses.
- $l(L_S)$: output connector losses.

- $l(0 < z < L_S)$: losses detected by the OTDR
- C_R: Raman efficiency scale factor.

Before running the simulation, GNPy executes an optimization algorithm in order to maximize the GSNR value. Then, the evaluation of GSNR is done through cognitive approach setting the amplifier's working point and estimating the value for each OLS.

4.1.3 Optical Line Controller

Since the amplifiers on which the experiment was carried out do not have an open controller, the characterization of these was exploited using a black-box approach. After the calculation of the working points has been carried out, as these do not expose standard interfaces for setting the values, the working point have been set via SSH.

4.1.4 Data Structures

OOTNC rely on multiple data structures. Some of these, are data structure populated through data incoming from other software components like ONC, while, other structures are retrieved from database as external configurations.

Spectral Information

Spectral Information is a data structure build upon static external configuration providing the central frequencies for all the signals following the ITU-T standard [67]. The purpose of this data structure is to provide a single configuration for channel frequencies for use in all software modules.

Virtual Topology

Virtual Topology is a set for classes describing nodes within the network. Each NE is described in terms of:

- IP+NETCONF_Port: this is provided from ONC which is the network element handling all the connections.
- Custom Name: in order to provide a more readable version to the user, a custom name is set. Usually, T followed by an incremental index in used to describe the endpoints, then, R followed by an incremental index in used to describe the ROADMs. A dataframe keeps the bind between custom names and IPs.

- Neighbors: is a list of all the neighboring nodes, which allows to obtain the description of the network in terms of connections.

Routing Space

Relying on virtual topology and spectral information, RS can be build. Starting from virtual topology, all the feasible paths within the network can be computed. Then, according to spectral information and retrieving all the available frequencies in the network, channel abstraction can be used. Spectral occupation is retrieved from onos.

	CH-1	CH-2	CH-3	CH-4
T-0_R-1_R-2_T-1	0	1	1	0
T-0_R-1_R-0_R-2_T-1	0	1	1	0
T-1_R-2_R-1_T-0	0	1	1	0
T-1_R-2_R-0_R-1_T-0	0	1	1	0

Figure 4.4: In this routing space simple example, only two path (with two directions) and 4 channel are shown. Only channel 2 and 3 are available, while channel 1 and 4 are marker as unavailable.

In order to keep the routing space and the modulation format information, two python dataframes [68] have been created. Those have been created in order to be overlappable. Each row represents a possible path among the network, and each column represents a feasible channel. As a free channel is represented as 1 and a busy is represented through 0, a logical AND between routing space and modulation formats dataframe will provide a table with all the modulation format for all the free channels within the network (fig. 4.5). In this way, in order to select the best modulation format, OOTNC just needs to select the maximum value and read row and column's value (fig. 4.6). After that a path, modulation format and channel have been selected, a request for the ONC is built and sent with all the parameters.

4.1.5 ONOS New Optical REST-API Interfaces

As described in the general structure, a communication interface is needed to exchange data between ONOS and OOTNC. During the develop, different solutions have been evaluated:

```
RS AND MF
                      CH-1  CH-2  CH-3  CH-4
T-0_R-1_R-0_R-2_T-1     0     2     2     0
T-0_R-1_R-2_T-1         0     2     4     0
```

Figure 4.5: Routing space and modulation format have been build in order to have the same structure. Thus, all the rows will contain all the path and the columns have been obtained on the basis of the spectral information, therefore the columns of both of DFs appear to be consistent with each other. Therefore, a third DF can be obtained by superimposing the previous two. Each value of this structure will contain the modulation format if available, zero otherwise.

```
Path selected:   T-0_R-1_R-2_T-1
Channel selected:    CH-3
Modulation format selected: 4
Deployed bit rate:  200
Remaining bit rate:  200
RS AND MF
                      CH-1  CH-2  CH-3  CH-4
T-0_R-1_R-0_R-2_T-1     0     2     2     0
T-0_R-1_R-2_T-1         0     2     0     0
```

Figure 4.6: Relying on on the data structure obtained, through a column sorting operation, it is possible to derive the best modulation format available to satisfy the request. The row index will be the path to follow, the column index will be the channel to be occupied, while the DT value will be the modulation format. In the example shown in the figure, the path chosen is "T-0_R-1_R-2_T-1", on channel 3 which guarantees a bit rate of 200Gbps. Then, the total amount of bit rate is decreased, and the routing space is updated by imposing a zero on the path-channel pair no longer available.

TAPI

TAPI has been already introduced, and fig. 4.2 describes how it is possible to accept requests through this interface [69].

Optical Network Model REST API

A REST module is already present in ONOS, including only three endpoints:

- GET - **/intents**: Get the optical intents on the network.
- POST - **/intents**: Submits a new optical intent.
- DELETE - **/intents/intentId**: Delete the specified optical intent

Custom Application

The last possible solution is to not use a ready-made solution but create a custom one. TAPI would have been the best solution from the point of view of compatibility and extensibility, however, build-in onos' TAPI implementation has a lot of troubles and a new one is ongoing. While, optical network model REST API inside ONOS doesn't offers all the needed information, and because of these a new custom application with all needed REST endpoints as been developed:

- POST - **/newoptical/intents**: accepts a body with new intent description. The main different with the build-in solution is the possibility to specify the modulation format. ONOS will push the intent from the "ingressPoint" to the "egressPoint" through the "suggestedPath".

```
{
    "appId": "org.onosproject.newoptical",
    "ingressPoint": ingress_point,
    "egressPoint": egress_point,
    "bidirectional": False,
    "signal": {
        "channelSpacing": "CHL_50GHZ",
        "gridType": "DWDM",
        "spacingMultiplier": spacing_multiplier,
        "slotGranularity": 4
    },
    "suggestedPath": {
        "links": links
    },
    "modulationFormat": mf
}
```

  This endpoint, if was able to configure all devices within the network, returns the new intent's id, errors otherwise.

- GET - **/newoptical/intents**: returns all the intents allocated in the network with all the characteristics.

- DELETE - **/newoptical/intents/id**: deactivate the intent with specified id.
- POST - **/newoptical/path-status**: accepts a list of paths in the body request and returns all the free frequencies for all the specified paths.
- GET - **/newoptical/central-frequency**: in order to build a coherent spectral information between ONC and OOTNC, both controllers have to use the same central frequency. So, during the boot phase this end point is used to retrieve the central frequency from the ONC upon that channels number are build .
- GET - **/newoptical/links-status**: retrieve all the free frequencies for all the links in the network with a granularity of 12,5 GHz. This end-point enable to retrieve the status from the network: according to the frequencies specified inside the spectral information for each path and for each link inside the path, all the available channel can be computed for all the path inside the network.

4.1.6 Logger

In order to be able to implements some recovery feature, the characteristics of the active connections must be kept in memory. So, a logger system is needed. The logger was implemented through MongoDB: a NoSQL database, using JSON-like documents.

```
{
    _id: ObjectId('62daa5e007ff4000e8f06835'),
    path: 'T-0_R-0_R-2_T-1',
    bitRate: 200,
    modulationFormat: 4,
    channel: 'CH-6',
    time: ISODate('2022-07-22T13:28:00.870Z')
}
```

Listing 4.1: Intent LOG example

As shown in listing 4.1, a record is composed by:

- **id**: field automatically filled by MongoDB, analogue to key in relational db.
- **path**: string containing all the devices crossed by the intent. All the ROADMs have "R" as root than a number to identify the device precisely, "T" is used for transponders. OOTNC keeps a data structure to bind this nomenclature with specific IP address.

- **bitRate**: used in the recovery part to compute the total lost bit rate after a failure.
- **modulationFormat**: keeps the modulation format selected on the intent.
- **channel**: maintains the channel used for the lightpath. Through signal information data structure, channel-id and the central frequency and be swapped.
- **time**: timestamp added by MongoDB once the record is saved.

4.2 Use-Case and Flows

The use case that has been exploited, in addition to managing a connection request, also aims to activate a recovery procedure when an error is reported. Figure 4.7 shows what the main steps are and which messages are changed between ONC, DT and OLSs. The experiment is divided into three phases:

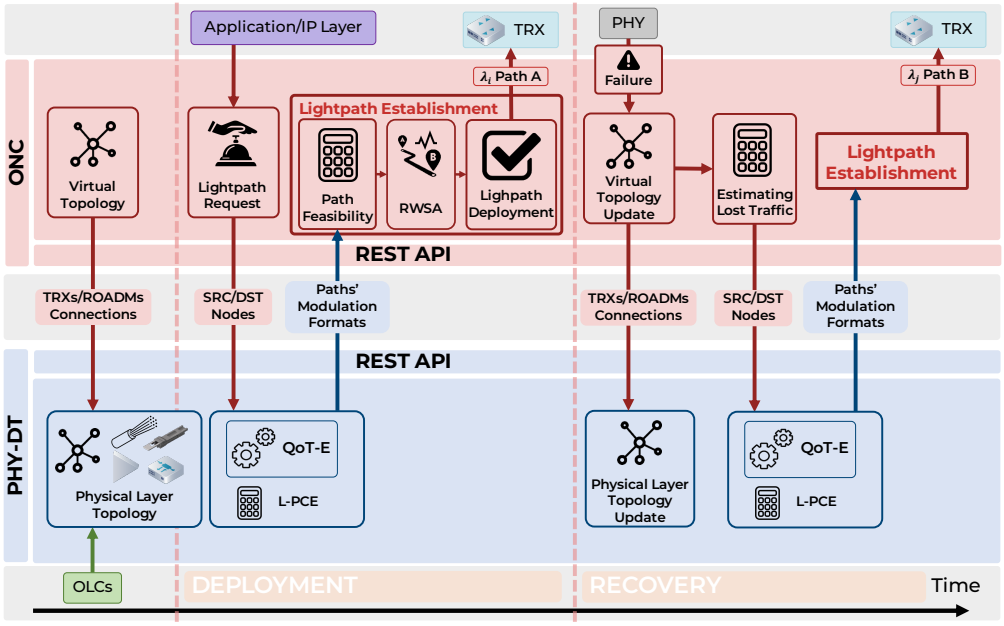


Figure 4.7: Describes the generic architecture flow [70]. It's divided in three main phases exploiting communication between all components.

1. **Boot**: primarily, framework needs to understand the network characteristics. In this case, ONC has the task of providing the description of the nodes that make up the network, while the DT communicates with the OLSs to obtain the

physical characteristics of the network. Also in this phase, OOTNC requests the status of the network from the ONC in order to build the routing space. At the end of this phase, OOTNC has knowledge of both the topology and the state. The network could have been assumed to be completely free, however, in a context of full spectral load it is possible to obtain information on which channels are modulated. Also, this way the framework in question may not be the only player on the network.

