

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

MASTER'S DEGREE

IN

TELECOMMUNICATIONS ENGINEERING

ORCHESTRATION
OF CONNECTIVITY RESOURCES
IN ELASTIC OPTICAL NETWORKS

Master Thesis

in

Laboratory of Advanced Networking M

Supervisor

Prof. GIANLUCA DAVOLI

Candidate

RAFFAELE DI TOMMASO

Co-supervisor

Prof. LUIZ H. BONANI

SESSION I

ACADEMIC YEAR 2024/2025

*«Considerate la vostra semenza:
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza.»*

Contents

Abstract	ix
1 Introduction	1
2 Overview	3
2.1 Traditional WDM optical networks	4
2.1.1 From point-to-point to WDM optical networks	4
2.1.2 Optical paths and constraints	4
2.1.3 Architecture and enabling technologies	5
2.1.4 Limitations of WDM networks	6
2.2 Elastic Optical Networks	7
2.2.1 From WDM to Elastic Optical Networks	7
2.2.2 Spectrum allocation in EONs	7
2.2.3 EONs enabling technologies	8
2.3 The RMSA problem	13
2.3.1 Routing	13
2.3.2 Modulation format	14
2.3.3 Wavelength and Spectrum assignment	14
2.3.4 The fragmentation problem	16
2.4 The future of EONs	17
2.4.1 Multi-Band Optical Networks	17
2.4.2 Space Division Multiplexed Optical Networks	20
2.4.3 Disaggregated optical networks	23
3 EON simulator	25
3.1 Core Classes	26
3.1.1 Node	26

3.1.2	Link	28
3.1.3	Spectrum	29
3.1.4	Route	30
3.1.5	Topology	32
3.1.6	Connection	33
3.2	Control Plane	35
3.2.1	Control Plane logic	35
3.2.2	Path computation	35
3.2.3	Costs management	37
3.2.4	Spectrum management	38
3.3	Spectrum allocation algorithms	40
3.3.1	Current implementations in EONs	40
3.3.2	Wavelength conversion algorithms	42
3.3.3	Policy	43
3.3.4	Logic	43
3.3.5	Spectrum usage	44
3.4	Allocation examples	44
3.4.1	First Fit base with conversion	47
3.4.2	First Fit best effort with can_break logic	47
3.4.3	First Fit best effort with no_break logic	48
3.4.4	First Fit stubborn with can_break	49
3.4.5	Random Fit algorithms	50
3.4.6	Longest Fit	52
4	Simulator engine and results	55
4.1	Simulator structure	55
4.2	Simulation engine	57
4.3	Control plane	60
4.4	Results	61
4.4.1	Benchmark comparison	63
4.4.2	Comparison of First Fit algorithms	64
4.4.3	Comparison of Random Fit algorithms	68
4.4.4	Comparison of Longest Fit algorithms	70
4.4.5	Test on a real-world topology	72
4.5	Wavelength conversion as a service	76

5	Conclusion and future work	77
5.0.1	Conclusions	77
5.0.2	Future work	78
A	Simulator components	81
A.1	Core components	81
A.1.1	Node	81
A.1.2	Link	84
A.1.3	Spectrum	84
A.1.4	Connection	84
A.1.5	Route	85
A.1.6	Topology	85

Abstract

This thesis provides a description of the evolution of optical networks over the years, starting from the first point-to-point communications, moving to Wavelength Division Multiplexed networks which led to the implementations of nowadays Elastic Optical Networks. The work proceeds with the description of the development of an EON simulator that we used to investigate the performance of several spectrum allocation algorithms with the possibility to exploit wavelength conversion. The current implementations comprehend different version of the already known First Fit and Random Fit algorithms and a novel approach that we called Longest Fit. The first two models present three versions each, one that performs link-by-link spectrum assignment called base, one that introduces the spectrum blocks logic called best_effort and one that allows backtracking in allocations called stubborn. Longest Fit instead only has best_effort and stubborn versions. Each of these has also a special parameter that allows or denies the possibility to break spectrum blocks. Simulations are provided on both a custom and a real-world network topology.

Chapter 1

Introduction

As we can state from the very recent years we are entering an era in which connectivity will re-shape several aspects of our lives. Emerging technologies such as Artificial intelligence, Internet of Things, Autonomous transportation, Extended Reality and many more will influence our way to interact with each other and therefore the way we communicate, how we learn, work and behave, carrying on the advancement of society that communication technologies have brought throughout the last decades enhancing the exchange of ideas breaking the limits of distances. But we also know that new technologies always come with a cost; they define new stringent requirements for our network infrastructures leading to a urge for the identification of architectures and frameworks that allow the development of next-generation networks which must be able to satisfy the requirements of new services and applications in terms of latency, throughput, scalability and energy efficiency. All of this brings a particular stress to what we can think of as the nervous system of our digital world: the optical backbone.

During the years optical fiber technologies have faced continuous evolution in order to provide more and more sophisticated and flexible infrastructure for data delivery starting from Wavelength Division Multiplexing (WDM) systems that provided connectivity by reserving an optical channel of 50 or 100 GHz in all the route between the endpoints, to which we refer to as lightpath, This type of approach had several limitations that were overcome thanks to the introduction of Elastic Optical Networks (EON) that introduced a flexible grid for spectrum allocation. In fact, unlike conventional systems, which allocate fixed spectral bandwidths to each optical channel, of typically 50 GHz, EONs dy-

namically assign spectrum resources based on actual service demands. This is achieved through the use of fine-grained spectrum slices and software-defined transceivers that can adapt their modulation format, bit rate, and spectral width adapting to the incoming connectivity requests. This property takes the name of elasticity and allows for more efficient utilization of the optical spectrum, reduces fragmentation, and improves scalability in response to increasing and varying bandwidth requirements and this is a crucial aspect in a scenario in which the dynamicity and heterogeneity of Internet traffic is still growing exponentially, therefore setting further requirements for the flexibility of optical communication systems. Those previously mentioned characteristics make EONs particularly well-suited for modern, high-capacity network environments with evolving and unpredictable traffic demands.

This work is structured as follows: chapter 2 presents an overview of the evolution of optical fiber systems from the first experiments to nowadays state of the art, highlighting the need for a continuous evolution and development of fiber technologies. Chapter 3 introduces the Elastic Optical Network simulator that I contributed developing describing its structure and the core classes used to emulate the physical layer, simulation parametrization and connectivity requests. Chapter 4 presents the simulation engine, so how the connection requests are managed, and the control plane logic including all the spectrum allocation algorithms and their corresponding logic and parameters. In the same chapter are also reported the results obtained from the simulations on a custom and a real-world network topology. Finally chapter 5 highlights the conclusions that we could get from the simulation results and presents a novel scenario for the application of the developed algorithms that is called WCaaS (Wavelength conversion as a service) which represents a novel possible application of wavelength conversion technologies.

Chapter 2

Overview

The history of optical fiber technologies for communication begins in the 1960s when the Dutch professor of physics Abraham Van Heel found a way to cover a bare glass fiber with a transparent coating, that we now call cladding, with a lower refraction index w.r.t. the core such that light could remain confined inside the fiber without leaking out exploiting the phenomenon of total internal reflection. Starting from that, during the years optical fiber communications have been improved with optimizations in terms of materials, bandwidth usage, data rate, signal transmission quality, noise reduction, dispersion compensation, and several other technical aspects that enabled faster, more efficient and more reliable communication. Nowadays almost all of wired communication networks exploit optical fiber technologies, therefore we can say that they play a pivotal role in our infrastructures and the ever increasing traffic demand and service needs lead to the necessity of continuous research for their optimization and development. In fact ITU reports such as [1] show that in 2024 fully 5.5 billion people are online, which represents the 68% of the total world population, a number that is continuously increasing during the years that clearly indicates an increasing trend in the request for connectivity resources.

This chapter presents an overview over the main optical network architectures and the path towards the implementation of Elastic Optical Networks that will be further discussed in this work introducing its fundamental concepts from both a theoretical and an architectural viewpoint that are integrated by the presentation of the EON simulator that I helped to develop.

2.1 Traditional WDM optical networks

2.1.1 From point-to-point to WDM optical networks

In their early implementations, optical fiber links were mainly used for short point-to-point transmissions, mainly to carry images, but the received signals suffered from severe degradation, so a poor output image quality. The previously mentioned introduction of the cladding layer, together with the development of improved purified materials for the fiber core, significantly reduced losses and enabled much longer transmission distances. However, even as signal quality and reach increased, the capacity of optical fibers became a growing concern as traffic demand was increasing more and more over the years: each fiber could still support only a single data stream, which was insufficient to satisfy the rapidly increasing traffic demand. This limitation drove the adoption of Wavelength Division Multiplexing (WDM), a technique that allows multiple signals to be transmitted simultaneously on the same fiber, each one modulated over a different frequency, or better over a distinct optical carrier. Initially, though, optical links remained confined to point-to-point connections because any routing operation required converting the optical signals into the electrical domain, processing them (e.g. signal regeneration), and then converting them back to light. Over time, researchers and network engineers recognized that optical fibers could offer more than simple end-to-end connectivity. This insight led to the emergence of a new generation of optical networks capable of performing switching and routing operations directly in the optical domain [2]. Given the performance bottlenecks imposed by repeated optical-electrical-optical conversions, maximizing the available optical bandwidth became essential. Among various proposed solutions, WDM proved to be the most effective, as it partitions the optical spectrum into discrete channels, typically spaced by 50 or 100 GHz, allowing multiple independent connections to travel simultaneously inside the same fiber, each one occupying its own assigned spectrum slot. This method of organizing the spectrum is commonly referred to as the Fixed Grid approach.

2.1.2 Optical paths and constraints

WDM optical networks work in a circuit switching mode, so for every incoming connection request that can be served, a spectrum channel, identified by

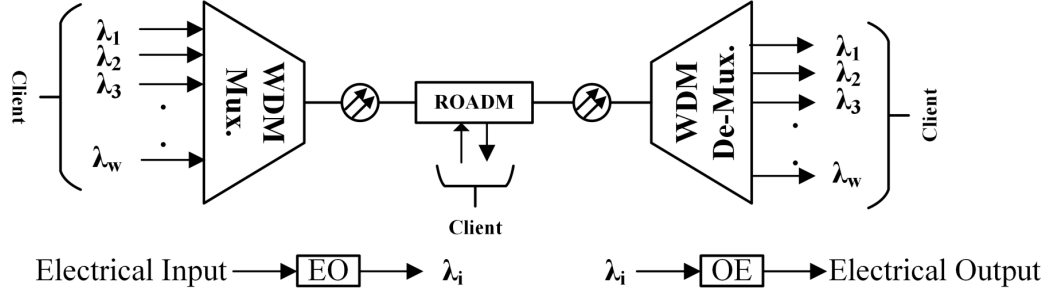


Figure 2.1: WDM based transmission with Multiplexer, ROADM, De-Multiplexer, W wavelengths ($\lambda_1, \lambda_2, \dots, \lambda_W$), Electrical to Optical Converters (EO) and Optical to Electrical converters (OE) [2]

a wavelength, is reserved for that connection in every optical link between the source and the destination nodes. This set of reserved resources for a given connection is called lightpath [3]. As previously mentioned in WDM networks the optical channel associated to a certain lightpath is associated to a wavelength that is related to the frequency over which the signal is modulated and must be the same along all the route. This necessity takes the name of Wavelength Continuity constraint while the problem of finding a path in the optical network that satisfies this constraint is called the Route and Wavelength Assignment problem (RWA). According to the ITU-T G.694.1 standard [4] by using a channel size of 50 GHz the frequencies that are allowed in WDM networks can be found by means of the following formula:

$$\nu = 193.1 \text{ THz} + n \cdot 0.05 \text{ THz}$$

where n is a positive or negative integer number including 0. The corresponding wavelength can be easily found by means of:

$$\lambda = \frac{c}{n\nu}$$

Where $c = 3 \cdot 10^8 \text{ m/s}$ is the speed of light in vacuum.