2. **Lightpath deployment**: this phase has been revised with respect to the one present in the literature. The main purpose is to obtain information about the best modulation format available, and select the path for the connection in consistency with it. For this purpose, when OOTNC receives a connection request, before starting the lightpath establishment phase, it asks the DT which are the possible modulation formats for each path having the same pair of source and destination. By combining these MFs and the information contained in the RS, OOTNC sends a specific request to ONOS towards the custom endpoints that have been created. ONOS will reply with intent's characteristics, which will be saved in the DB and the RS is updated marking the channel as busy. Among the information returned there is also the amount of allocated bit rate, which is decreased with respect to the requested bit rate. If this value should be greater than zero, then OOTNC iterates the same procedure allocating other connections until all the bit rate is satisfied.

3. **Recovery**: one of the most exploited use chase in disaggregated optical networks is a hard and soft failure scenario [71]. This implementation is also a modification of what is present in the literature, with the aim of carrying out a recovery procedure consistent with the lightpath deployment. OOTNC exposes an endpoint where it is possible to notify the failure of a link or a node. Therefore, previously introduced logger can be exploited, through which all the connections lost due to failure can be retrieved. From this information, it is possible to reconstruct the amount of traffic lost for the source and destination pairs involved. Thus, it is possible to build traffic requests that are in line with the previous point, and the recovery will take place automatically. Also in this case, the difference with what is already present in the literature is that the recovery is done by selecting the path according to the best modulation format.

4.3 Laboratory Network Setup

4.3.1 Devices

Cassini AS7716-24SC

The Cassini packet transponder makes it simple for network operators to upgrade and expand their existing metro and long-haul Dense Wavelength Division Multiplexing (DWDM) networks to support new 100G capacities as well as Layer 3 and inter-datacenter services. With 3.2Tbps of system throughput and a 1.5RU

Figure 4.8: Cassini AS7716-24SC with 8 x DCO cards, 16 x QSFP28 100G Ethernet ports, management port and console port.

form factor, the Cassini is based on Broadcom StrataXGS Tomahawk Plus switch technology. In addition to eight linecard slots for a customizable combination of extra 200GbE ports or DCO optical connections based on coherent DSP and optical transceivers from top optical technology partners, the Cassini design features sixteen fixed 100 Gigabit Ethernet QSFP28 ports. The Open Network Install Environment (ONIE), which facilitates the installation of network operating system (NOS) software, is preinstalled on the open network switch. This NOS software includes the open source alternatives Open Network Linux and various commercial NOS products. For safe encrypted communications on client-side links as well as metro or wide area connections, Cassini supports Ethernet and optical line cards with MACsec security.

The TRXs are Lumentum CFP2-DCO coherent pluggables that are set up to provide 4 separate signals (DP-QPSK or DP-16-QAM modulated) and continually track the associated bit error rate (BER).

Lumentum ROADM-20 Whitebox

A route-and-select architecture can be used to build a flexible-spectrum colorless-directionless (CD-F) or colorless-directionless-contentionless (CDC-F) ROADM solution using the TrueFlex® Twin Wavelength Selective Switches (Twin 2x9, Twin 2x20, or Twin 2x32 depending on application demand). Modern switchable gain preamp and booster EDFAs offer a significantly wider dynamic range and improved NF performance than older variable-gain EDFAs. While the EDFA is not in use, a dynamic gain operation range can be chosen.

Figure 4.9: Lumentum ROADM Graybox with: variable gain pre-amp and booster EDFAs, twin 1x20/1x9/1x32 WSS for express and add/drop fan-out, OCM implementing channel monitoring and OSC termination.

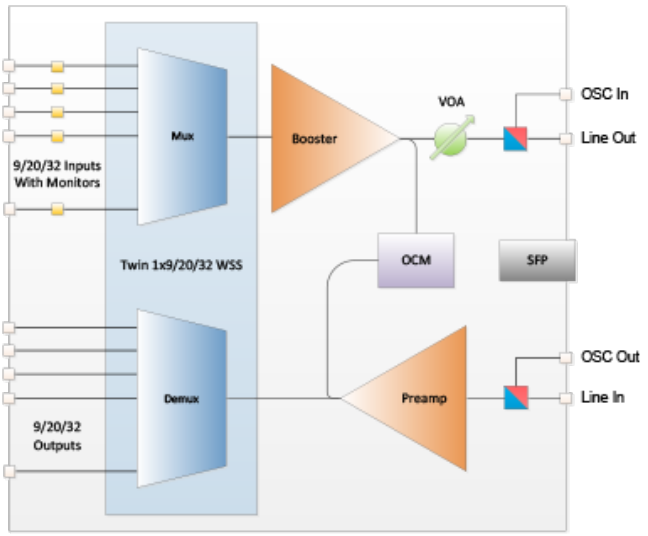


Figure 4.10: Internal Lumentum ROADM Schematic.

4.3.2 Topology

The network topology used in the experiment is shown in the figure 4.11. The network implements a triangular topology with three ROADM a two transponders. Line 1 is composed by 6 single mode fiber spans with as many Erbium Doped Fiber Amplifiers; five spans of 100Km are used to compose Line 2A and Line 2. The

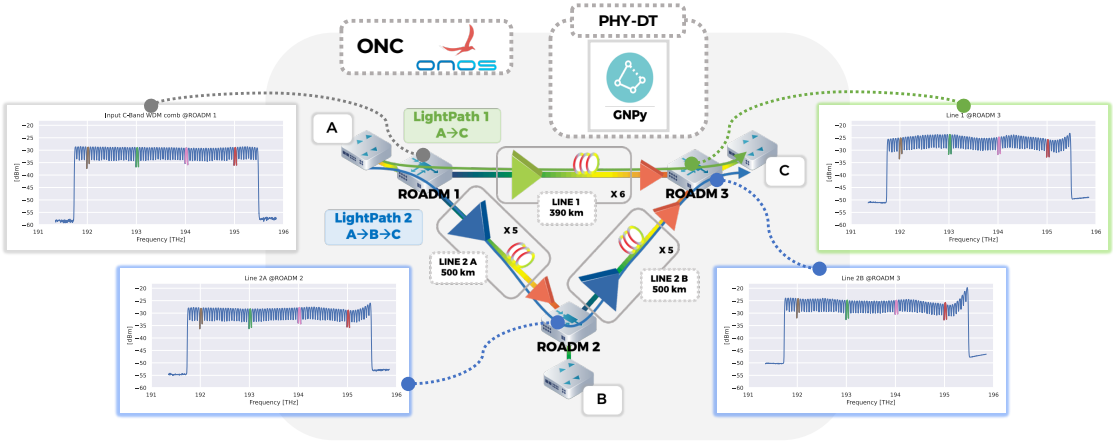


Figure 4.11: Scheme of the set-up of the experiment in the laboratory. The topology has a triangular shape with the cassini located in the vertices, therefore, each endpoint can be reached through a path and a long one.

triangular arrangement each end device (A-B-C) can be reached from a short path or a long one. For instance, if our source is A and the destination is C, a lightpath can go through ROADM-1 and ROADM-3 or through ROADM-1, ROADM-2 and ROADM-3. Each transponder has four CFP2-DCO transceivers modulating four channels centered at 192, 193, 194 and 195 THz. A server equipped with Ubuntu 20.04 runs ONOS with the custom REST application running on. Inside ONOS a docker container is configured to shape the output of an ASE noise source, yielding 71 channels that, together with the 4 CUTs, build the 75 channels fully covering the C-band.

Chapter 5

Results

This chapter will present all the results obtained within this experiment.
The first part will present all the PHY characterization and all the feasible modulation formats. Then, all the results from the recovery use case will be shown.

5.1 Physical Layer Characterization and QoT-E

As first, physical layer characterization is exploited in order to understand all the parameters needed in GNPy. Table 5.1, report all the physical layer characterization. Thanks to the physical layer characterization, GNPy can calculate the optimum working points of each amplifier (tab. 5.2). As extensively discussed in the previous chapters, once the amplifiers can be described in terms of gain and noise, GNPy can compute in GSNR along the entire path. Equal to this it is then possible to obtain the modulation format. BER's measures and GNPy prediction are presented in 5.3:

First of all, it is possible to notice how the DP-16-QAM modulation format is not feasible at the long path, but only at the short path. Then, observing the values of the simulation on BER obtained through GNPy, it is possible to notice how all these are conservative, with an acceptable margin along all the lines.

5.2 Recovery experiment

The purpose of the experiment is to demonstrate how it is possible to achieve automatic recovery after a hard failure. This type of scenario has been extensively exposed in the literature [72, 71, 73], however, this solution not only aims to reroute lost traffic, but does so by choosing the path according to the best modulation format that can be offered. ONOS' topology view is shown in 5.1.

LINE	SPAN	L_S [km]	C_R [1/W/km]	D [ps/nm/km]	$l(0)$ [dB]	$l(L_S)$ [dB]
1	1	65.5	0.34	16.6	5.5	0.1
	2	65.3	0.34	16.8	1.4	0.3
	3	65.5	0.44	16.7	1.6	0.1
	4	65.6	0.34	16.7	0.2	1.4
	5	65.2	0.42	16.7	0.5	0.4
	6	65.8	0.34	16.5	0.1	1.3
2A	1	106.2	0.34	17.5	3.6	0.2
	2	107.5	0.44	17.9	1.2	0.7
	3	106.2	0.44	17.7	1.5	0.1
	4	108.8	0.42	17.7	0.6	0.1
	5	108.3	0.42	17.8	0.2	0.1
2B	1	106.2	0.42	17.9	1.1	0.2
	2	106.8	0.34	17.7	0.1	0.1
	3	106.4	0.34	17.7	0.2	0.7
	4	107.3	0.42	17.8	0.2	0.1
	5	108.3	0.42	17.8	0.5	2.3

Table 5.1: PHY Characterization [70].

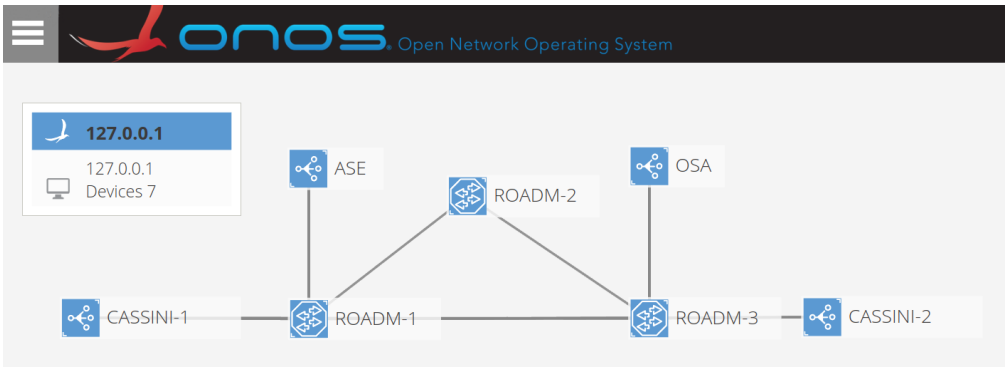


Figure 5.1: Topological view of ONOS on the triangular network. Two devices have been added through docker: ASE ensures that the network is full shaped and an optical spectrum analyzers.