2.1.3 Architecture and enabling technologies

From figure 2.1 we can see that since to generate the signal on one wavelength we need one transmitter, with W optical channels, and therefore W possible

corresponding wavelengths, we need to have W transmitters and the signals that they generate are then multiplexed in order to be transmitted on a fiber link. At the output the WDM signal will be de-multiplexed into W distinct signals that will be provided to their respective receivers (e.g. PIN or APD photodiodes). Moreover from the same figure we can also see part of the technologies that have allowed the deployment of WDM optical networks such as Reconfigurable Optical Add/Drop Multiplexers (ROADM) that are designed to add/drop signals from or into the network and may be connected to other fiber links or to Optical Line Terminal (OLT) units. Another element that was used in WDM networks is the Optical Cross-Connect (OXC) that consists in an $N \times N$ optical switch that has N input and N output fiber links. OXCs are responsible for the routing of optical signals from one specific wavelength in the input link to the same exact wavelength in the output link due to the Wavelength Continuity constraint.

2.1.4 Limitations of WDM networks

WDM networks, also referred to as All-Optical networks, avoid the need for converting signals into the electronic domain. This provided better performance compared to earlier network architectures. However, as communication demands continued to grow in terms of throughput and flexibility, the limitations of WDM networks gradually started to appear. In particular the main drawbacks of WDM networks was the waste in spectrum resource usage, as to each connection request a full channel of 50 or 100 GHz was reserved independently from the bandwidth requirements of the specific network service and therefore there were unused spectrum holes between the channels that couldn't be used by other requests and this translated into a poor spectrum usage efficiency. Moreover if a specific service required a bandwidth that was larger w.r.t. the channel size it couldn't be served. Those limitations opened the path for studies on more flexible and efficient optical network infrastructures that led to the development of Elastic Optical Networks.

2.2 Elastic Optical Networks

2.2.1 From WDM to Elastic Optical Networks

As previously mentioned the concept of Elastic Optical Networks comes from the need to overcome the limitations of rigid Fixed-Grid WDM networks with the idea of being able to tailor the allocated bandwidth and the data rate to specific service requirements. This is done by introducing a new type of grid for spectrum assignment which is called Flexible Grid (or Flex-Grid) and changing the multiplexing technique from Dense Wavelength Division Multiplexing to Optical-Orthogonal Frequency Division Multiplexing (O-OFDM). The size of a base frequency slot is reduced to 12.5 GHz increasing the granularity for spectrum allocation and allowing the combination of several basic slots to form superchannels that may exceed the maximum size that was allowed by the Fixed Grid, enabling the support for services that require more than 50 or 100 GHz. In fact EONs were first identified as a way to go beyond data rates of 100 Gb/s that would not fit into a traditional WDM network implementing the Fixed Grid standard and simple On-Off Keying (OOK) modulation [5]. Even by pushing higher data rates inside 50 GHz channels by using a more complex modulation scheme to reach 400 Gb/s would make the signal not able to travel long distances. Another technique that was thought for achieving beyond 100 Gb/s data rates in WDM networks was the virtual concatenation of multiple channels, but also in this way the spectrum utilization was not very efficient since for every channel we had to insert guard bands while this happens only once with EONs since we have just one single spectrum slice of the desired size leading to an increased spectrum utilization.

2.2.2 Spectrum allocation in EONs

With the ITU-T Flexible Grid standard for an allocation we must specify a central frequency with a granularity of 6.25 GHz (the center of a 12.5 GHz slot) and we can do it by means of the following formula:

$$\nu = 193.1 \text{ THz} + n \cdot 0.00625 \text{ THz}$$

where n is a positive or negative integer number including 0. Moreover we need to specify a spectrum slot width that must be a multiple of a unitary

slot, therefore a multiple of 12.5 GHz and that we can express as:

$$B = 12.5 \text{ GHz} \cdot m$$

where m is a strictly positive integer number.

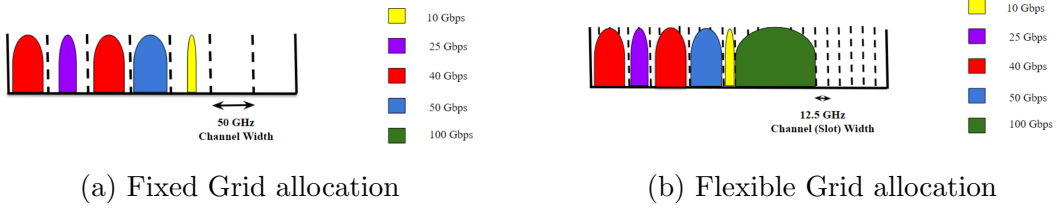


Figure 2.2: Comparison between Fixed and Flexible grids [2]

In figures 2.2a and 2.2b we can see how the usage of a Flexible Grid allocation leads to a better usage of the spectrum as in figure 2.2a we clearly see that reserving a 50 GHz channel to a service that doesn't need it leads to a waste in the usage of the spectrum, while in figure 2.2b the allocation of the same services is much more efficient. Moreover in the Flexible Grid case we can also allocate a service with a bandwidth requirement which is larger than a spectrum slot of the Fixed Grid and therefore couldn't be allocated otherwise. In the case of EON since now the resource is no more related to a wavelength as in the WDM case but to a set of spectrum slots we need now to add a constraint to the already mentioned Continuity Constraint which is the Contiguity Constraint which means that once we identify the spectrum slice to be reserved for a given connection, which will be composed of multiple contiguous unitary slots of 12.5 GHz, we need to allocate the same set of slots along the whole route. In this case the problem of finding a suitable route and spectrum slice to serve a connection request is called Routing and Spectrum Assignment problem (RSA).

2.2.3 EONs enabling technologies

In order to allow the deployment of Elastic Optical Networks the technology that was developed includes Bandwidth Variable Transponders (BVT) and Sliceable Bandwidth Variable Transponders (SBVT) in order to be able to convert a signal from the electronic realm to the optical one and vice versa

with the possibility to have a specific bandwidth on-demand. The main difference between BVTs and SBVTs is that the former when transmitting at low bit rates leaves some part of its capacity unused which led to the introduction of the latter [6] [7]. Moreover also ROADM technologies were improved in order to support bandwidth variability exploiting technologies like Micro-Electro-Mechanical Systems (MEMS), Liquid Crystals On Silicon (LCOS) or integrated photonics technologies such as Silica Planar Lightwave Circuits (PLCs), while for switching purposes Bandwidth-Variable Wavelength Cross-Connect (BV-WXC) elements were introduced. With the introduction of BVTs another important innovation was implemented; the possibility to modify the modulation format of the transmission based on the connection needs. In fact each modulation format choice represents a trade-off between spectral efficiency, so how much information we are able to transmit per bandwidth unit, and the distance that the signal can travel inside the optical fibers links. The key technology that enabled adaptive modulation formats was coherent optical detection techniques, which permitted to detect both the amplitude and the phase of the received signal [8]. This allowed us to exploit modulation schemes that use both amplitude and phase modulation therefore carrying more than one bit per symbol. For binary modulation schemes, so for those in which we represent the information using zeros and ones, if we have N quantization levels the number of bits per symbol that are carried can be found with:

$$M = \log_2(N)$$

As we increase N the symbols get closer in the constellation diagrams which means that there is less margin in terms of energy between them and therefore from the receiver viewpoint is harder to distinguish between two symbols in presence of noise, so it is clear that if we need to meet a specific Bit Error Rate (BER) requirement the distance that we can travel with a given modulation format decreases when the number of quantization levels increases. Figure 2.4 shows this relation between the modulation level (number of quantization levels) and the transmission range.

We know in fact that signals propagating inside optical fibers undergo deterioration due to linear and nonlinear effects like Self-Phase Modulation (SPM) and Cross-Phase Modulation (XPM), but also the amplitude gets affected by Four-Wave-Mixing (FWM) effect and Amplified Spontaneous Emission (ASE) noise. A complete list of all linear and nonlinear impairments in optical fibers is briefly described in figure 2.3. Those impairments accumulate as the route

Linear effects	Non-linear effects	Other effects
Fiber attenuation: optical signal power is attenuated during signal propagation in fiber Chromatic dispersion (CD): Different spectral components of the pulse travel at different group velocities in fiber causing a pulse broadening. Polarization mode dispersion (PMD): two different polarization of the signal travel at different velocities in fiber, due to imperfections and asymmetries, causing a pulse broadening	Self-phase modulation (SPM): because of the Kerr effect, the power dependence of the fiber refractive index causes a phase shift induced by the power of the signal itself Cross-phase modulation (XPM): because of the Kerr effect, the power dependence of the fiber refractive index causes a phase shift in the signal due to other signals propagating in the fiber at different carrier frequencies.	Amplified spontaneous Emission (ASE): spontaneous emissions add noise to the signal during its amplification Crosstalk: the power leak among OXC ports causes an interference among signals

Figure 2.3: Complete list of physical linear and nonlinear impairments in optical fiber signals propagation [9]

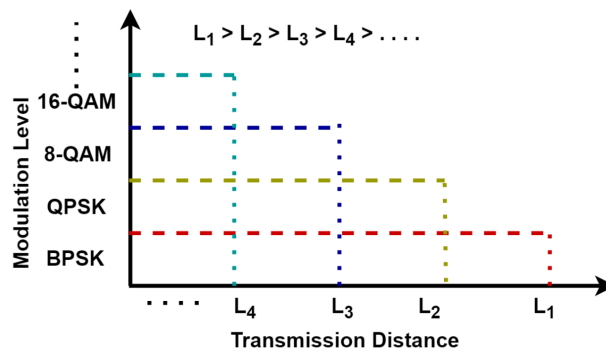


Figure 2.4: Relation between modulation level and transmission distance to meet specific BER requirements [2]

Table 2.1: Modulation formats against path length [2]

Modulation Format	Bits per Symbol (M)	Data Rate (Gbps)	Maximum Reach (Km)
DP-BPSK	1	25	8000
DP-QPSK	2	50	4000
DP-8QAM	3	75	2000
DP-16QAM	4	100	1000
DP-32QAM	5	125	500
DP-64QAM	6	150	250

length increases which means that at a given distance the higher the modulation format, the higher will be its requirement in terms of Signal to Noise Ratio in order to make the receivers able to distinguish between different symbols and fulfill specific requirements for a minimum Bit Error Rate [10]. Moreover we also need to consider the fact that ROADMs act a little bit as filters for the signals passing through them reducing the optical bandwidth for the channels [5], so when a signal crosses multiple ROADMs along its route the resulting bandwidth may be narrowed, affecting the quality of the signal itself. With EONs the spectrum that we allocate for a given connection can be slightly increased in order to take this effect into account and therefore it represents another advantage w.r.t. traditional WDM networks.