5.2.1 Boot

The first phase, OOTNC needs to boot and fill all the data structures. In this case it is assumed that the network and the ONC are up and running. Fig. 5.2 shows all the traffic request between OOTNC and ONOS. At the end of the boot phase, all the OOTNC's data structures will be populated and updated with the current

LINE	AMPLIFIER	G [dB]	T [dB]	P_{OUT} [dBm]
1	BST	–	–	21.8
	ILA 1	15.0	-0.1	–
	ILA 2	15.0	-1.4	–
	ILA 3	15.0	0.0	–
	ILA 4	15.0	0.6	–
	ILA 5	15.7	-1.0	–
	PRE	–	–	20.0
2A	BST	–	–	21.8
	ILA 1	23.3	-5.0	–
	ILA 2	22.1	-5.0	–
	ILA 3	21.6	-1.9	–
	ILA 4	22.9	-1.0	–
	PRE	–	–	23.0
2B	BST	–	–	19.2
	ILA 1	22.0	-5.0	–
	ILA 2	22.2	-4.8	–
	ILA 3	23.3	-1.9	–
	ILA 4	23.0	-1.4	–
	PRE	–	–	20.0

Table 5.2: EDFA Optimal Working Point [70]

		LP 1 (A ->C)				LP 2 (A ->B ->C)			
		CUT 1 DCO (192 THz)	CUT 2 ACO (193 THz)	CUT 3 DCO (194 THz)	CUT 4 ACO (195 THz)	CUT 1 DCO (192 THz)	CUT 2 ACO (193 THz)	CUT 3 DCO (194 THz)	CUT 4 ACO (195 THz)
GNPy Prediction [dB]		24.0	23.7	23.7	23.6	18.4	17.8	18.1	17.6
QPSK (100G)	BER	1.6e-8	9.5e-8	1.2e-08	8.6e-08	4.2e-05	1.9e-04	3.5e-05	1.4e-04
	GSNR [dB]	27.1	24.6	27.5	24.7	19.1	17.7	19.2	18.0
	Margin [dB]	3.1	0.9	3.8	1.1	0.7	-0.1	1.1	0.4
16-QAM (200G)	BER	3.9e-03	9.9e-3	4.2e-03	1.1e-02	–	–	–	–
	GSNR [dB]	26.3	25.0	26.0	24.7	–	–	–	–
	Margin [dB]	2.3	1.3	2.3	1.1	–	–	–	–

Table 5.3: Network Transmission Performance Validation Results [70]

state of the network.

Time	Source	Destination	Protocol	Info
10.660422834	OOTNC	ONOS	HTTP	GET /onos/v1/devices HTTP/1.1
10.697731017	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
10.722922629	OOTNC	ONOS	HTTP	GET /onos/v1/devices/netconf:192.168
10.754414131	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
10.780547147	OOTNC	ONOS	HTTP	GET /onos/v1/devices/netconf:10.100
10.837254512	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
10.862924577	OOTNC	ONOS	HTTP	GET /onos/v1/devices/netconf:10.100
10.919462935	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
10.945357734	OOTNC	ONOS	HTTP	GET /onos/v1/devices/netconf:192.168
10.976959775	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
11.001834292	OOTNC	ONOS	HTTP	GET /onos/v1/links HTTP/1.1
11.060687452	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
11.086651467	OOTNC	ONOS	HTTP	GET /onos/newopticalrest-app/newopt
11.113870264	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
11.141667017	OOTNC	ONOS	HTTP	GET /onos/newopticalrest-app/newopt
12.558236487	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
29.170926906	OOTNC	connecti…	HTTP	GET / HTTP/1.1
29.310023888	connectiv…	OOTNC	HTTP	HTTP/1.1 204 No Content

Figure 5.2: Wireshark [74] traffic capture with all the traffic requests. The figure shows how OOTNC exploits both its own ONOS endpoints and those created ad hoc for this experiment. The amount of time it takes to exchange messages is about 2 seconds, while OOTNC is ready to accept other requests after others 17 seconds.

5.2.2 Traffic Deployment

As first, a traffic request as the one in the listing 5.1 is generated.

```
{
    "src": "T-0",
    "dst": "T-1",
    "bit_rate": 400,
    "qot-e" : true
}
```

Listing 5.1: Within the request the path is not specified, but only the source and destination node and the amount of traffic that must be allocated in terms of Gbps. A Boolean specifies whether DT should be used to evaluate the modulation format.

Figure 5.3 schematises all the requests that are exchanged before allocating the traffic. Once the request has been received, based on the virtual topology present in itself, OOTNC will ask the DT for all modulation formats for all paths that have the same source-destination pair as the request just received. Relying on the GNPY prediction, all the modulation format will be returned to OOTNC. According to the simulation, the shortest path (CASSINI-1 -> ROADM-1 -> ROADM-3 -> CASSINI-2) can provide intents with a 16-QAM modulation format. At this point, OOTNC will build the requests to be sent to the ONC until the bit rate is exhausted. All the intents characteristics are saved within a database. All the messages exchanged are reported in 5.4 and according to this, to total amount to

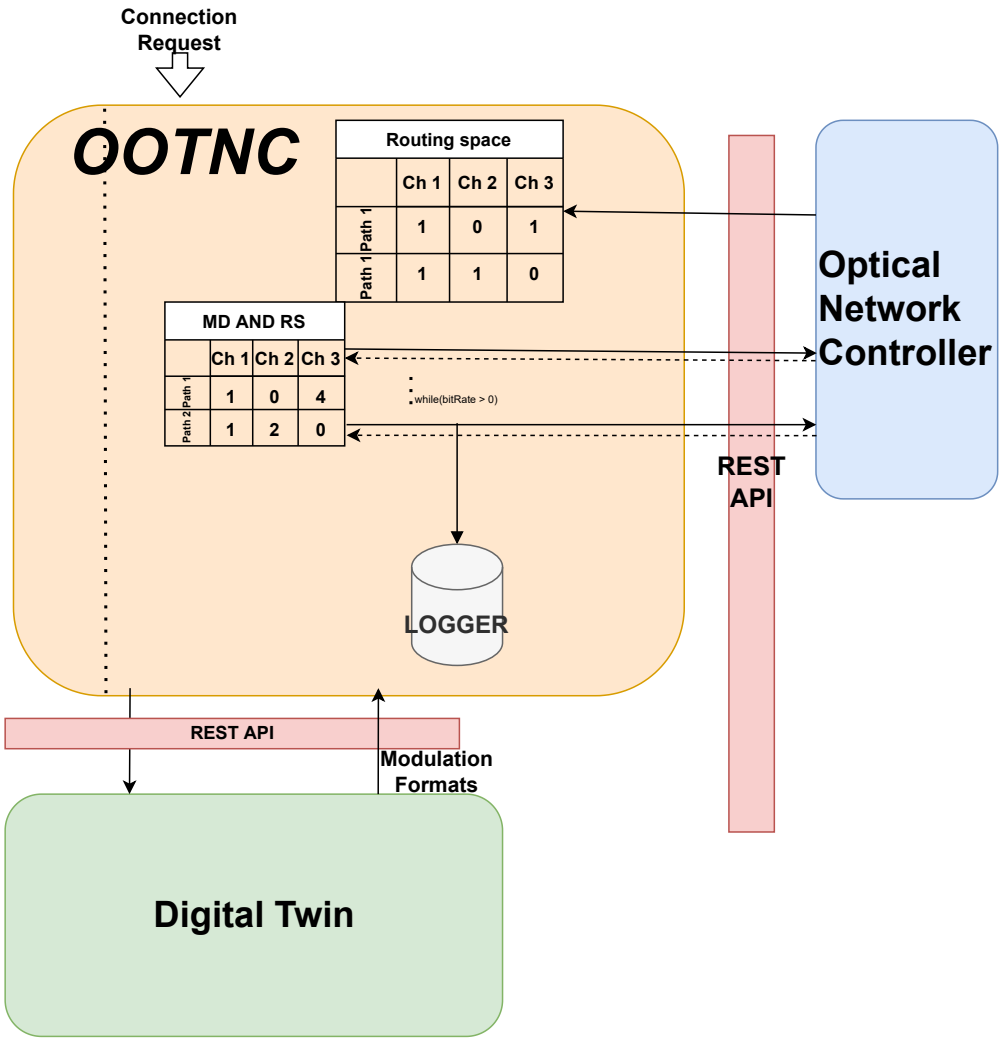
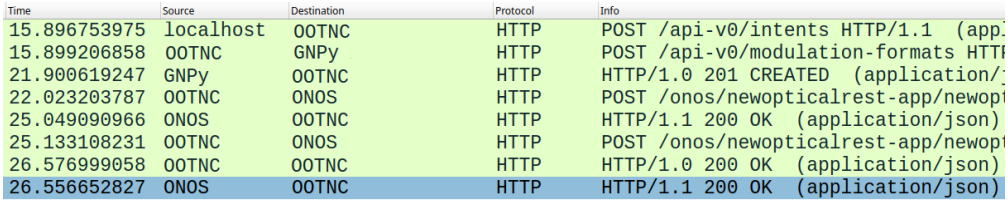


Figure 5.3: Lightpath request flowchart. Each connection request is handled by OOTNC by asking the best modulation format to the DT. Then, a number of requests towards the ONC is generated consistent with the bit rate to be allocated. For each request that was successful, all the data structure are updated and the connection's data are kept within the database.

time to serve a request of 400Gbps is about 11 seconds.

5.2.3 Recovery

At this point, the network is in a stable situation, with two intents active along the short path. By exploiting the OOTNC endpoints reported in listing 5.2, an error is reported about a link break. In this case, the OOTNC virtual topology is



Time	Source	Destination	Protocol	Info
15.896753975	localhost	OOTNC	HTTP	POST /api-v0/intents HTTP/1.1 (appl
15.899206858	OOTNC	GNPy	HTTP	POST /api-v0/modulation-formats HTTP
21.900619247	GNPy	OOTNC	HTTP	HTTP/1.0 201 CREATED (application/j
22.023203787	OOTNC	ONOS	HTTP	POST /onos/newopticalrest-app/newopt
25.049090966	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
25.133108231	OOTNC	ONOS	HTTP	POST /onos/newopticalrest-app/newopt
26.576999058	OOTNC	OOTNC	HTTP	HTTP/1.0 200 OK (application/json)
26.556652827	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)

Figure 5.4: Wireshark traffic capture for a traffic request. The request has been satisfied through two connections in about 11 seconds.

updated allowing to emulate the breaking of a link.

```
{
    "type": "link",
    "srg": "R-0",
    "dst": "R-2"
}
```

Listing 5.2: Broken link notification. The type is needed because even breaking a node is supported.