Figure 2.5 represents different modulation schemes, that we are able to adopt thanks to coherent detection technologies, among which we have BPSK, QPSK, 8-QAM and 16-QAM. Figure 2.6 shows how by adopting some modulation schemes we can push our data rates beyond 100 Gb/s and how those rates can be achieved in both Fixed and Flexible Grid standards highlighting the advantages of a flexible approach where in some cases the improvement in spectrum usage is 150%. While table 2.1 shows how the different modulation schemes limit the transmission reach of the optical signals.

Another approach that can be used in order to increase the data rate is Dual Polarization (DP) which uses both vertical and horizontal polarizations, exploiting orthogonality, that allows to double the data rate that we can extract from a spectrum slice. In fact, in table 2.1 the data rates are referring to a single 12.5 GHz spectrum slot, showing how in just a single slot is theoretically possible to transmit up to 150 Gb/s for a distance of at most 250 Km. By

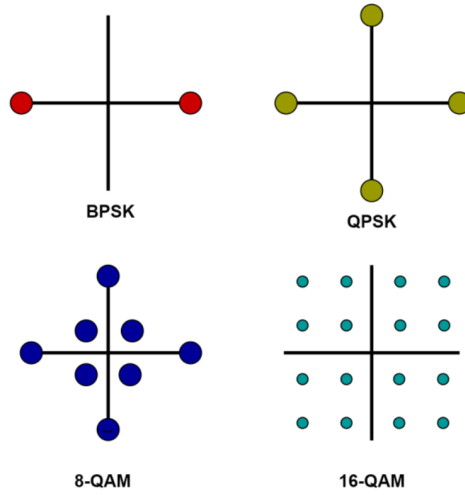


Figure 2.5: Modulation formats enabled by coherent detection [2]

Demand bit rate (Gb/s)	Modulation format	Channel bandwidth (GHz)	Fixed grid solution	Efficiency increase for EON
40	DP-QPSK	25+10	1 50 GHz channel	35 GHz vs. 50 = 43%
100	DP-QPSK	37.5+10	1 50 GHz channel	47.5 GHz vs. 50 = 5%
100	DP-16QAM	25+10	1 50 GHz channel	35 GHz vs. 50 = 43%
400	DP-QPSK	75+10	4 100 Gb/s in 4 50 GHz channels	85 GHz vs. 200 = 135%
400	DP-16QAM	75+10	2 200Gb/s in 2 50 GHz channels	85 GHz vs. 100 = 17%
1000	DP-QPSK	190+10	10 100G in 10 50 GHz channels	200 GHz vs. 500 = 150%
1000	DP-16QAM	190+10	5 200Gb/s in 5 50 GHz channels	200 GHz vs. 250 = 25%

Figure 2.6: Modulation format gains using the Flexible Grid standard [5]

exploiting Dual Polarization the data rate of a single spectrum slot can be computed as:

$$B_r = 12.5 \text{ Gb/s} \cdot 2 \cdot M$$

where the 2 is given by DP and M is the number of modulation levels. Vice versa if we want to achieve a specific data rate B_r and we want to know how many spectrum slots we need to allocate we can do it by means of:

$$N_{slots} = \left\lceil \frac{B_r}{12.5 \cdot 2 \cdot M} \right\rceil$$

When in EONs we also want to take into account the possibility to manage the modulation format the RSA problem becomes a Routing, Modulation and Spectrum Assignment (RMSA) problem where algorithms can achieve an higher spectrum efficiency w.r.t. RWA and RSA scenarios but of course the computational complexity is higher. Typically RMSA algorithms start by computing the route that allows the allocation of the desired spectrum slice and then based on its length decide which modulation format is better to use. Then the spectrum resources are allocated and the lightpath is set up. The working principle of RMSA algorithms is further investigated in the following section.

2.3 The RMSA problem

As previously mentioned with RMSA we address the problem of finding a suitable combination of route, modulation format and spectrum slice to fulfill connections requests to set up lightpaths inside Elastic Optical Networks.

2.3.1 Routing

The first thing that this type of algorithms do is to find a path, or multiple paths, between the source and destination endpoints of the connection. In the literature several routing strategies have been investigated which can be distinguished into the following three categories:

- **Fixed Routing:** in this case the routes are precomputed between every source/destination pair using a shortest path algorithm so that when a request arrives can be immediately served by picking the corresponding

route. This is an approach that is possible only in static environments and it is the approach that we used in our EON simulator where, in order to improve computational efficiency, we used Dijkstra's algorithm in order to compute the shortest paths between each source/destination node pairs and just picking the corresponding entry from a list when a connection request arrives.

- **Fixed Alternate Routing:** in this case a k -Shortest Path algorithm is used to pre-compute the paths between endpoints so that if later the spectrum assignment fails for the first available path between those specific endpoints we can try with the second available one and so on up to k trials. This may be useful to reduce blocking in several scenarios.
- **Adaptive Routing:** in this case we abandon the proactive routing approach in order to perform reactive path computation by looking at the network current state upon the arrival of connection requests in order to take better decisions in path selection to increase the overall network efficiency and reduce blocking.

2.3.2 Modulation format

Once the route has been established we need to find the optimal modulation format to use in order to optimize the usage of the available spectrum resources. To do so we need to retrieve information about the length of the links in the route and compute the total length of the path and based on data such as the ones reported in table 2.1 and chose for the appropriate modulation format that allows us to achieve the desired BER while minimizing the spectrum usage. In case we have multiple possible paths we may compute the most suitable modulation format for all of them for a faster lightpath setup in case of allocation failure for the first entries.

2.3.3 Wavelength and Spectrum assignment

Once the route and the modulation format have been chosen we need to assign the spectrum resources to the connection. In order to do that we need to distinguish between the two types of optical networks that we have seen since the unit for spectrum resources is different and therefore also the constraints that we need to satisfy.

In the case of a WDM network the assignment of a spectral resource is related to a specific wavelength λ that corresponds to the frequency ν of the carrier over which our transmitted signal is modulated. The spectrum that is assigned to each connection is fixed by the Fixed Grid standard to 50 or 100 GHz depending on the implementation and it must satisfy the two following constraints:

- **Wavelength Continuity:** the wavelength that is assigned to a connection must be the same throughout the whole route that has been selected.
- **Wavelength Non-Overlapping:** the selected wavelength can't be the same for any two connections that pass through the same link.

In order to fulfill those constraints and assign the proper resources we can follow different strategies. The most widely used are the following:

1. **First Fit:** the channel with the lowest available wavelength is selected.
2. **Last Fit:** the channel with the highest available wavelength is selected.
3. **Random Fit:** a random wavelength is selected among the available ones.
4. **First-Last Fit:** the channel with lowest available wavelength is selected for odd-numbered connection requests and the channel with highest available wavelength for even-numbered ones.

When we deal instead with Elastic Optical Networks the spectrum assignment is no more related to just a wavelength but also to another parameter that indicates the width of the spectrum portion that we want to allocate. The adoption of the Flexible Grid standard in fact enabled the possibility to assign to different connection request spectrum portions of different size based on the service needs, occupying what we can call a Frequency Slot (FS). The size of that FS can be adapted but it must be a multiple of 12.5 GHz which is the size of what we call Frequency Slot Unit (FSU) [6]. In this case the constraints that we need to satisfy are slightly different and can be expressed as follows:

- **Spectrum Contiguity:** the FSU slots that compose the FS we assign must be contiguous one each other, or in other words we can only group together in a single FS units that are adjacent one each other.

- **Spectrum Continuity:** the spectrum FS that has been assigned must be the same over the entire path.
- **Spectrum Non-Overlapping:** the spectrum FSs that we assign to different connection requests having at least one common link must not share any FSU with each other and this must be true for every connection and every link.

In this case the possible strategies that we can adopt in order to fulfill the requirements are the adapted versions of the previously mentioned First Fit, Last Fit, Random Fit and First-Last Fit such that they support the assignment of FSs of different sizes, but with the addition of another strategy that has been called **Exact Fit**. It consist in finding a suitable FS such that the adjacent FSU on both sides are occupied. In case there is no such FS then the assignment is typically done by means of a First Fit strategy.

Efficient algorithms to perform dynamic routing, modulation and spectrum assignment (RMSA) in EONs have been in proposed in [11] and [12], where the former adopts a genetic algorithm for multi-objective optimization while the latter exploits a Deep Learning algorithm in order to optimize the spectrum allocations, referring to the procedure as deepRMSA.

2.3.4 The fragmentation problem

Since our networks operate under dynamic and heterogeneous traffic scenarios connection requests arrive continuously and therefore lightpaths are set up and released very frequently. This can arise the problem of Spectrum Fragmentation. We can define fragmentation as the scenario in which there would be enough spectrum resources to serve an incoming connection request but we can't use them because of the allocation constraints. For example if we have a service that requires a spectrum FS of 100 GHz in an EON we would need (not taking into account guard bands) 8 FSUs which may be available considering the whole spectrum but if they are not adjacent one with each other we can't use them due to the Spectrum Contiguity constraint. Also in WDM networks if we want to assign an optical channel to a request we may not be able to do it because in one of the links in the route that channel is already used by another connection and therefore we wouldn't be able to satisfy the Wavelength Continuity constraint. In those cases of non-compliance with the assignment constraints the requests are blocked, so fragmentation management becomes

crucial if we want to improve the network performance in admitting connection requests which means that we need to adapt techniques and technologies that allow us to minimize fragmentation. One of the possible approaches to this problem is to exploit Wavelength Conversion as we implemented in our simulator.

2.4 The future of EONs

As the traffic workload keeps increasing more and more during the years, the transmission capacity of single-core fiber systems will soon reach its fundamental limits, therefore we must find smart ways to overcome this limit in order to pave the way for the next generation of Elastic Optical Networks. The main fields of research in this direction are: the exploitation of optical bands beyond the C-band and the implementation of multi-core optical fiber links. Moreover the optical fiber market is bending towards the optical disaggregation concept which is expected to provide an increased flexibility in network infrastructures and opens the door to the possibility of a faster network innovation.

2.4.1 Multi-Band Optical Networks

When a signal travels inside an optical fiber link its amplitude decreases with distance according to the following formula:

$$P(L) = P(0) \cdot 10^{-\frac{\alpha L}{10}}$$

where:

- $P(L)$ is the power of the optical signal at a distance L from the transmission point.
- $P(0)$ is the initial transmitted power at $L = 0$.
- α is the **attenuation coefficient** of the fiber. It represents how much the signal is attenuated over a certain distance measured in decibels per kilometer (dB/km).
- L is the distance from the beginning of the optical fiber link at which we measure the received power expressed in kilometers (km).

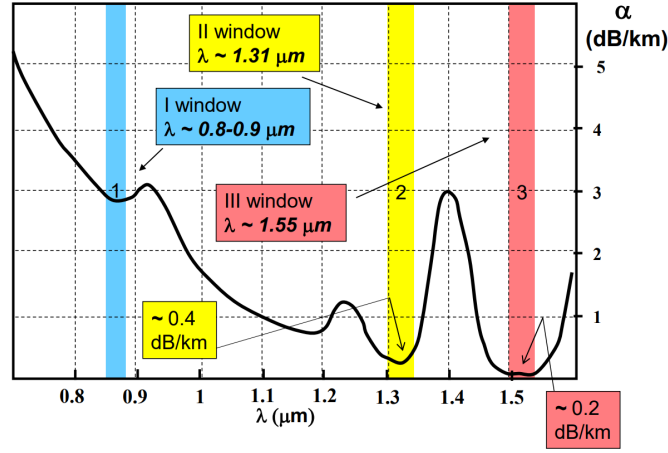


Figure 2.7: Classical optical transmission windows

In traditional optical fiber systems the attenuation coefficient α plotted w. r. t. the wavelength λ shows a fluctuating behavior with three minima keeping a constant value in a neighborhood around them. Those regions have been called the three optical windows and as shown in figure 2.7 they are called:

- **First Window:** centered at $850nm$
- **Second Window:** centered at $1310nm$
- **Third Window:** centered at $1550nm$

The optical window that showed the lowest attenuation among those three was the third one that is also called the **C-band** which was therefore used for communication purposes.

During the years has been discovered that those fluctuations were mainly caused by the materials that composed the fiber, in particular the presence of some ions. By the usage of innovative materials scientists were able to provide a more desirable behavior, as shown in figure 2.8, which opened the door for studies addressing the extension of optical windows (bands) that can be used for data transmission.

In particular the bands that have been identified as suitable for transmission have been named:

- **O-band** (Original): $1260 - 1360nm$

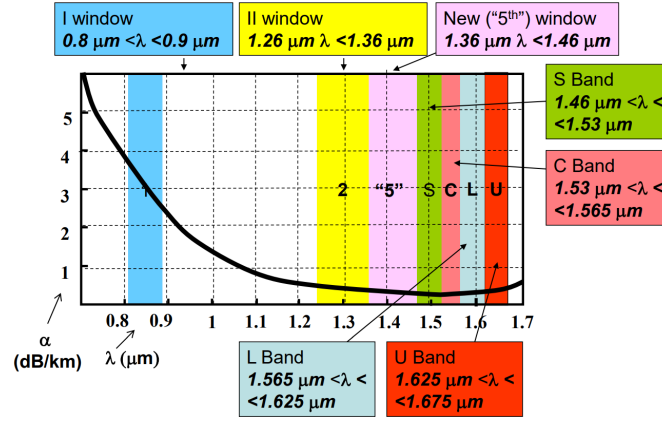


Figure 2.8: Recent definition of new transmission windows

Name	O-Band	E-Band	S-Band	C-Band	L-Band
Wavelength Range (nm)	1260-1360	1360-1460	1460-1530	1530-1565	1565-1625
Multi-Band System	← 365 nm →				

Figure 2.9: Multi-Band Optical Networks transmission window [2]

- **E-band** (Extended): 1360–1460nm
- **S-band** (Short-wavelength): 1460–1530nm
- **C-band** (Conventional): 1530–1565nm
- **L-band** (Long-wavelength): 1565–1625nm
- **U-band** (Ultra-long-wavelength): 1625–1675nm

We refer to the combined usage of all those frequency bands in optical networks as **Multi-Band Optical Networks** that have the goal to improve the system's capacity without the need to deploy additional fiber links. This kind of approach is summarized by figure 2.9.

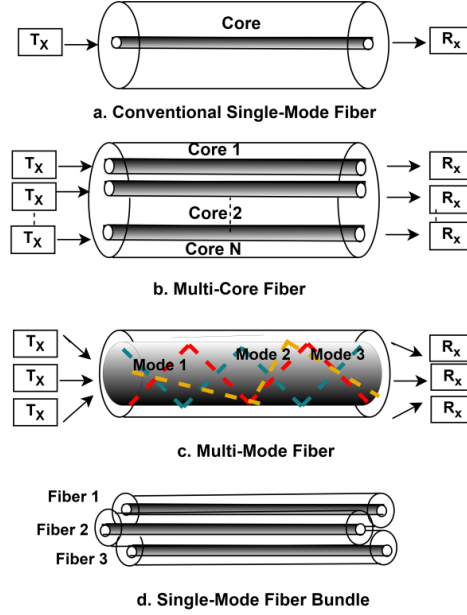


Figure 2.10: Different types of fibers used in SDM technology [2]

2.4.2 Space Division Multiplexed Optical Networks

Another approach to improving EONs capacity is to exploit the spatial dimension adding multiple fiber links, multiple cores inside a single link or multiple propagation modes inside the same core. we refer to this kind of approaches as Space Division Multiplexing (SDM) and its main approaches are represented in figure 2.10.

These main approaches include:

- **Single-Mode Fiber Bundle (SMFB):** in this case an optical link is composed of multiple distinct fibers packed together to form a bundle. The bundle may contain some fibers that are unused, to which we refer to as *black fibers*, in order to be able to exploit them whenever we will need it in the future without having to place them physically in the infrastructure. One of the main drawbacks of this approach is its poor scalability, in fact using multiple independent fibers also requires to have as many transceivers, amplifiers and other network components as the fibers in the bundle. This means that the cost per bit of this technology increases linearly with the number of fibers that we use.

- **Few-Mode Fiber (FMF)**: this type of fibers are designed to support the a few degenerated mode groups exploiting the different spatial distribution of the energy carried by the different modes allowing for the simultaneous transmission of multiple data streams with minimal crosstalk. These modes travel inside the fiber core with different propagation constants and they are multiplexed in order to be exploited as independent channels. The main drawbacks of this approach are the fact that even though the crosstalk is minimal between the different modes as they propagate they still couple with each other and therefore FMFs can be only used for short range optical links and moreover always due to the modes coupling the signals can only be routed towards the same destination where Multiple-Input, Multiple-Output (MIMO) techniques are exploited to decouple the multiple signals and be able to extract the different data streams.
- **Multi-Core fiber (MCF)**: this approach consists in collecting multiple Single-Mode Fiber cores inside one common cladding as it shown in figure 2.11 where a 7 core MCF with an hexagonal structure is reported. In recent years ultra-low crosstalk MCFs have been investigated and developed becoming the most popular approach to SDM technologies. MCF fibers provide a very high capacity (up to Petabits per second) and have also a high spatial efficiency as they have just one common cladding.
- **Few-Modes Multi-Core Fiber (FM-MCF)**: this approach is a combination of the two described above (FMF and MCF) where one common cladding confines multiple optical cores inside each multiple modes propagate simultaneously. As in MCFs this type of fibers are limited to short range links and the interference between different modes is limited by means of MIMO digital signal processing techniques at the receiver side.

Among the above described approaches to Space Division Multiplexing the Multi-Core Fiber one is the one on which most of the research is currently focused as it is more spatial and energy efficient w.r.t. the others. Designing and developing this kind of fiber though presents some challenges such as maximizing the number of core to insert in the same cladding without deteriorating the signal's quality which requires the minimization of Inter-core Crosstalk (IC-XT). With IC-XT we mean the evanescent field (the one that propagates outside the core) coupling between the different cores. It is typically computed

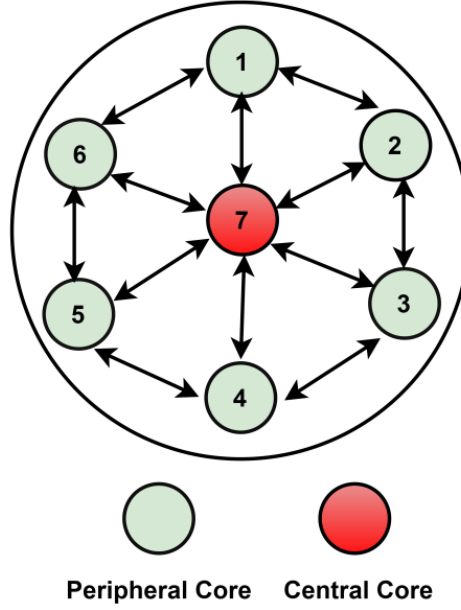


Figure 2.11: Multi-Core Fiber with 7 cores [2]

by means of the following crosstalk formula:

$$\text{XT} = \frac{n - n \cdot e^{-(n+1) \cdot 2hL}}{1 + n \cdot e^{-(n+1) \cdot 2hL}}$$

where

$$h = \frac{2k^2 r}{\beta \Lambda}$$

Here, k , β , r , and Λ are respectively the fiber coupling coefficient, propagation constant, the fiber radius, and the core pitch (so the distance between the center of two cores). While n is the number of adjacent cores to the core under consideration, and L is the total length of the fiber. In MCFs we have IC-XT happens only between spectrum slots that have the same indexes and between adjacent cores, in fact if the indexes are the same but cores not adjacent the coupling is negligible. If slot indexes are different there is no interference even among adjacent fiber cores. Another additional complexity of this kind of approach is the fact that when a new request arrives at a network node other than routing, modulation choice and spectrum assignment the node also

needs to select the best suitable fiber core in order to minimize the IC-XT. In this scenario the RMSA problem actually becomes RMCSA where the "C" indicates the core selection. Crosstalk aware algorithms have been proposed by [13] and [14].

2.4.3 Disaggregated optical networks

Another new approach that is increasingly gaining importance over the years is the one of optical network disaggregation, a concept that broke the traditional monolithic structure of optical network systems. In fact in the past when an optical network had to be deployed the vendors used to propose proprietary solutions of the entire infrastructure, therefore providing transponders, optical line systems, ROADMs and the respective management software. In fact each vendor had its specific software needed to interact with the network devices and manipulate their configuration leading to what has been called the *vendor lock-in* problem which led to the deployment of closed vendor-proprietary networks. The concept of disaggregation lies in the creation of open networks that can be composed by network devices of different vendors that interoperate seamlessly and that can be controlled via vendor-agnostic software. There are two planes of disaggregation:

- **Vertical disaggregation:** regards the way that optical devices can be configured, separating the hardware and software layers allowing those devices to be controlled by means of open vendor-neutral software. This approach opened the door for Software Defined Networking (SDN) implementations over optical network infrastructures.
- **Horizontal disaggregation:** refers to the possibility of having network devices that are interoperable among different vendors allowing customers to avoid the vendor lock-in problem bringing an increased flexibility in network design and deployment.

The combination of these two approaches is leading to an increased innovation speed and flexibility in optical networks since now each customer can build and update its network independently from the innovations introduced by one single vendor and moreover can introduce more advanced management and control architectures exploiting the benefits of the most recent products based on SDN architectures such as TeraFlowSDN [15], developed by ETSI, which is a

framework for a microservice-based, AI-integrated SDN controller that brings together optical transport integration and intent-based networking features, positioning it as a promising solution for managing multi-domain networks in a service-oriented way.

In the following chapter a description of the EON python simulator is provided together with its core components.