OOTNC supports both the possible breaking of a link and that of a node. In this specific use case the breaking of a link has been simulated. A link is expressed in terms of source and destination, as links are always considered to be unidirectional. Recovery phase is articulated in steps:

1. **Topology Update**: Once the notification of the error is received, the framework has the task of updating all the data structures. In particular, within the routing space all the lines containing the link specified in the error are zeroed.

2. **Lost traffic estimation:** the purpose of the experiment is not only to provide an alternative path to the fallen one, but to allocate through this all the traffic lost on the broken path. It is therefore necessary to query the database by retrieving all the paths that contain the broken link (or node). In the example, entries will be the two active connections:

```
{
    path: 'T-0_R-0_R-2_T-1',
    bitRate: 200,
    modulationFormat: 4,
    channel: 'CH-6',
    time: ISODate('2022-07-22T13:28:00.870Z')
}
```

Listing 5.3: First intent json returned from the database after the query that searches for all the paths containing the broken link.

```
{
    path: 'T-0_R-0_R-2_T-1',
    bitRate: 200,
    modulationFormat: 4,
    channel: 'CH-26',
    time: ISODate('2022-07-22T13:29:00.560Z')
}
```

Listing 5.4: Second intent json returned from the database after the query that searches for all the paths containing the broken link.

In this way it is possible to compute all the lost traffic (in terms of bit rate) for all the source-destination couple.

3. **L-PCE**: given the modification of the topology, the previous structure that preserves the modulation formats is invalidated and OOTNC needs again to contact the DT for the modulation formats along the paths of interest.

4. **Lightpath Establishment**: with all data and all data structures intent recovery can be performed. OOTNC will automatically create a new intent request with the same source-destinations fields and bit rate equals to the computed lost traffic.

```
{
    "src": "T-0",
    "dst": "T-1",
    "bit_rate": 400,
    "qot-e" : true
}
```

Listing 5.5: New request that is created by OOTNC and handled by itself. The request is exactly the same as the one sent before the break, however, as the topology has been changed, the controller will allocate different paths than it did before.

This request can be handle like a normal intent request since the routing space has been updated and the broken paths are marked as not available.

In table 5.6 all the time for all the steps are reported. Topology update and lost traffic estimation are negligible when compared with the others, given the efficiency of the data frames used. Instead, the other two have a much greater weight, but for different reasons. L-PCE is a process CPU-bound [75], while the lightpath establishment is strongly influenced by the response times of the equipment.

Interaction	Time [s]
Topology Update	0.017
Lost Traffic Estimation	0.012
L-PCE	6.580
Lightpath Establishment	4.870
Total Recovery	11.708

Table 5.4: Times related to the various steps performed during recovery.

The complete answer provided by OOTNC is reported in 5.6. Through the logger, OOTNC is capable to retrieve the two broken paths, and these are replaced with 4 intents because the only feasible path has a lower modulation format according to GNPy's simulations. The total time is approximately 23,5 seconds: about 11,5 for each broken intent. Considering that, this is the total amount (sum of database query, bit rate computation etc.) and that for each broken path two paths are needed, these results look in accordance with existing literature.


```
{
    "brokenPaths":[
        {
            "bitRate":200,
            "channel":"CH-6",
            "id":"62daa5e007ff4000e8f06835",
            "modulationFormat":4,
            "path":"T-0_R-0_R-2_T-1"
        },
        {
            "bitRate":200,
            "channel":"CH-26",
            "id":"62daa5e007ff4000e8f06836",
            "modulationFormat":4,
            "path":"T-0_R-0_R-2_T-1"
        }
     ],
    "allocatedPaths":[
       {
          "path":[
```

```
            {
                "bitRate":100,
                "channel":"CH-26",
                "modulationFormat":2,
                "path":"T-0_R-0_R-1_R-2_T-1"
            },
            {
                "bitRate":100,
                "channel":"CH-46",
                "modulationFormat":2,
                "path":"T-0_R-0_R-1_R-2_T-1"
            }
        ],
        "recoveredPath":"62daa5e007ff4000e8f06835",
        "recoveryTime":11.43623971939087
    },
    {
        "path":[
            {
                "bitRate":100,
                "channel":"CH-6",
                "modulationFormat":2,
                "path":"T-0_R-0_R-1_R-2_T-1"
            },
            {
                "bitRate":100,
                "channel":"CH-66",
                "modulationFormat":2,
                "path":"T-0_R-0_R-1_R-2_T-1"
            }
        ],
        "recoveredPath":"62daa5e007ff4000e8f06836",
        "recoveryTime":11.970945596694946
    }
],
"dst":"R-2",
"srg":"R-0",
"totalRecoveryTime":23.417049884796143,
"type":"link"
}

```

Listing 5.6: Recovery results

All the exchanged messages are reported in 5.5. This result are coherent with the flow described in the previous chapters and the time is coherent with the ones described below.

Time	Source	Destination	Protocol	Info
11.091335753	localhost	OOTNC	HTTP	POST /api-v0/errors HTTP/1.1 (application/json)
11.127307751	OOTNC	GNPy	HTTP	POST /api-v0/modulation-formats HTTP/1.1 (applic
17.236051299	GNPy	OOTNC	HTTP	HTTP/1.0 201 CREATED (application/json)
17.362465178	OOTNC	ONOS	HTTP	POST /onos/newopticalrest-app/newoptical/intents
18.487326339	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
18.573653680	OOTNC	ONOS	HTTP	POST /onos/newopticalrest-app/newoptical/intents
21.201772318	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
21.207483642	OOTNC	GNPy	HTTP	POST /api-v0/modulation-formats HTTP/1.1 (applic
27.168019805	GNPy	OOTNC	HTTP	HTTP/1.0 201 CREATED (application/json)
27.285936842	OOTNC	ONOS	HTTP	POST /onos/newopticalrest-app/newoptical/intents
28.606968544	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
28.692266427	OOTNC	ONOS	HTTP	POST /onos/newopticalrest-app/newoptical/intents
31.219421985	ONOS	OOTNC	HTTP	HTTP/1.1 200 OK (application/json)
31.224022594	OOTNC	localhost	HTTP	HTTP/1.0 200 OK (application/json)

Figure 5.5: Wireshark traffic capture for the recovery phase. The image shows how, once the request has been received, the communication first takes place with GNPy with the aim of obtaining the new modulation formats, from which the various requests to be sent to ONOS are obtained.

Chapter 6

Future Applications and Developments

The work just completed has shown the potential of an open and disaggregated optical network. The network disaggregation enables the development of tools independent from vendor implementations. Furthermore, separation of concerns is even empowered, allowing modularization and the utilization of standard interfaces.

Thus, two important conclusions has been achieved in this thesis work exploiting openness and implementing disaggregation:

1. Network automation is enabled and the carried out experiments encourage its adoption, allowing to manage massive data traffic. Recovery of soft and hard failures can be performed thanks to network virtualization, in addition to re-routing lost connections.

2. Quality of transmission estimation is the main novelty. While automation has been already explored on the other open systems interconnection (OSI) layers, QoT estimation has been used within the automated optical network framework as peculiar feature of the PHY, allowing the maximization of network performance.

The experimental results have been obtained in the photonics laboratory of LINKS Foundation, performing both the automated management of connections on a QoT-validated real optical network and the recovery procedure of the lost data traffic after the detection of a hard failure.
The main future goal will be testing the whole framework within a different network topology using a larger variety of multi-vendor devices. Thanks to the collaboration

with GARR, it is in program to validate the software within their laboratory aiming to add QoT estimation in the production network if encouraging results will be achieved. In the meanwhile, the framework development will go ahead towards a DevOps orientation. Lastly, a contribution for the development of a Transport-API for GNPy is planned.