Chapter 3

EON simulator

In order to study the impact of all-optical wideband wavelength converters in Elastic Optical Networks, their benefits in terms of blocking probability and resource utilization and to analyze potential deployment costs I helped to develop an EON simulator that can be exploited to assess network performances on real-world topologies with realistic traffic patterns. The simulator is based on the concept of an existing simulator that was developed in Java called OfNetSim, presented in [16]. Our implementation was in the first place a translation of the Java logic into Python language and then to expand the work by adding several capabilities such as the possibility to use real-world topologies or custom ones for simulations, the exploitation of wavelength conversion, the selection of different algorithms that we designed and developed and the possibility to run simulation campaigns, so to be able to run multiple simulations with a single command. In this way the previous simulator became for us just a benchmark to assess that the results we were obtaining with basic simulation functionalities were consistent, so we used it to ensure the results matched using the same parameters.

This chapter presents the core Python classes that we used in order to model network elements for the simulations from the physical layer (e.g. nodes, links, topology) to the elements that we need to manage the simulation start and execution (e.g. spectrum, connection, route).

3.1 Core Classes

3.1.1 Node

The **Node** class models an abstract network node within the Elastic Optical Network simulator that emulates an optical switch from the physical layer viewpoint. Each node is responsible for managing its own set of input and output ports, which represent the connections to other nodes in the network. Each output port is associated with a **Link** object that carries spectral resources (spectrum slots) used for routing optical signals. Moreover each node is responsible for the management of the wavelength converters that are deployed on that node when needed for the allocation of resources for specific connections. We limited the number of supported connection for each converter to 5, provided that they all have the same frequency shift, after which we need to deploy an additional one, taking into account both the costs for deployment and usage of resources, such that the simulation can realistically compare capital and operational expenditures between the different allocation strategies. The main characteristics of a node object are:

- **input_ports**: a NumPy array of integers representing incoming link costs from other nodes (0 if no incoming link exists).
- **output_ports**: a NumPy array of **Link** objects, where each non-None element represents an outgoing connection to another node.
- **nID**: the unique integer identifier of the node.
- **converters**: a dictionary tracking active wavelength converters in the node. Each key is a tuple composed by the shift and the converter ID, while the values indicate how many simultaneous conversions are active for each converter. In fact with wideband wavelength converters we have the possibility to serve multiple conversions at the same time provided that they have the same shift in terms of spectrum slots.
- **max_conversions**: an integer value representing the maximum number of simultaneous conversions that each converter can support (by default we use 5).
- **placement_cost**: the cost associated with deploying a new converter in the node. Currently costs are implemented as adimensional values that

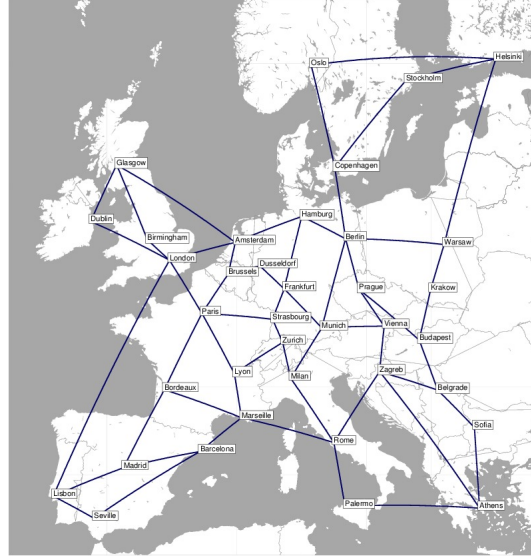


Figure 3.1: Optical Network Topology with 37 nodes and 57 links

are therefore used only as comparison metrics between the allocation algorithms that we implemented.

- **usage_cost**: the cost incurred for using an existing converter. This cost is much lower w.r.t. the deployment of a new converter.

Upon instantiation, the class parses the network topology matrix to determine the connectivity between nodes, initializing the corresponding ports accordingly. Topologies are taken from SNDlib [17] [18], a database of real-world network topologies that allow us to consider the routing problem taking into account physical (geographical) distances between the nodes which enables a better simulation quality while still keeping open the possibility to use also custom topologies by adding a properly formatted file. In our simulations we used both those possibilities, the custom one in order to match the simulation scenario of the reference simulator in which the topology was hard-coded, while the real topologies in order to observe the simulator's behavior in more realistic scenarios. An example of a real network taken from SNDlib is reported in figure 3.1.

Constructor Parameters

- `topology (np.ndarray)`: the adjacency matrix of the network, where a nonzero entry at (i, j) indicates a directed link from node i to node j .
- `node_id int`: the identifier of the node being constructed.
- `num_slots int`: the number of available spectrum slots on each outgoing link.

Key methods

Since we implemented wavelength conversion features we need the nodes to be able to manage the deployment and usage of wideband wavelength converters and their respective costs. We do that by means of the following functions:

- `use_converter(shift: int, debug_header: str = "")`: requests the usage of a wavelength converter for a given frequency shift. If an existing converter is available and not at full capacity, it is exploited, increasing the cost of just a usage cost. Otherwise, a new converter is installed, increasing the cost by a placement cost plus usage cost.
- `release_converter(shift: int, debug_header: str = "")`: resets all converters for a specific shift, effectively releasing all active conversions that have that specific frequency shift.
- `enable_debug(enable=True)`: enables debug output for tracing converter usage and management during simulation. We used this function in order to investigate the correct behavior of conversion management functionalities.

3.1.2 Link

The **Link** class models a unidirectional physical connection between two network nodes in the network. It encapsulates attributes related to the spectrum resources availability on that link and the physical length (or cost) associated with that connection, in fact typically if we use real-world topologies the meaning of the cost coincides with the one of geographical distance, but while using custom network topologies those costs can represent something more symbolic than the physical link's length. When a link is instantiated all its spectrum

slots are marked as free and while the simulation runs their status is dynamically updated according to both the establishment and release of optical paths that involve the specific links. The parameters that characterize a link object are:

- **link_cost**: the cost or length associated with this link (e.g., in km or arbitrary units).
- **input_node**: the identifier of the source node (start of the link).
- **output_node**: the identifier of the destination node (end of the link).
- **spectrum_status**: a NumPy array representing the availability of spectrum slots on this link (1 = free, 0 = occupied).

This class is mainly used by the **Node** and **ControlPlane** classes to handle spectrum assignment and release operations during the simulation considering the three allocation constraints that have been identified in section 2.3.3, which are Non-Overlapping Constraint (always considered), Contiguity Constraint (always considered) and Continuity Constraint (considered only in case wavelength conversion is not allowed).

Constructor Parameters

- **link_cost** (float): distance or cost metric of the link.
- **input_node** (int): the source node ID of the link.
- **output_node** (int): the destination node ID of the link.
- **num_slots** (int): total number of available spectrum slots on this link.

3.1.3 Spectrum

The **Spectrum** class models a spectrum slice within a link in the Elastic Optical Network. A spectrum slice refers to a contiguous set of frequency slots allocated to a connection for data transmission. This class serves as a simple container of information related to allocation, mainly used by the **Connection** and **ControlPlane** components during spectrum assignment and release operations. When we consider the spectrum for allocations in the simulator

we mainly refer to spectrum blocks which are described as a couple of tuples representing in order: the index of the starting link, the starting slot position index, the number of contiguous free slots (contiguity condition) and the number of links for which the same slots are available (continuity condition). A spectrum object can be seen as a block that cover only one link, therefore we only consider the following parameters to describe them:

- **size (int)**: the number of contiguous frequency slots allocated in this spectrum slice.
- **position (int)**: the starting index (slot number) in the spectrum where the allocation begins.

This object is particularly useful for ensuring compact representation and manipulation of assigned spectrum resources. It provides sufficient information to determine both the location and the extension of a spectrum slice on specific links.

Constructor Parameters

- **size (int)**: The number of adjacent slots that constitute the spectrum allocation.
- **position (int)**: The index of the first slot in the allocated slice.

Notes

The **Spectrum** class is designed for efficient handling and checking allocation constraints, particularly when determining whether a contiguous block of free slots exists for a new connection request or if it necessary to block that request or use wavelength conversion.

3.1.4 Route

The **Route** class models an abstract representation of a path within the network topology. It comprehends both the sequence of traversed nodes and the associated links total cost. This class is used to enable efficient management and evaluation of routing decisions during simulation. This is essential for the routing and spectrum assignment processes, as it provides a structured

format for storing and classifying candidate paths generated by the routing algorithms. Currently for routing, since our objective was to study the performances of spectrum allocation algorithms, we have only adopted a Fixed Routing approach as described in section 2.3.1, where by means of a Dijkstra's shortest path algorithm we compute all possible paths between endpoints at the beginning of the simulation which are not further modified during the simulation. With this approach we have a very efficient path assignment to the requests since it only consists of a lookup based on the connection's source and destination nodes. For future evaluations possible alternative approaches are the implementation of Fixed Alternate Routing techniques with the precomputation of k shortest paths instead of only one and using them as backup if the spectrum allocation for the first path fails. An even more advanced approach would include an Adaptive Routing technique which computes paths in a reactive way upon the arrival of connection requests which may be useful to perform routing considering the current links' conditions which can bring a significant improvement in terms of spectrum utilization and would reduce requests blocking. The attributes that we use to characterize a route object are:

- **route** (`List[int]`): an ordered list of node IDs representing the end-to-end path from the source to the destination node.
- **route_cost** (`float`): the total cost or length of the route, typically calculated as the sum of the link costs along the path.
- **route_id** (`int`): a unique integer identifier assigned to distinguish a specific route instance.

This class is typically used by the **ControlPlane** module when selecting feasible routes to manage connectivity requests. To have this kind of structure is particularly useful in scenarios involving multiple candidate paths as k -shortest paths algorithm or disjoint path routing since, as the number of simulated events gets large the time required for the simulation grows, having precomputed routes allowed us to reduce simulation times which is particularly useful while running simulation campaigns. Moreover this somehow limited approach does not affect the focus of our simulations which is the comparison of spectrum allocation algorithms.

Constructor Parameters

- **route** (`List[int]`): the ordered sequence of node identifiers that defines the path.
- **route_cost** (`float`): the computed total cost of the route.
- **route_id** (`int`): an identifier that uniquely distinguishes a specific route from others.

The `getRoute()` method provides external access to the route information, which is useful when spectrum assignment algorithms or event handling mechanisms need to reference to a particular path structure. In future extensions, this class could be enhanced in order to support additional parameters such as modulation formats in order to be able to select the most suitable one according to the route's length or the usage of wavelength conversion as described in section 2.3.2.

3.1.5 Topology

The **Topology** class plays a central role in modeling the physical layer of the simulator. It is responsible for importing and representing the network topology from a structured YAML file which can be specified by the user or otherwise selected as by default parameter. The file can contain both a custom user topology or a real-world scenario (e.g. from SNDlib [17] [18]) which can be imported directly using the converter we included in the project which converts a simple txt file into a properly structured YAML file.. This file defines the network in terms of its nodes, their geographical positions, and the links interconnecting them. When the class is initialized it reads this information and constructs an adjacency matrix, stored as a NumPy array, where each non-zero element represents the euclidean distance between two connected nodes. Moreover, the class extracts node identifiers and link structures into accessible Python objects. The `load_topology()` method performs the core parsing and matrix construction, while the `show_topology()` method is used to graphically visualize the network using `matplotlib` and `networkx` libraries, automatically saving and opening the resulting plot. This modular and portable design supports rapid prototyping and scenario customization using human-interpretable YAML configurations. The attributes that we use to characterize a topology object are:

- `yaml_file` (`str`): the path of the YAML file containing the structured topology description.
- `num_nodes` (`int`): the total number of nodes in the network.
- `topology` (`np.ndarray`): an $N \times N$ numpy array with elements representing the costs of the connections between nodes. The cost is zero if there is no connection between that node pair.
- `nodes` (`List[str]`): a list with network node IDs.
- `links` (`List[Link]`): a list of links.
- `node_positions` (`dict`): geographical coordinates of nodes.