Appendix A

Code

A.1 app.py

```
from time import sleep

from flask import Flask, request
from json import load

from components import http_utils
from pandas import DataFrame, read_json
from components import virtual_topology
from components import db

app = Flask(__name__)
http_utils = http_utils.HttpUtils()
apiVersion = "/api-v0"

virtual_topology = virtual_topology.VirtualTopology()
virtual_topology.build_routing_space()

intent_db = db.Database()
intent_db.connect_db("logger")

spectral_info_db = db.Database()
spectral_info_db.connect_db("spectral_info")
spectral_info_db.save_spectral_info()


@app.post(apiVersion + '/intents')
def post_intents():
    content_type = request.headers.get('Content-Type')
    if content_type == 'application/json':
```

```
30          try:
31              request_body = request.json
32              source = request_body["src"]
33              dest = request_body["dst"]
34              bit_rate = request_body["bit_rate"]
35              is_qot_required = request_body["qot-e"]
36              if "suggested_path" in request_body:
37                  suggested_path = request_body["suggested_path"]
38                  return allocate_path(is_qot_required, source, dest,
    bit_rate, suggested_path)
39              else:
40                  return allocate_path(is_qot_required, source, dest,
    bit_rate)
41
42          except KeyError as err:
43              print(err)
44              return f'Missed param: {err}', 422
45      else:
46          return "Request must contain a json body", 422
47
48
49  @app.post(apiVersion + '/intents/all-channels')
50  def post_all_intents():
51      content_type = request.headers.get('Content-Type')
52      if content_type == 'application/json':
53          try:
54              request_body = request.json
55              min_ch = request_body["min-ch"]
56              max_ch = request_body["max-ch"]
57              suggested_path = request_body["suggested_path"]
58              return allocate_all_channels(suggested_path, min_ch,
    max_ch)
59          except KeyError as err:
60              print(err)
61              return f'Missed param: {err}', 422
62      else:
63          return "Request must contain a json body", 422
64
65
66  def allocate_all_channels(path, min_ch, max_ch):
67      central_frequency = virtual_topology.central_frequency
68      path_onc = virtual_topology.translate_path_to_onc(path)
69
70      central_frequency = float(central_frequency) * 1e9
71
72      for ch in [min_ch, max_ch - 1]:
73          channel_multiplier = (virtual_topology.spectral_info.
    frequency[int(ch)] - central_frequency) / 50e9
```

```
74          response = http_utils.post_intent_on_onc(2,
    channel_multiplier, path_onc)
75
76      return
77
78
79  def allocate_path(is_qot_required, source, dest, bit_rate,
    suggested_path=None):
80      if not is_qot_required:
81          if suggested_path is not None:
82              print("sending the request to the SDN controller with the
     path:", suggested_path)
83              return request
84          else:
85              print("sending request to the SDN controller without path
    ")
86      else:
87          if suggested_path:
88              if suggested_path in virtual_topology.paths:
89                  response = http_utils.post_intent_on_dt(src=source,
    dst=dest, suggested_path=suggested_path)
90              else:
91
92                  return 'Suggested path is not present in the network'
    , 422
93          else:
94              response = http_utils.post_intent_on_dt(src=source, dst=
    dest)
95
96          mf_df = DataFrame.from_dict(response["modulation_format"])
97
98          print("Modulation Formats:")
99          print(mf_df)
100
101         allocated_paths_desc = []
102
103         while bit_rate > 0:
104             print("routing space")
105             print(virtual_topology.routing_space)
106             rs = virtual_topology.routing_space
107
108             results = evaluate_available_channels(mf_df, rs)
109             if results is None:
110                 break
111             else:
112                 try:
113                     mf, channel, path = rwsa(results)
114
115                 except IOError as err:
```

```
116                     pass
117
118                 allocated_path_desc = send_req_to_onc(mf.item(),
    channel, path, virtual_topology.central_frequency)
119                 print("allocatend path desc:")
120                 print(allocated_path_desc)
121
122                 if not allocated_path_desc == -1:
123                     allocated_paths_desc.append(allocated_path_desc)
124                     if allocated_path_desc["bitRate"] > 0:
125                         bit_rate -= allocated_path_desc["bitRate"]
126                     else:
127                         pass
128                 if len(allocated_paths_desc) == 0:
129                     return "No path allocated", 504
130
131                 _ = virtual_topology.update_rs_path_status(path,
    channel[3:])
132
133         res = request.json
134         res["allocatedPaths"] = allocated_paths_desc
135         intent_db.save_paths(res)
136         return res
137
138
139 def send_req_to_onc(modulation_format, channel, path,
    central_frequency):
140     path_fix = list(path)
141     path_fix[-1] = path[2]
142     fixed_path = ''.join(path_fix)
143     path_onc = virtual_topology.translate_path_to_onc(fixed_path)
144
145     is_response_successfully = False
146     test = 0
147
148     print("channel: ", channel)
149
150     central_frequency = float(central_frequency) * 1e9
151
152     channel_multiplier = ((virtual_topology.spectral_info.frequency[
    int(channel[3:])] - central_frequency) / 50e9) - 1
153
154     while (not is_response_successfully) and test < 3:
155         response = http_utils.post_intent_on_onc(modulation_format,
    channel_multiplier, path_onc)
156         if response == 400:
157             test += 1
158
159         else:
```

```
160             is_response_successfully = True
161         if test > 3:
162             return -1
163             break
164     mf_deployed = response["modulation"]
165     trx_mf2bitrate = load_json("./resources/trx_mf2bitrate.json")
166     bit_rate_dep = trx_mf2bitrate[mf_deployed]
167 
168     allocated_path_desc = {
169         "path": path,
170         "channel": channel,
171         "modulationFormat": int(modulation_format),
172         "bitRate": int(bit_rate_dep)
173     }
174 
175     return allocated_path_desc
176 
177 
178 @app.post(apiVersion + '/errors')
179 def post_errors():
180     content_type = request.headers.get('Content-Type')
181     if content_type == 'application/json':
182         try:
183             request_body = request.json
184             print(request_body)
185             if request_body["type"] == "link":
186                 sub_path = request_body["srg"] + '_' + request_body["
    dst"]
187                 paths_to_be_recovered = intent_db.find_path(sub_path)
188                 print(paths_to_be_recovered)
189                 broken_paths = []
190                 new_paths = []
191 
192                 for path in paths_to_be_recovered:
193                     broken_paths.append(path["path"])
194                     srg = path["path"][0] + path["path"][1] + path["
    path"][2]
195                     dst = path["path"][-3] + path["path"][-2] + path[
    "path"][-1]
196                     new_path = allocate_path(True, srg, dst, path["
    bitRate"])
197                     new_paths.append(new_path)
198                     intent_db.delete_path_from_id(path["_id"])
199 
200                 request_body["broken_paths"] = broken_paths
201 
202                 return request_body, 200
203             return "Ok", 200
204         except AssertionError as err:
```

```
205             return f'bad request {err}', 422
206     else:
207         return 'bad req', 422
208
209
210 @app.get(apiVersion + '/paths')
211 def get_paths():
212     res = {
213         "paths": virtual_topology.paths
214     }
215     return res
216
217
218 @app.get(apiVersion + '/devices')
219 def get_devices():
220     res = {
221         "dev": virtual_topology.nodes_custom_to_ip
222     }
223     return res
224
225
226 @app.get(apiVersion + '/links')
227 def get_links():
228     res = {
229         "links": virtual_topology.links_custom_to_ip
230     }
231     return res
232
233
234 @app.delete(apiVersion + '/intents')
235 def delete_all_intents():
236     intents = http_utils.get_intents_from_onc()
237     appId = request.args.get('appId')
238
239     for intent in intents:
240         http_utils.delete_intent(intent["key"], application_id=appId)
241     return "Done", 204
242
243
244 def evaluate_available_channels(df_mf: DataFrame, df_rs: DataFrame):
245     index_mf = df_mf.index.values
246     df_rs_sel = df_rs.loc[index_mf]
247     df = df_mf * df_rs_sel
248
249     df_zeros = df.transpose().sum()
250     if df_zeros.sum() == 0:
251         return None
252     index_zeros = df_zeros.index.values
```

```
253     df_clean = df.loc[[col for col in index_zeros if df_zeros[col] !=
        0]]
254
255     df_clean.to_csv("./results/and.csv")
256     return df_clean
257
258
259 def rwsa(df: DataFrame):
260     df.to_csv("./results/res.csv")
261     if df.empty:
262         raise IOError("rwsa cannot process empty dataframe")
263     sort_index = df.transpose().max().sort_values(ascending=False).
        index.values
264     df_sorted = df.loc[sort_index]
265
266     selected_path = df_sorted.iloc[0]
267     path = df_sorted.index.values[0]
268     channel = selected_path.idxmax()
269     mf = selected_path[channel]
270
271     return mf, channel, path
272
273
274 if __name__ == '__main__':
275     app.run(host="127.0.0.1", port=5001)
276
277
278 def load_json(file_path):
279     with open(file_path, 'r') as f:
280         data = load(f)
281     return data
```

A.2 virtualTopology.py

```
1  import json
2
3  from numpy import argmin, array
4  from numpy import abs as abs_np
5  from pathlib import Path
6  from pandas import DataFrame
7  from components import http_utils
8  from components import db
9
10 from osi.core.info import SpectralInformation
11 from osi.tool.configurations import Config
```

```
12
13 root = Path(__file__).parent.parent
14
15 http = http_utils.HttpUtils()
16
17
18 class VirtualTopology:
19     def __init__(self):
20         self._nodes = {}
21         self._paths = []
22         self._routing_space = None
23         self._nodes_ip_to_custom = {}
24         self._nodes_custom_to_ip = {}
25         self._links_ip_to_custom = {}
26         self._links_custom_to_ip = {}
27         self._grid_type = "FIXED"
28
29         spectral_info_db = db.Database()
30         spectral_info_db.connect_db("spectral_info")
31
32         si_json = spectral_info_db.get_spectral_info()
33         config = Config.as_config(si_json)
34         self._spectral_info = SpectralInformation.from_config(config)
35
36         if not all(spacing == self._spectral_info.slot_width[0] for
    spacing in self._spectral_info.slot_width):
37             self._grid_type = "FLEX"
38
39         device_response = http.get_devices_from_onc()
40         if device_response is None:
41             return
42
43         devices_list = device_response["devices"]
44
45         roadm_count = 0
46         terminal_count = 0
47         for device in devices_list:
48              custom_name: str
49              if device["type"] == "ROADM":
50                  custom_name = 'R-' + str(roadm_count)
51                  roadm_count += 1
52              if device["type"] == "TERMINAL_DEVICE":
53                  custom_name = 'T-' + str(terminal_count)
54                  terminal_count += 1
55
56             custom_name = topo_fix[device["id"]]
57
58         for device in devices_list:
59             if device["type"] == "TERMINAL_DEVICE":
```

```
60                  content = http.get_ports_on_dev_id(device["id"])["
    ports"]
61                  ports = [el for el in content if el["type"] == "och"]
62                  self._nodes[self._nodes_ip_to_custom[device["id"]]]["
    ports"] = ports
63
64          link_response = http.get_links_from_onc()
65
66          links_list = link_response.json()["links"]
67
68          for link in links_list:
69              ip1 = link["src"]["device"]
70              ip2 = link["dst"]["device"]
71
72              ip_and_port1 = link["src"]["device"] + '/' + link["src"]["
    port"]
73              ip_and_port2 = link["dst"]["device"] + '/' + link["dst"]["
    port"]
74
75              self._links_ip_to_custom[ip_and_port1 + '-' +
    ip_and_port2] = self._nodes_ip_to_custom[
76
                link["src"]["device"]] + "_" + \
77
          self._nodes_ip_to_custom[ip2]
78              self._links_custom_to_ip[
79                  self._nodes_ip_to_custom[ip1] + "_" + self.
    _nodes_ip_to_custom[ip2]] = ip_and_port1 + '-' + ip_and_port2
80
81              self._nodes[self._nodes_ip_to_custom[ip1]]["
    connected_nodes"].append(self._nodes_ip_to_custom[ip2])
82
83
84          self.compute_all_paths()
85          print(self._paths)
86
87          self._central_frequency = http.get_central_frequency()["
    centralFrequency"]
88
89      @property
90      def spectral_info(self):
91          return self._spectral_info
92
93      @property
94      def paths(self):
95          return self._paths
96
97      @property
98      def nodes_custom_to_ip(self):
```

```
99          return self._nodes_custom_to_ip
100
101     @property
102     def links_custom_to_ip(self):
103         return self._links_custom_to_ip
104
105     @property
106     def routing_space(self):
107         return self._routing_space
108
109     @property
110     def central_frequency(self):
111         return self._central_frequency
112
113     def compute_all_paths(self):
114         for src in self._nodes:
115             for dst in self._nodes:
116                 if src != dst:
117                     if "T-0" in src or "T-1" in src:
118                         if "T-0" in dst or "T-1" in dst:
119                             paths = self.find_paths(src, dst)
120                             for path in paths:
121                                 if path not in self._paths:
122                                     self._paths.append(path)
123
124     def find_paths(self, start, end, path=[]):
125         graph = self._nodes
126         path = path + [start]
127         if start == end:
128             return [path]
129         if start not in graph.keys():
130             print("Node not in chart")
131             return []
132         paths = []
133         for node in graph[start]["connected_nodes"]:
134             if node not in path:
135                 newpaths = self.find_paths(node, end, path)
136                 for newpath in newpaths:
137                     paths.append(newpath)
138         return paths
139
140     def build_routing_space(self):
141         free_channel_res = http.get_all_free_channel()
142         links_status = []
143         if free_channel_res is None:
144             return
145         if "Links" not in free_channel_res:
146             print("No link found")
147             return
```

```
148         for link in free_channel_res["Links"]:
149             sorted_freq = array(sorted(link["available"], key=float))
150             links_status.append({
151                 self._links_ip_to_custom[link["src"] + "-" + link["
    dst"]]: sorted_freq
152             })
153 
154         if self._grid_type == "FIXED":
155             spacing = self._spectral_info.baud_rate[0] / 1e9
156             links_availability = {}
157             for link in links_status:
158                 for k in link.keys():
159                     if k.split("_")[0] != k.split("_")[1]:
160                         channel_availability = []
161                         for signal_freq in self._spectral_info.
    frequency:
162                             signal_freq = signal_freq / 1e9
163                             f_min = signal_freq - spacing / 2
164                             f_max = signal_freq + spacing / 2
165                             available = True
166                             for key, value in link.items():
167                                 available_min_freq = value[argmin(
    abs_np(f_min - value))]
168                                 available_max_freq = value[argmin(
    abs_np(f_max - value))]
169                                 if abs_np(available_min_freq - f_min)
     < 6.25 and abs_np(
170                                         available_max_freq - f_max) <
     6.25:
171                                     next_freq = available_min_freq
172                                     while next_freq <
    available_max_freq:
173                                         next_freq = next_freq + 12.5
174                                         if not (next_freq in value):
175                                             available = False
176                                             break
177                                 else:
178                                     available = False
179                             channel_availability.append(available)
180                         links_availability[key] =
    channel_availability
181 
182         pathIndex = []
183 
184         for path in self._paths:
185             path_as_a_string = ""
186             for idx, element in enumerate(path):
187                 path_as_a_string += element
188                 if idx != len(path) - 1:
```

```
189                     path_as_a_string += '_'
190             pathIndex.append(path_as_a_string)
191 
192         n_mod_ch = len(self._spectral_info.frequency)
193         columns = [f'CH-{i + 1}' for i in range(n_mod_ch)]
194 
195         rs = DataFrame()
196 
197         for i_path, path in enumerate(self._paths):
198             df = DataFrame(index=[pathIndex[i_path]])
199             temp = path[0]
200             is_path_available = links_availability[path[0] + '_' +
    path[1]]
201             for next in path[1:]:
202                 is_path_available = [a and b for a, b in zip(
    is_path_available, links_availability[temp + '_' + next])]
203                 temp = next
204             df[columns] = [is_path_available]
205             rs = rs.append(df)
206         rs.to_csv("./results/routing_space.csv")
207 
208         self._routing_space = rs
209 
210     def update_rs_path_status(self, path, channel):
211         self._routing_space[f"CH-{channel}"].loc[path] = False
212         return self._routing_space
213 
214     def translate_path_to_onc(self, custom_path):
215         nodes = custom_path.split("_")
216         path = []
217         temp = nodes[0]
218 
219         for node in nodes[1:]:
220             custom_link_name = self._links_custom_to_ip[temp + '_' +
    node]
221             link = [custom_link_name.split("-")[0], custom_link_name.
    split("-")[1]]
222             temp = node
223             path.append(link)
224         return path
225 
226     def get_free_transceiver(self):
227         for dev_id, dev in enumerate(self._nodes):
228             if "ports" in self._nodes[dev]:
229                 for port in self._nodes[dev]["ports"]:
230                     if port["isEnabled"] == True:
231                         port["isEnabled"] = False
232                         print(port["port"], dev)
233                         return port["port"], dev
```

```
234
235     @property
236     def nodes(self):
237         return self._nodes
238
239
240 if __name__ == '__main__':
241     vt = VirtualTopology()
242     vt.build_routing_space()
```

A.3 httpHandler.py

```
1  from asyncio import sleep
2
3  import requests
4  from requests.auth import HTTPBasicAuth
5  import json
6  from pathlib import Path
7
8  root = Path(__file__).parent.parent
9
10 DT_HOST = "localhost"
11
12 class HttpUtils:
13     @staticmethod
14     def post_intent_on_dt(src, dst, suggested_path=None):
15
16         body = {
17             "src": src,
18             "dst": dst
19         }
20
21         if suggested_path:
22             body["suggested_path"] = suggested_path
23
24         try:
25             response = requests.post(f"http://{DT_HOST}:5001/api-v0/
    modulation-formats", json=body)
26             print(response)
27             return response.json()
28         except requests.exceptions.HTTPError as errh:
29             print(errh)
30         except requests.exceptions.ConnectionError as errc:
31             print(errc)
32         except requests.exceptions.Timeout as errt:
```

```
33              print(errt)
34          except requests.exceptions.RequestException as err:
35              print(err)
36
37          return -1
38
39      @staticmethod
40      def get_central_frequency():
41          try:
42              response = requests.get(f"http://{HOST}:8181/onos/
    newopticalrest-app/newoptical/central-frequency",
43                                      auth=HTTPBasicAuth('karaf', '
    karaf'))
44              return response.json()
45          except requests.exceptions.HTTPError as errh:
46              print(errh)
47          except requests.exceptions.ConnectionError as errc:
48              print(errc)
49          except requests.exceptions.Timeout as errt:
50              print(errt)
51          except requests.exceptions.RequestException as err:
52              print(err)
53          return -1
54
55      @staticmethod
56      def get_ports_on_dev_id(device_id):
57          try:
58              response = requests.get(f"http://{HOST}:8181/onos/v1/
    devices/{device_id}/ports",
59                                      auth=HTTPBasicAuth('karaf', '
    karaf'))
60              return response.json()
61          except requests.exceptions.HTTPError as errh:
62              print(errh)
63          except requests.exceptions.ConnectionError as errc:
64              print(errc)
65          except requests.exceptions.Timeout as errt:
66              print(errt)
67          except requests.exceptions.RequestException as err:
68              print(err)
69          return -1
70
71      @staticmethod
72      def post_intent_on_onc(mf, spacing_multiplier, path):
73          links = []
74          print("spacing mux: ", spacing_multiplier)
75
76          with open(root / 'resources/ports_lab_fix.json ') as
    json_file:
```

```
77              ports_fix = json.load(json_file)
78
79          src_dst = ports_fix[str(int(spacing_multiplier))]
80          for link in path:
81              json_element = {
82                  "src": link[0],
83                  "dst": link[1]
84              }
85              links.append(json_element)
86              if link[1].split("/")[0] == "netconf:192.168.88.31:830":
87                  json_element = {
88                      "src": "netconf:192.168.88.31:830/3001",
89                      "dst": "netconf:192.168.88.31:830/3001"
90                  }
91                  links.append(json_element)
92
93              if link[1].split("/")[0] == "netconf:192.168.88.32:830":
94                  json_element = {
95                      "src": "netconf:192.168.88.32:830/5210",
96                      "dst": "netconf:192.168.88.32:830/4110"
97                  }
98                  links.append(json_element)
99
100             if link[1].split("/")[0] == "netconf:192.168.88.33:830":
101                 json_element = {
102                     "src": "netconf:192.168.88.33:830/3001",
103                     "dst": "netconf:192.168.88.33:830/3001"
104                 }
105                 links.append(json_element)
106
107         ingress_point = {
108             "device": src_dst[0]["src"].split("/")[0],
109             "port": src_dst[0]["src"].split("/")[1]
110         }
111         egress_point = {
112             "device": src_dst[1]["dst"].split("/")[0],
113             "port": src_dst[1]["dst"].split("/")[1]
114         }
115
116         links[0] = src_dst[0]
117         links[-1] = src_dst[1]
118
119         body = {
120             "appId": "org.onosproject.newoptical",
121             "ingressPoint": ingress_point,
122             "egressPoint": egress_point,
123             "bidirectional": False,
124             "signal": {
125                 "channelSpacing": "CHL_50GHZ",
```

```
126                 "gridType": "DWDM",
127                 "spacingMultiplier": spacing_multiplier,
128                 "slotGranularity": 4
129             },
130             "suggestedPath": {
131                 "links": links
132             },
133             "modulationFormat": mf
134 
135         }
136 
137         print(body)
138         response = requests.post(f"http://{HOST}:8181/onos/
    newopticalrest-app/newoptical/intents", json=body,
139                                  auth=HTTPBasicAuth('karaf', 'karaf')
    )
140 
141         print(response.status_code)
142         if response.status_code != 200 & response.status_code != 201:
143             return response.status_code
144         return response.json()
145 
146     def request_free_channel_on_paths(self, paths_json):
147         try:
148             response = requests.post(f"http://{HOST}:8181/onos/
    newopticalrest-app/newoptical/path-status",
149                                      json=paths_json, auth=
    HTTPBasicAuth('karaf', 'karaf'))
150             return response.json()
151         except requests.exceptions.HTTPError as errh:
152             print(errh)
153         except requests.exceptions.ConnectionError as errc:
154             print(errc)
155         except requests.exceptions.Timeout as errt:
156             print(errt)
157         except requests.exceptions.RequestException as err:
158             print(err)
159         return -1
160 
161     def get_all_free_channel(self):
162         try:
163             response = requests.get(f"http://{HOST}:8181/onos/
    newopticalrest-app/newoptical/links-status",
164                                     auth=HTTPBasicAuth('karaf', '
    karaf'))
165             return response.json()
166         except requests.exceptions.HTTPError as errh:
167             print(errh)
168         except requests.exceptions.ConnectionError as errc:
```

```
169             print(errc)
170         except requests.exceptions.Timeout as errt:
171             print(errt)
172         except requests.exceptions.RequestException as err:
173             print(err)
174 
175     def get_devices_from_onc(self):
176         try:
177             return requests.get(f"http://{HOST}:8181/onos/v1/devices"
    ,
178                                 auth=HTTPBasicAuth('karaf', 'karaf'))
    .json()
179         except requests.exceptions.HTTPError as errh:
180             print(errh)
181         except requests.exceptions.ConnectionError as errc:
182             print(errc)
183         except requests.exceptions.Timeout as errt:
184             print(errt)
185         except requests.exceptions.RequestException as err:
186             print(err)
187 
188     @staticmethod
189     def get_intents_from_onc():
190         try:
191             return requests.get(f"http://{HOST}:8181/onos/v1/intents"
    ,
192                                 auth=HTTPBasicAuth('karaf', 'karaf'))
    .json()["intents"]
193         except requests.exceptions.HTTPError as errh:
194             print(errh)
195         except requests.exceptions.ConnectionError as errc:
196             print(errc)
197         except requests.exceptions.Timeout as errt:
198             print(errt)
199         except requests.exceptions.RequestException as err:
200             print(err)
201 
202     @staticmethod
203     def delete_intent(key=None, application_id="org.onosproject.
    optical-rest"):
204         try:
205             res = requests.delete(f"http://{HOST}:8181/onos/v1/
    intents/{application_id}/{key}",
206                                   auth=HTTPBasicAuth('karaf', 'karaf'
    )).json()
207             return res
208         except requests.exceptions.HTTPError as errh:
209             print(errh)
210         except requests.exceptions.ConnectionError as errc:
```

```
211             print(errc)
212         except requests.exceptions.Timeout as errt:
213             print(errt)
214         except requests.exceptions.RequestException as err:
215             print(err)
216 
217     def get_links_from_onc(self):
218         return requests.get(f"http://{HOST}:8181/onos/v1/links",
219                             auth=HTTPBasicAuth('karaf', 'karaf'))
220 
221     @staticmethod
222     def post_topology_on_dt(vt):
223 
224         try:
225             response = requests.post(f"http://{DT_HOST}:5001/api-v0/
    virtual-topologies", json=vt)
226 
227             return response.json()
228         except requests.exceptions.HTTPError as errh:
229             print(errh)
230         except requests.exceptions.ConnectionError as errc:
231             print(errc)
232         except requests.exceptions.Timeout as errt:
233             print(errt)
234         except requests.exceptions.RequestException as err:
235             print(err)
236         return -1
```