Constructor Parameters

- `yaml_file` (`str`): the path of the YAML file containing the structured topology description.

3.1.6 Connection

The **Connection** class models an active optical network connection (a light-path), containing information about the assigned spectrum, the routing path, and associated characteristics such as the requested service type and the connection start and release time. For what concerns the duration of connections exponentially distributed service holding times have been implemented. This class plays a key role in the event-driven simulation by representing individual service instances that occupy spectral resources temporarily and release them after a certain holding time. In order to characterize a connection the following attributes are used:

- `time_stamp` (`float`): the scheduled release time of the connection, used to free up spectral resources.
- `route` (`List[int]`): the path taken by the connection, represented as an ordered list of node IDs.
- `spectrum` (`List[Spectrum]`): the spectrum resources allocated to this connection.

- **service_type** (**int**): an integer indicating the service type (e.g., indicating how much bandwidth is required for that connection). In fact we have a multi-service EON simulator, so we may have different type of services with different requirements in terms of bandwidth. In the future other requirements may be added for different service types to influence for example the length of the path, the modulation format or the usage of wavelength conversion.
- **id_conn** (**int**): a unique identifier used to track the specific connection instance.

The connection object is created during service arrival events and is scheduled for release based on its holding time. When a connection request arrives the control plane finds the best path, by means of a lookup among precomputed routes, and allocates the required spectrum resources. Upon release those resources are freed and made available for future connection requests.

Constructor Parameters

- **time_stamp** (**float**): the time at which the connection is expected to be released.
- **route** (**List[int]**): a list of node identifiers representing the connection route.
- **spectrum** (**List[Spectrum]**): a list of **Spectrum** objects indicating the allocated spectrum slices along the route.
- **service_type** (**int**): an integer representing the service type.
- **id_conn** (**int**): a unique identifier for the specific connection.

Notes

The `__lt__` method allows **Connection** objects to be directly used in priority queues, which is essential for event scheduling, which is managed in the simulation engine. This behavior ensures that connections are released in the correct temporal order, enabling accurate modeling of spectrum resources availability over time during dynamic simulations.

3.2 Control Plane

3.2.1 Control Plane logic

The control plane plays a pivotal role inside our network modeling logic. In fact it is responsible of the main EONs management operations, among which we have: path computation (and selection), spectrum assignment and release, service type selection, characterization of connection attributes and output metrics computations (e.g. bandwidth blocking, service blocking and costs).

We implemented all this inside the **ControlPlane** class which requires for its initialization a network topology, that we take starting from one of the attributes of the **Topology** class described in section 3.1.5, more in particular we consider its topology attribute, a NumPy array that is a matrix in which each element (i, j) represents the cost of the connection between nodes i and j , currently implemented as related to geographical distance, but it can be therefore generalized in order to consider also eventual different types of costs. In case the element (i, j) is zero it means that those two nodes are not directly connected. In fact the first function the **Topology** class contains is `load_topology(self)` which takes the YAML file that the user specified, or the default file name if didn't, and converts it in the previously mentioned topology NumPy array needed to initialize the control plane. Another required attribute is the edge nodes list which contains the identifiers of the nodes that are candidate to be endpoints of connections. By default, if the user doesn't specify it, we consider all nodes as edge nodes. To complete the control plane initialization we also need traffic probabilities which is an array containing as many elements as the services we want to deploy in our network and each entry represents the probability that a new network request is of a certain service type. Our simulator automatically computes in a random way the traffic probabilities if the flag that allows the user to specify custom ones is set to false.

3.2.2 Path computation

Once the control plane is up and running the first thing that we do is to compute all possible valid paths inside the topology in order to reduce simulation times. In fact at the beginning we used to compute a route each time a connection requested a path between two endpoints but when we wanted to

optimize the simulator's performances we decided to switch from a reactive to a proactive approach according to a Fixed Routing strategy described in section 2.3.1. In fact one of the advantages of adopting a Software Defined Networking (SDN) scenario is the availability of global network information so we completely know the topology and therefore we can pre-compute all possible routes between each pair of endpoints just once and then pick the best fit when a connection arrives. This was a big improvement since each simulation was generating up to 200.000 connection requests. Moreover this allowed us to compute in advance some statistics about the routes like the maximum and average number of hops and this approach doesn't collide with our goal which is the comparing the different spectrum allocation algorithms that we designed to exploit wavelength conversion capabilities of the network. Of course we need to take into account that all this simulator works on a static network environment, so the topology doesn't change over time, therefore a Fixed Routing approach doesn't affect the simulations, but in the future may be useful also to implement Fixed-Alternate Routing and Adaptive Routing approaches for a more complete representation of the RMSA problem described in section 2.3.

In order to compute the best paths between each pair of endpoints (edge nodes) we used an optimized Dijkstra's algorithm implementation that exploits scipy libraries and sparse CSR matrices for faster computations. The function returns a tuple containing a list of node IDs that compose the selected route and a float that represents the cost associated to that route, which is simply the sum of the geographical distances between node pairs. The only real limit of this Fixed Routing strategy is the fact that when we compute paths at the beginning and associate a cost with the path we can't take into account the current links' status and therefore the need to perform wavelength conversion which to some extent increases the cost of that route, not in terms of geographical distance but most probably on a latency metric viewpoint. This means that there may be some specific scenarios, even though probably rare, in which the shortest path we chose in which we perform wavelength conversion, has a cost that is higher than a longest path in which instead we didn't need to perform any conversion. Taking this into account would require Adaptive Routing strategies. For the moment we consider the costs related to the actual length of the path as separate from the costs of the lightpath establishment where we consider the path cost as the number of hops needed to move from source to destination along that route.

3.2.3 Costs management

In order to compare the operational costs of our implemented allocation algorithms we decided to assign a-dimensional cost values to the usage and deployment of resources. For every connection we check the spectrum that has been allocated and if the position of the slice that was chosen for two consecutive links is different it means that we need to have wavelength conversion in the node between them. At first we compute the shift and look at the dictionary where converters are stored (see section 3.1.1) in order to see if we already have an existing converter serving conversions with that specific shift on that specific node. If there is one we look at the currently active conversions for that particular converter since we know that wideband converters can support multiple conversions at the same time provided that they have all the same frequency shift in terms of spectrum slots. We decided to allow the support for a maximum of five concurrent conversions, so if the converter has already all five active conversions then we need to deploy a new one to serve the current request. At the moment the costs that we assigned to the different connectivity resources are:

- Converter placement: 100
- Converter usage: 5
- Spectrum slot usage: 1
- Node usage: 2

There are generic costs for resource utilization and considered to be like this as we considered that placing a new converter is a very big cost as it falls in the category of Capital Expenditure (CAPEX) while all the others, using an already deployed converter, using nodes and spectrum slots, fall under the Operational Expenditure (OPEX) costs and therefore are considered significantly lower. Computing the total cost of allocation for each simulation point we are able to compare the performances of the different algorithms, providing a possible impact picture of what their usage in real world deployments may lead to.

3.2.4 Spectrum management

Then the **ControlPlane** class contains the functions needed for spectrum management operations starting from the ones to assign it (mark it as occupied) or to release it (mark it as available). In fact, as we have seen in section 3.1.2 the availability of the spectrum on a specific link is related to its **spectrum_status** attribute that we can manage by interacting with a **node** object's output ports. Such attribute is a boolean vector of zeros and ones where a zero represents that the slot is not available, so occupied by another network connection, and a one represents available slots. When we perform an allocation on a specific route we chose a set of contiguous spectrum slots, according to service needs, for each link. Those consecutive slots must be the same for each link in the route if we need to maintain wavelength continuity, or they can be shifted between consecutive links if we can perform wavelength conversion, but still the slots must be contiguous since this is a key characteristic of elastic optical networks. In our case we consider on each link a total available bandwidth of 4THz and we conform to the WDM flexible grid standard assigning to each spectrum slot (also called slice) 12.5GHz which means that on each link there will be 320 slots available slots. Moreover we can see that in our simulations we consider mainly two service types, even though the simulator can be used for as many as we want. The first service requires 100GHz, which translates into 8 contiguous spectrum slices, while the second one 400GHz, so 32 contiguous slots. Each time a request can be served a spectrum resource for each link in the path is chosen and allocated using the **assign_spectrum** function and released once the connection request expires using the **return_spectrum** one.

Now we are ready to introduce the concept of **spectrum block** and the functions that we use to prepare blocks for the allocation algorithms. In fact traditional elastic optical networks work with spectrum assignment algorithms that require wavelength continuity and therefore look for a certain number of adjacent spectrum slots that are available along the whole path. Recent studies though showed the possibility to exploit wideband wavelength converters to improve blocking performance and their feasibility in EONs, which opens the door to the design and development of new versions of allocations algorithms that exploit this possibility. Even though it seems quite straightforward in order to address the problem we need to take into consideration the spectrum fragmentation problem which limits the efficiency in the resource usage,

especially in contexts, like the one we simulated, where we have service heterogeneity, which leads to different sizes of the spectrum slices that we allocate. In order to reduce the fragmentation problem when we perform allocation when possible we consider spectrum blocks and also the possibility to break or not break available blocks. With the expression spectrum block we mean a set of contiguous spectrum slots that are available on one or more consecutive links. We address them by considering 4 parameters for each block: their size (in number of spectrum slots), the slot from which they start, the link number they start from and the number of links they cover. For example the block [3, 0, 2, 3] is a block of size 3 slots that starts from slot 0 that covers from link 2 to link 5. We have designed algorithms that exploit this description in order to take smarter decisions allocating blocks of the maximum possible size when available according still to the policy they describe. Since they are the main focus of the work the allocation algorithms will be described in a dedicated chapter.

Here we can briefly describe the function that are used by those algorithms in order to prepare the spectrum resources in order to chose the best ones according to their internal logic. These functions are:

- `get_per_link_spectrum_matrix`: Starting from an array of Node objects it returns a matrix where each row corresponds to a link in the route and each column indicates whether the corresponding spectrum slot is available or not.
- `get_available_boolean_spectrum`: Computes the available spectrum slots along a whole route. For each spectrum slot it looks if it is available along the whole route adding to a vector a 1 if that slot is available end-to-end and 0 if not.
- `create_vector_spectrum`: Converts a binary list representing the spectrum availability situation into a list of Spectrum objects.
- `extract_available_blocks_from_matrix_jit`: Extracts all valid spectrum blocks of a given required_bandwidth from a spectrum availability matrix. Each row in the matrix corresponds to a link along the path, and each column represents a spectrum slot. A slot is 1 if available, 0 if occupied. For every possible starting slot, this function checks if a block of required_bandwidth contiguous slots is available (all 1s) for each

link. It then counts how many consecutive links the block covers. For optimization purposes it uses `njit` `numba` libraries, compiling the function code instead of interpreting it as in standard python making the execution much faster for multiple usages.