A.4 database.py

```
1  from json import load
2  from pymongo import MongoClient
3  from pathlib import Path
4  import datetime
5  from re import IGNORECASE, compile
6  from bson.objectid import ObjectId
7 
8  MONGO_HOST = "localhost"
9  MONGO_PORT = "27017"
10 MONGO_DB = "logger"
11 MONGO_USER = "root"
12 MONGO_PASS = "example"
13 
14 root = Path(__file__).parent
15 
```

```
16
17 class Database:
18     def __init__(self):
19         self._db = None
20
21     def connect_db(self, database=MONGO_DB):
22         uri = "mongodb://{}:{}@{}:{}/{}?authSource=admin".format(
   MONGO_USER, MONGO_PASS, MONGO_HOST, MONGO_PORT,
23
   database)
24         client = MongoClient(uri)
25         self._db = client.admin
26
27     def save_paths(self, paths):
28         logger = self._db.logger
29         for path in paths["allocatedPaths"]:
30             allocated_path = {
31                 "path": path["path"],
32                 "bitRate": path["bitRate"],
33                 "modulationFormat": path["modulationFormat"],
34                 "channel": path["channel"],
35                 "time": datetime.datetime.utcnow()
36             }
37             post_id = logger.insert_one(allocated_path).inserted_id
38         return 0
39
40     def find_path(self, sub_path):
41         logger = self._db.logger
42         regx = compile("^.*" + sub_path + ".*", IGNORECASE)
43         mon_paths = logger.find({"path": regx})
44         paths = []
45         for path in mon_paths:
46             paths.append(path)
47         return paths
48
49     def delete_path_from_id(self, _id):
50         logger = self._db.logger
51         logger.delete_one({'_id': ObjectId(_id)})
52         return _id
53
54     def save_spectral_info(self):
55         print("Saving spectral info to db")
56         spectral_info_db = self._db.spectral_info
57         with open("resources/triangular_network_launch_spectrum.json"
   , "r") as read_file:
58             spectral_info_json = load(read_file)
59         spectral_info_db.update_one({'_id': 0}, {"$set":
   spectral_info_json}, upsert=True)
60         return
```

```
61
62      def get_spectral_info(self):
63          spectral_info_db = self._db.spectral_info
64          return spectral_info_db.find_one({"_id": 0})
```

Bibliography

[1] Roland Wenzlhuemer. «The development of telegraphy, 1870–1900: a European perspective on a world history challenge». In: *History Compass* 5.5 (2007), pp. 1720–1742 (cit. on p. 2).

[2] W.H. Page and A.W. Page. *The World's Work*. Doubleday, Page & Company, 1907. URL: `https://books.google.it/books?id=3IfNAAAAMAAJ` (cit. on p. 2).

[3] Peter L Dorlan. *An introduction to computer networks*. Autoedición, 2016 (cit. on p. 3).

[4] Yadong Li, Danlan Li, Wenqiang Cui, and Rui Zhang. «Research based on OSI model». In: *2011 IEEE 3rd International Conference on Communication Software and Networks*. IEEE. 2011, pp. 554–557 (cit. on pp. 3, 19).

[5] Robert W Tkach. «Scaling optical communications for the next decade and beyond». In: *Bell Labs Technical Journal* 14.4 (2010), pp. 3–9 (cit. on p. 5).

[6] Jingchi Cheng, Chongjin Xie, Yizhao Chen, Xi Chen, Ming Tang, and Songnian Fu. «Comparison of coherent and IMDD transceivers for intra datacenter optical interconnects». In: *2019 Optical Fiber Communications Conference and Exhibition (OFC)*. IEEE. 2019, pp. 1–3 (cit. on p. 4).

[7] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. «A survey on software-defined networking». In: *IEEE Communications Surveys & Tutorials* 17.1 (2014), pp. 27–51 (cit. on p. 4).

[8] James W Paulson, Giancarlo Succi, and Armin Eberlein. «An empirical study of open-source and closed-source software products». In: *IEEE transactions on software engineering* 30.4 (2004), pp. 246–256 (cit. on p. 6).

[9] Guido Schryen and Rouven Kadura. «Open source vs. closed source software: towards measuring security». In: *Proceedings of the 2009 ACM symposium on Applied Computing*. 2009, pp. 2016–2023 (cit. on p. 6).

[10] Srinivasan Raghunathan, Ashutosh Prasad, Birendra K Mishra, and Hsihui Chang. «Open source versus closed source: software quality in monopoly and competitive markets». In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 35.6 (2005), pp. 903–918 (cit. on p. 6).

[11] Emilio Riccardi, Paul Gunning, Óscar González de Dios, Marco Quagliotti, Vıctor López, and Andrew Lord. «An operator view on the introduction of white boxes into optical networks». In: *Journal of Lightwave Technology* 36.15 (2018), pp. 3062–3072 (cit. on pp. 6, 20).

[12] Gangxiang Shen and Rodney S Tucker. «Translucent optical networks: the way forward [topics in optical communications]». In: *IEEE Communications Magazine* 45.2 (2007), pp. 48–54 (cit. on p. 8).

[13] Emmanuel Desurvire and Michael N Zervas. «Erbium-doped fiber amplifiers: principles and applications». In: *Physics Today* 48.2 (1995), p. 56 (cit. on p. 10).

[14] AAM Saleh, RM Jopson, JD Evankow, and J Aspell. «Modeling of gain in erbium-doped fiber amplifiers». In: *IEEE Photonics Technology Letters* 2.10 (1990), pp. 714–717 (cit. on p. 10).

[15] Yongcheng Li, Li Gao, Gangxiang Shen, and Limei Peng. «Impact of ROADM colorless, directionless, and contentionless (CDC) features on optical network performance». In: *Journal of Optical Communications and Networking* 4.11 (2012), B58–B67 (cit. on p. 12).

[16] Jonathan Homa and Krishna Bala. «ROADM architectures and their enabling WSS technology». In: *IEEE Communications Magazine* 46.7 (2008), pp. 150–154 (cit. on p. 12).

[17] R Senguttuvan, S Bhattacharya, and A Chatterjee. «Design considerations and effect of manufacturing process variations on UWB transceiver specifications». In: *2005 IEEE International Conference on Ultra-Wideband*. IEEE. 2005, pp. 553–558 (cit. on p. 13).

[18] JE Johnson, DR Stauffer, and K Gass. «„Implementation Agreement for Integrated Dual Polarization Intradyne Coherent Receivers “, Optical Internetworking Forum». In: *Techn. Ber., Nov* (2013) (cit. on p. 13).

[19] Schwartz Mischa. «Information transmission modulation and noise». In: (1990) (cit. on p. 13).

[20] Mischa Schwartz. *Information Transmission, Modulation and Noise*. McGraw-Hill, New York, 1990 (cit. on p. 16).

[21] Govind P Agrawal. *Fiber-optic communication systems*. John Wiley & Sons, 2012 (cit. on p. 17).

[22] Peter J Winzer and Ren-Jean Essiambre. «Advanced modulation formats for high-capacity optical transport networks». In: *Journal of Lightwave Technology* 24.12 (2006), pp. 4711–4728 (cit. on p. 16).

[23] Andrew S Tanenbaum. *Computer networks*. Pearson Education India, 2003 (cit. on p. 17).

[24] Kaida Kaeval, Tobias Fehenberger, Jim Zou, Sander Lars Jansen, Klaus Grobe, Helmut Griesser, Jörg-Peter Elbers, Marko Tikas, and Gert Jervan. «QoT assessment of the optical spectrum as a service in disaggregated network scenarios». In: *Journal of Optical Communications and Networking* 13.10 (2021), E1–E12 (cit. on p. 18).

[25] Truong-Xuan Do and Younghan Kim. «Control and data plane separation architecture for supporting multicast listeners over distributed mobility management». In: *ICT Express* 3.2 (2017), pp. 90–95 (cit. on p. 18).

[26] Steven Gringeri, Nabil Bitar, and Tiejun J Xia. «Extending software defined network principles to include optical transport». In: *IEEE Communications Magazine* 51.3 (2013), pp. 32–40 (cit. on p. 19).

[27] Laia Nadal et al. «SDN-enabled S-BVT for disaggregated networks: design, implementation and cost analysis». In: *Journal of Lightwave Technology* 38.11 (2020), pp. 3037–3043 (cit. on p. 19).

[28] Christian Guillemot et al. «Transparent optical packet switching: The European ACTS KEOPS project approach». In: *Journal of lightwave technology* 16.12 (1998), p. 2117 (cit. on p. 19).

[29] Jose Alberto Hernandez, Marco Quagliotti, Emilio Riccardi, Victor Lopez, Oscar Gonzalez de Dios, and Ramon Casellas. «A techno-economic study of optical network disaggregation employing open source software business models for metropolitan area networks». In: *IEEE Communications Magazine* 58.5 (2020), pp. 40–46 (cit. on p. 20).

[30] Fei Hu, Qi Hao, and Ke Bao. «A survey on software-defined network and openflow: From concept to implementation». In: *IEEE Communications Surveys & Tutorials* 16.4 (2014), pp. 2181–2206 (cit. on p. 20).

[31] Reza Nejabati, Eduard Escalona, Shuping Peng, and Dimitra Simeonidou. «Optical network virtualization». In: *15th International Conference on Optical Network Design and Modeling-ONDM 2011*. IEEE. 2011, pp. 1–5 (cit. on pp. 21, 22).

[32] Vittorio Curri. «GNPy model of the physical layer for open and disaggregated optical networking». In: *Journal of Optical Communications and Networking* 14.6 (2022), pp. C92–C104 (cit. on pp. 21, 24, 25, 28–30).

[33] Ana Isabel Montoya-Munoz, Daniela Casas-Velasco, Felipe Estrada-Solano, Oscar Mauricio Caicedo Rendon, and Nelson L Saldanha da Fonseca. «An approach based on Yet Another Next Generation for software-defined networking management». In: *International Journal of Communication Systems* 34.11 (2021), e4855 (cit. on p. 22).

[34] *http://www.openconfig.net* (cit. on p. 22).

[35] Francesco Paolucci and Andrea Sgambelluri. «Telemetry in disaggregated optical networks». In: *2020 International Conference on Optical Network Design and Modeling (ONDM)*. IEEE. 2020, pp. 1–3 (cit. on p. 22).

[36] Francesco Paolucci, Andrea Sgambelluri, Filippo Cugini, and Piero Castoldi. «Network telemetry streaming services in SDN-based disaggregated optical networks». In: *Journal of Lightwave Technology* 36.15 (2018), pp. 3142–3149 (cit. on p. 22).

[37] Andrea Sgambelluri, Francesco Paolucci, and Filippo Cugini. «Telemetry-driven validation of operational modes in OpenConfig disaggregated networks». In: (2019) (cit. on p. 22).

[38] *http://www.openroadm.org* (cit. on p. 22).

[39] Ramon Casellas, Alessio Giorgetti, Roberto Morro, Ricardo Martinez, Ricard Vilalta, and Raul Muñoz. «Virtualization of disaggregated optical networks with open data models in support of network slicing». In: *Journal of Optical Communications and Networking* 12.2 (2020), A144–A154 (cit. on pp. 23, 24).

[40] Andrea Sgambelluri, Alessio Giorgetti, Davide Scano, Filippo Cugini, and Francesco Paolucci. «OpenConfig and OpenROADM automation of operational modes in disaggregated optical networks». In: *IEEE Access* 8 (2020), pp. 190094–190107 (cit. on p. 23).

[41] Vittorio Curri. «Software-defined WDM optical transport in disaggregated open optical networks». In: *2020 22nd International Conference on Transparent Optical Networks (ICTON)*. IEEE. 2020, pp. 1–4 (cit. on p. 24).

[42] Mark Filer, Mattia Cantono, Alessio Ferrari, Gert Grammel, Gabriele Galimberti, and Vittorio Curri. «Multi-Vendor Experimental Validation of an Open Source QoT Estimator for Optical Networks». In: *J. Lightw. Technol.* 36.15 (Aug. 2018), pp. 3073–3082. DOI: 10.1109/JLT.2018.2818406 (cit. on p. 26).

[43] Ori Gerstel and Victor Lopez. «The need for SDN in orchestration of IP over optical multi-vendor networks». In: *2015 European Conference on Optical Communication (ECOC)*. IEEE. 2015, pp. 1–3 (cit. on p. 27).

[44] Ramon Casellas, Ricardo Martınez, Ricard Vilalta, and Raül Muñoz. «Metro-haul: SDN control and orchestration of disaggregated optical networks with model-driven development». In: *2018 20th International Conference on Transparent Optical Networks (ICTON)*. IEEE. 2018, pp. 1–4 (cit. on p. 28).

[45] Alessio Ferrari, Giacomo Borraccini, and Vittorio Curri. «Observing the generalized SNR statistics induced by gain/loss uncertainties». In: (2019) (cit. on p. 29).

[46] Vittorio Curri, Andrea Carena, Andrea Arduino, Gabriella Bosco, Pierluigi Poggiolini, Antonino Nespola, and Fabrizio Forghieri. «Design strategies and merit of system parameters for uniform uncompensated links supporting Nyquist-WDM transmission». In: *Journal of Lightwave Technology* 33.18 (2015), pp. 3921–3932 (cit. on p. 29).

[47] Rui Manuel Morais and João Pedro. «Machine learning models for estimating quality of transmission in DWDM networks». In: *Journal of Optical Communications and Networking* 10.10 (2018), pp. D84–D99 (cit. on p. 29).

[48] Francesco Musumeci, Cristina Rottondi, Avishek Nag, Irene Macaluso, Darko Zibar, Marco Ruffini, and Massimo Tornatore. «An overview on application of machine learning techniques in optical networks». In: *IEEE Communications Surveys & Tutorials* 21.2 (2018), pp. 1383–1408 (cit. on p. 29).

[49] Javier Mata, Ignacio de Miguel, Ramon J Duran, Noemı Merayo, Sandeep Kumar Singh, Admela Jukan, and Mohit Chamania. «Artificial intelligence (AI) methods in optical networks: A comprehensive survey». In: *Optical switching and networking* 28 (2018), pp. 43–57 (cit. on p. 29).

[50] Andy Bierman, Martin Bjorklund, and Kent Watsen. *RESTCONF protocol*. Tech. rep. 2017 (cit. on p. 31).

[51] Martin Bjorklund. *YANG-a data modeling language for the network configuration protocol (NETCONF)*. Tech. rep. 2010 (cit. on p. 31).

[52] Martin Bjorklund. *The YANG 1.1 data modeling language*. Tech. rep. 2016 (cit. on p. 31).

[53] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. *Hypertext transfer protocol–HTTP/1.1*. Tech. rep. 1999 (cit. on p. 31).

[54] Marc De Leenheer, Yuta Higuchi, and Guru Parulkar. «An open controller for the disaggregated optical network». In: *2018 International Conference on Optical Network Design and Modeling (ONDM)*. IEEE. 2018, pp. 230–233 (cit. on p. 37).

[55] *https://opennetworking.org/onos/* (cit. on p. 38).

[56] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java language specification*. Addison-Wesley Professional, 2000 (cit. on p. 38).

[57] *https://maven.apache.org* (cit. on p. 38).

[58] *https://karaf.apache.org* (cit. on p. 38).

[59] Karl Pauls, David Savage, Stuart McCulloch, and Richard Hall. *OSGi in action: Creating modular applications in Java*. Simon and Schuster, 2011 (cit. on p. 38).

[60] Alessio Giorgetti, Ramon Casellas, Roberto Morro, Andrea Campanella, and Piero Castoldi. «ONOS-controlled disaggregated optical networks». In: *2019 Optical Fiber Communications Conference and Exhibition (OFC)*. IEEE. 2019, pp. 1–3 (cit. on pp. 38, 39).

[61] Andrea Campanella et al. «ODTN: Open Disaggregated Transport Network. Discovery and control of a disaggregated optical network through open source software and open APIs.» In: *Optical Fiber Communication Conference*. Optical Society of America. 2019, M3Z–4 (cit. on p. 38).

[62] Andrea Campanella. «Intent based network operations». In: *2019 Optical Fiber Communications Conference and Exhibition (OFC)*. IEEE. 2019, pp. 1–3 (cit. on p. 38).

[63] *GitHub repository of GNPy*. Version v2.0. DOI: 10.5281/zenodo.3458319. DOI: 10.5281/zenodo.3458319. URL: https://doi.org/10.5281/zenodo.3458319 (cit. on p. 39).

[64] *https://telecominfraproject.com* (cit. on p. 39).

[65] Alessio Ferrari, Mark Filer, Karthikeyan Balasubramanian, Yawei Yin, Esther Le Rouzic, Jan Kundrát, Gert Grammel, Gabriele Galimberti, and Vittorio Curri. «GNPy: an open source application for physical layer aware open optical networks». In: *Journal of Optical Communications and Networking* 12.6 (2020), pp. C31–C40 (cit. on p. 40).

[66] Jean-Luc Auge, Vittorio Curri, and Esther Le Rouzic. «Open design for multi-vendor optical networks». In: *Optical Fiber Communication Conference*. Optical Society of America. 2019, Th1I–2 (cit. on p. 39).

[67] $https://www.itu.int/rec/dologin_pub.asp?lang = eid = T - REC - G.694.1 - 202010 - I!!PDF - Etype = items$ (cit. on p. 41).

[68] Claus Fuhrer, Jan Erik Solem, and Olivier Verdier. *Scientific Computing with Python: High-performance scientific computing with NumPy, SciPy, and pandas*. Packt Publishing Ltd, 2021 (cit. on p. 42).

[69] C Manso, R Muñoz, N Yoshikane, R Casellas, R Vilalta, R Martınez, T Tsuritani, and I Morita. «TAPI-enabled SDN control for partially disaggregated multi-domain (OLS) and multi-layer (WDM over SDM) optical networks». In: *Journal of Optical Communications and Networking* 13.1 (2021), A21–A33 (cit. on p. 43).

[70] Giacomo Borraccini et al. «Independent Data and Control Planes in Partially Disaggregated Optical Networks: an Experimental Proof of Concept». In: *IEEE-TNSM* (submitted) (cit. on pp. 46, 52, 53).

[71] Quan Pham-Van et al. «Demonstration of alarm correlation in partially disaggregated optical networks». In: *Optical Fiber Communication Conference*. Optica Publishing Group. 2020, M3Z–6 (cit. on pp. 47, 51).

[72] Masaki Shiraiwa et al. «Experimental demonstration of disaggregated emergency optical system for quick disaster recovery». In: *Journal of Lightwave Technology* 36.15 (2018), pp. 3083–3096 (cit. on p. 51).

[73] Kayol S Mayer, Rossano P Pinto, Jonathan A Soares, Dalton S Arantes, Christian E Rothenberg, Vinicius Cavalcante, Leonardo L Santos, Filipe D Moraes, and Darli AA Mello. «Demonstration of ML-assisted soft-failure localization based on network digital twins». In: *Journal of Lightwave Technology* 40.14 (2022), pp. 4514–4520 (cit. on p. 51).

[74] Shaoqiang Wang, DongSheng Xu, and ShiLiang Yan. «Analysis and application of Wireshark in TCP/IP protocol teaching». In: *2010 International Conference on E-Health Networking Digital Ecosystems and Technologies (EDT)*. Vol. 2. IEEE. 2010, pp. 269–272 (cit. on p. 54).

[75] Andrew Tanenbaum. *Modern operating systems*. Pearson Education, Inc., 2009 (cit. on p. 57).