3.3 Spectrum allocation algorithms

3.3.1 Current implementations in EONs

As mentioned in section 2.3.3 at the current state to establish lightpaths inside EONs we must satisfy the Spectrum Continuity condition which means that in every link of that path our allocation algorithm must select the same spectrum resources and still satisfy the Non-Overlapping and the Contiguity constraints. Using the spectrum blocks terminology, introduced in section 3.1.3, to satisfy this continuity constraint means that the algorithms need to pick for the allocation a spectrum block of the desired size which covers a number of links equal to the total length of the path.

As a consequence each allocation will always be just one spectrum block. To face this scenario several possible strategies have been investigated for the choice of that block as mentioned in section 2.3. The most widely recognized are: **First Fit**, **Last Fit** and **Random Fit** allocation strategies. In our case in order to distinguish them from the other approaches we have implemented we refer to those simple scenarios with two algorithms called `first_fit_base` and `random_fit_base`. The functioning of those algorithms is very straightforward; the `first_fit_base` picks the first available spectrum block that covers the whole path while the `random_fit_base` among all available blocks picks a random one. In order to prepare the spectrum for those functions, since we don't have to consider wavelength conversion, we used a function called `get_available_boolean_spectrum` that performs a bitwise AND operation between each slot with the same slot in the other links of the path, generating a boolean vector that indicates which spectrum slots are available along the whole route, a situation that allows us to fulfill the Spectrum Continuity constraint. Once chosen according to the selected policy, and therefore the corresponding algorithm, we transform the selected 1-link block into as many spectrum object as links in the path and for each of them we mark with the `assign_spectrum` function, those spectrum portions as occupied on each link in that route.

Let's consider the following example of a simple path that involves 3 links, each one with 10 spectrum slots. The following matrix S_{links} represents the status of the spectrum slots along the links; each row corresponds to a link and each column corresponds to a spectrum slot, where 0 means the slot is already occupied by another connection, and therefore it can't be selected for an allocation, while 1 means that the slot is available.

$$S_{\text{links}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

In this case the `get_available_boolean_spectrum` function will generate the following boolean vector A_{slots} where 1 mean that the corresponding slot is available along all the three links while 0 means that at least in one of the links that slot is occupied and can't be used to satisfy the Spectrum Continuity constraint.

$$A_{\text{slots}} = [1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0]$$

Assuming that a new connection request arrives at the control plane with a bandwidth requirement of $21GHz$ then we need to convert that requirement in terms of spectrum slots by means of the following formula:

$$N_{\text{slots}} = \left\lceil \frac{B_{\text{required}}}{B_{\text{slot}}} \right\rceil$$

Where B_{slot} is the bandwidth of a spectrum slot that corresponds to $12.5GHz$, so in our case we have:

$$N_{\text{slots}} = \left\lceil \frac{21GHz}{12.5GHz} \right\rceil = 2$$

This means that we need to allocate two contiguous slots to accomplish the request, so by looking at the boolean vector we obtained we can see that the possible choices that we make, using slot indexes $[0; 9]$, are: $[0; 1]$, $[1; 2]$ and $[6; 7]$. The `first_fit_base` algorithm will pick the choice with the lowest possible starting index, so here $[0; 1]$, while the `random_fit_base` will randomly pick among the three possibilities with equal probability. Once selected, for example $[0; 1]$, we convert it into spectrum objects, one for each link in the path even if in case of no wavelength conversion those objects are identical, so in this example a

tuple with starting slot index and number of slots is generated, so we will have [0 2] for link 1, [0 2] for link 2 and [0 2] for link 3. This allows independent allocation for the different links and the same is valid when the connection is dismissed. Then with the function `assign_spectrum` we mark as occupied the addressed slots and therefore the S_{links} matrix becomes:

$$S_{\text{links}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Performing the assignment with the `random_fit_base` algorithms instead we would have picked with a probability of 33% one of the three blocks that have been previously individuated.

3.3.2 Wavelength conversion algorithms

As already mentioned in chapter 2, one of the key-enablers of our investigation is the implementation of **wideband wavelength conversion** to allow the spectrum allocation algorithms to assign blocks that don't start from the same slot position on each link. Wideband converters are a particular type of wavelength converters also referred to as **all-optical converters**. In fact their peculiarity is the fact that they work completely in the optical regime without the need, like in the case of OEO converters, to move to the electrical domain. This characteristic though also introduces some physical limitations, as a matter of fact a wideband converter can't convert each wavelength to any other wavelength, it can only perform frequency shifts with a certain range and moreover it can only deal with optical signals that have a bandwidth smaller of a certain amount. Both those limitations depend on the specifications of the device, so we may have some wideband converters that work with certain signals while others don't. Moreover an important characteristic of this kind of converters is the fact that they can handle multiple conversions at the same time provided that their frequency shift is the same. Our implementations comprehend an adaptation of standard allocation algorithms with the addition of wavelength conversion capabilities, but also some more sophisticated logic including the so previously mentioned blocks logic, break/no break feature and introduction of memory for recursive allocation. Our implementations are reported in table 3.1:

Algorithm	Conversion	No Break Logic
first_fit_base	Yes	No
random_fit_base	Yes	No
first_fit_best_effort	Yes	Yes
random_fit_best_effort	Yes	Yes
first_fit_best_effort	Yes	No
random_fit_best_effort	Yes	No
first_fit_stubborn	Yes	Yes
random_fit_stubborn	Yes	Yes
first_fit_stubborn	Yes	No
random_fit_stubborn	Yes	No
longest_fit_best_effort	Yes	Yes
longest_fit_best_effort	Yes	No
longest_fit_stubborn	Yes	Yes
longest_fit_stubborn	Yes	No

Table 3.1: Designed and implemented wavelength conversion algorithms for allocation of spectrum resources

In order to properly describe them we can distinguish them by means of their characteristics: policy, logic and use of the spectrum.

3.3.3 Policy

We have three policies of algorithms:

- **First fit:** it looks to the spectrum and tries to allocate the first available slice that fits the service bandwidth required.
- **Random fit:** among all the available and allowed slices it allocate one of them choosing randomly.
- **Longest fit:** it looks to all spectrum blocks and picks the one that covers the highest number of links and starting from that it constructs the allocation for the remaining uncovered links.

3.3.4 Logic

We distinguished between three types of algorithm logic:

- **Base:** it's the simplest implementation since it doesn't look to spectrum block but works with a link-wide view, taking into account the available spectrum on each link separately.
- **Best effort:** it is provided with a path-wide view of the spectrum, therefore it has a global view and considers all available blocks supporting a less fragmented spectrum usage.
- **Stubborn:** it is the heaviest logic from a computational viewpoint. As the best effort model it works with spectrum blocks but while the previous one picks a block, tries the allocation and blocks if it fails, this version can backtrack the blocks that it already choose and if the allocation fails it goes back and tries again with another one. It is particularly useful when we have to consider physical shift limitations of wideband converters, since they provide a better chance of finding an available allocation, but of course they come with the cost of high computation demand.

3.3.5 Spectrum usage

We distinguished between two types of approaches for the management of the spectrum in order to try the effects of the reduction of the spectrum fragmentation problem. The first approach is the one that doesn't care and so it can pick blocks regardless of the blocks conditions. We called this approach **break**. On the other side instead we have versions of the algorithms that work with a **no break** condition. Before picking a block they check if they are breaking a longest block or not, so if among the slots of the block they want to allocate at least one is occupied on the previous link in the path and if the block ends in a link where immediately after has at least one of the slots occupied.

3.4 Allocation examples

We have already seen in section x an example of allocation by means of the First Fit (FF) algorithm which in that case did not allow wavelength conversion. The same thing could have been done for the Random Fit (RF), but even in that case the allocation would have been done without exploiting conversion, the blocks logic and the break/no break spectrum usage logic. Now in this section a few allocation examples that exploit wavelength conversion

are provided in order to clearly see the differences between the several policies and logic of the algorithms. In order to do that is important to remind the meaning that was associated to a spectrum block. In fact by means of block I refer to a set of four numbers: the index of the link from which the block starts, the index of the spectrum slot from which the block starts, the number of slots that the block comprehends and the number of links that the block covers. Let's make an example by means of the following matrix:

$$S_{\text{links}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Assuming that we are looking for spectrum blocks of size 3 contiguous slots then what we find using the a function called **extract available blocks from matrix** is reported table 3.2:

Starting Link	Starting Slot	Bandwidth	Links Covered
0	0	3	3
0	6	3	2
0	7	3	2
3	1	3	1
3	2	3	1
3	4	3	1
3	5	3	1
3	6	3	1

Table 3.2: Blocks with 3 slots of bandwidth.

Let's see now how the different algorithms behave. We consider a route in which 5 links are involved with all their 15 spectrum slots available. We use a function called **get per link spectrum matrix** which retrieves the spectrum availability information for the links involved in a specific route. In this case since all the spectrum slots in each of the 5 links are available the matrix we obtain is a matrix filled with all 1s.

	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15
N0-N1															
N1-N2															
N2-N3															
N3-N4															
N4-N5															

Figure 3.2: S_{links} matrix with some unavailable slots

$$S_{\text{links}} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

We mark some slots as occupied in order to see in action the logic of the different algorithms, so for example we can start our allocations from the following matrix:

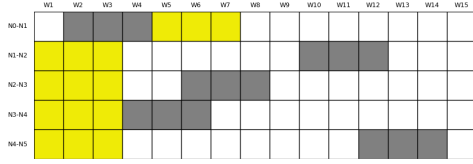
$$S_{\text{links}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In the simulator we have implemented a function called `plot spectrum availability` that allow us to graphically visualize the availability matrices and also if a list of spectrum objects is given also to plot the allocation that an algorithm performed. In this case the S_{links} matrix is graphically represented in figure 3.2.

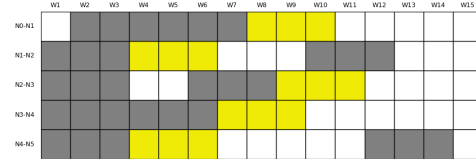
As we can see the gray slots represent the 0s in the matrix, so the unavailable spectrum slots, while the white slots the 1s, so the available ones. Now we proceed by allocating the spectrum for a connection request, that needs 3 contiguous spectrum slots, with different algorithms

3.4.1 First Fit base with conversion

The First Fit base algorithm is the simplest version of FF and therefore its working principle is very straightforward. It looks at each link individually and assigns the first available slice in that link composed of the specified number of slots, which in our example are 3. If in one of the links the desired number of contiguous slots can't be found it blocks the request, which in our simulator logic is implemented as returning a list of spectrum objects, one per link in the route, with position and size equal to -1 that tells the function responsible for marking as occupied the assigned slots that nothing has been assigned and therefore nothing has to be marked as occupied.



(a) First allocation



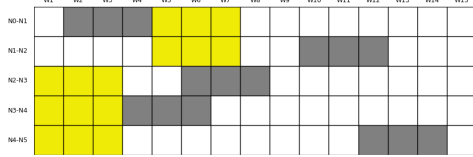
(b) Second allocation

By using the algorithm in our example to perform two assignments the results we obtain are represented in figures 3.3a and 3.3b. As we could expect the algorithm for the first allocation picks the first available slots on each of the links performing a wavelength conversion between the first and the second link. With the second allocation we can notice how the spectrum of the first one has already been assigned as occupied and therefore we satisfy the Non-Overlapping constraint as the first and second allocation belong to two different connection requests that share the same route. Moreover we can notice how the algorithm works in a link-by-link logic without considering any kind of spectrum blocks concept.

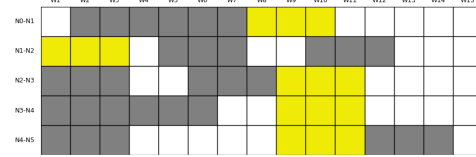
3.4.2 First Fit best effort with can_break logic

The best effort version of the FF algorithms brings with it the spectrum blocks concept which means that it starts scanning the first link in order to find a suitable number of contiguous spectrum slots and if it finds them it looks if the same slots are available also in the next links. As far as they are available the algorithms picks the same slots on consecutive links, while if they are occupied it exploits wavelength conversion. As we can see from figure 3.4a in the first

allocation we made on the S_{links} matrix, differently w.r.t. the FF base version, it picked on the second link the same slots it picked in the first one since they were available. After that those slots are occupied in the third link and therefore wavelength conversion is performed.



(a) First allocation

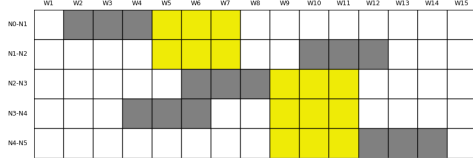


(b) Second allocation

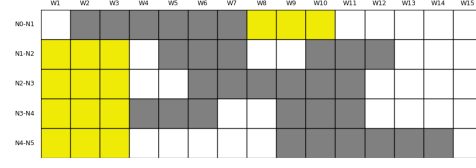
Also in the second allocation, reported in figure 3.4b we can see that in the third, forth and fifth links the same spectrum slots are chosen even though on link 4 and 5 slots with a lower index were available. This blocks logic has been implemented in order to reduce the costs for converters' placement, in fact as reported in section 3.2.3, to place a new converter is a very high cost and therefore trying to minimize it can lead to significant advantages from an implementation viewpoint even though it may lead to a slightly increased blocking probability for connection requests.

3.4.3 First Fit best effort with no_break logic

This version of the best effort FF algorithm brings with it the concept of non-breaking spectrum blocks which means that only blocks that have at least one spectrum slot occupied above and below can be chosen. This is done in order to try reducing the spectrum fragmentation problem mentioned in section 2.3.4. By looking at figure 3.5a we can see how only blocks that have above and below at least one slot occupied (of course we assume that above the first link and below the last one all slots are occupied), in fact the first 3 slots on the third, forth and fifth links can't be picked since on the second link those same slots are available. The same happens in figure 3.5b where again the first available blocks that satisfy the no-break constraint are allocated.



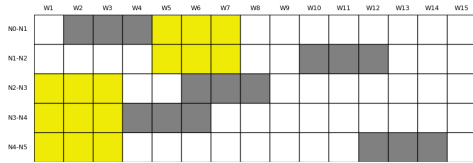
(a) First allocation



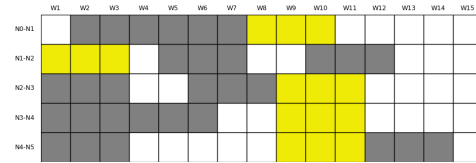
(b) Second allocation

3.4.4 First Fit stubborn with can_break

We've then designed a stubborn version of the previous FF algorithm which allows for a recursive look for a suitable allocation. In fact the best effort algorithm after choosing the first block goes on trying to continue the allocation and if not possible it blocks. This stubborn version instead if after choosing the first block the allocation fails it can go back to its steps and picking another first block and try to finish the allocation from there. The best effort and stubborn version are completely equivalent from the blocking viewpoint in scenarios in which we can convert of whichever shift in number of spectrum slots, but with the stubborn version being computationally inefficient when it comes to blocking requests as it recursively tries all possible blocks for allocation. The idea behind this version is the fact that wideband converters as mentioned in section 2 can only allow frequency shifts up to a certain number of spectrum slots that depends on the specific converter. We performed tests in which we limited the number of slots of which we can shift with a conversion to 2 and the stubborn version performed much better w.r.t. the best effort version because if there was a suitable allocation it was finding it while the best effort version was limited to try the allocation starting from only the first available block. In our example with the S_{links} matrix this stubborn version, without shift range limitations, performed the allocation reported in figures 3.6a and 3.6b that as expected are completely equivalent to the ones of the best effort version.



(a) First allocation



(b) Second allocation

We've also designed the no_break version of the stubborn algorithm which

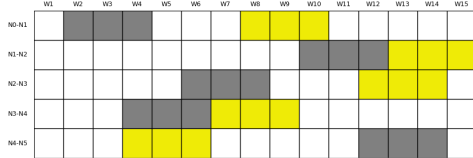
again in absence of shift limitations performed exactly the same allocation of the no_break version of the First Fit best effort.

3.4.5 Random Fit algorithms

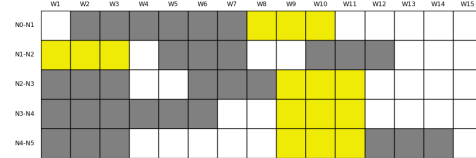
As reported in table 3.1 we have the same approaches seen with FF algorithms also applied to a Random Fit policy. Starting from the **RF base** we can say that such as the FF base in case of allowed wavelength conversion it scans link by link the spectrum and individuates the suitable slices that fulfill the bandwidth requirements and then randomly picks one among them. This algorithm is typically used for pre-allocation, so to create some noise in a situation in which all spectrum slots are available. For example this is the algorithm we used to generate the S_{links} matrix.

$$S_{\text{links}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

In fact, in this case we have on each link a slice of 3 slots that have been assigned and the position of the slices is independent on each link. On that matrix we also performed other 2 allocations which are reported in figures 3.7a and 3.7b.



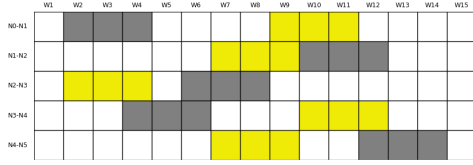
(a) First allocation



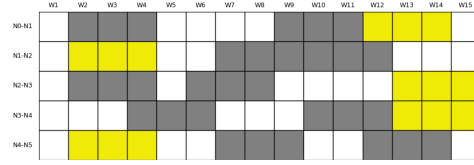
(b) Second allocation

As we can clearly see w.r.t. the previous algorithms this is the most chaotic since on each link in the route an independent choice of spectrum slice is performed without any kind of spectrum blocks concept. That concept is in fact introduced by the **RF best effort** version that as its FF counterpart finds all available spectrum blocks of the desired bandwidth on the first link and randomly picks one, then performs the same starting from the first uncovered link. Of course also blocks that cover only one link are considered and for

example starting from our S_{links} matrix situation a quite peculiar allocation made with this algorithm is reported in figure 3.8a, where the blocks that have been picked could cover each time only one link since the same slots on the following link were always occupied. A similar thing happened in figure 3.8b, the second allocation, where instead when it was possible from link 3 to link 4 the same slots were used since this version works in a blocks logic. Obviously since it is a random policy algorithm if we try the allocation several times most likely it will give different results.

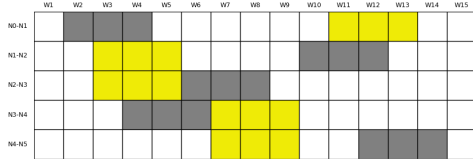


(a) First allocation

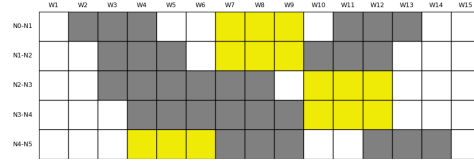


(b) Second allocation

Of the RF best effort we have also the version that doesn't allow to break spectrum blocks and therefore must pick blocks that have at least one slot occupied in both the previous link and the next one (still assuming that before the first link in the route everything is occupied and the same after the last link). In this case the two test allocations are reported in figures 3.9a and 3.9b respectively. We can notice that for example the first allocation that was done is only one of the possible that were available, in fact an alternative could have been for example to chose the blocks: $[0; 5; 3; 2]$, $[2; 9; 3; 2]$, $[5; 3; 3; 1]$ and probably if the algorithm is run again it may the chosen allocation.



(a) First allocation

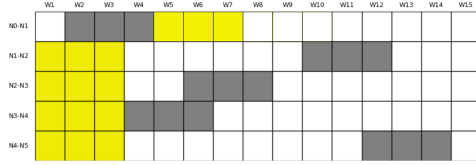


(b) Second allocation

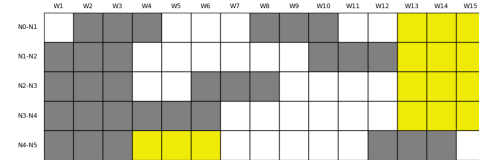
The last versions we implemented with Random Fit policies are the **RF stubborn** with break and with no_break versions. Also in this case the behavior, without the frequency shift limitation, is the same of the best effort version but of course performance may slightly differ since the blocks are randomly chosen.

3.4.6 Longest Fit

Longest Fit algorithms are a novel approach based on the concept of scanning the whole available spectrum to find the suitable block that covers the maximum number of links and starting the allocation from there. In other words the algorithms once individuated all the available blocks picks the longest one and then recursively performs the same procedure to complete the allocation for the uncovered links. We have developed four versions of this algorithm, two implementing the best effort logic and the other two the stubborn one, each type with both the no_break and can_break option. Both make sense only in case of wavelength conversion availability since when we need to satisfy the Spectrum Continuity constraint we can only allocate blocks of length equal to the number of links in the path, so it makes no sense to look for the longest blocks. Moreover in case of two blocks being both the longest one the one with lowest starting slot index is chosen and therefore an LF algorithm in case of no wavelength conversion is equivalent to a FF one. We can see from figures 3.10a and 3.10b how in both cases the algorithm picked the longest available spectrum block.



(a) First allocation

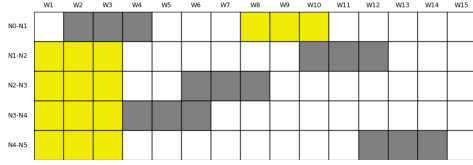


(b) Second allocation

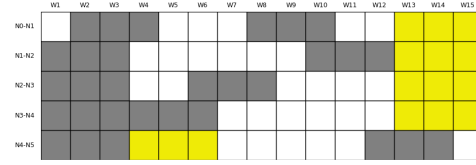
In the first allocation what happened is that the algorithm scanned all the available spectrum blocks looking at their last index that represents the number of covered links in order to find the longest block. What was found by the algorithm in this case is that we have two blocks that both cover 4 links which are: $[2; 0; 3; 4]$ and $[1; 12; 3; 4]$. Now among those two the algorithm picks the one with lower starting slot index and therefore the first of the two blocks is picked. Then once the longest block has been chosen the only link that remains uncovered is the first one, so the algorithm looks for blocks that cover only the first link and among those blocks it picks still the one with lowest starting slot index which in this case is the block that starts from the slot $W5$ (slot index 4). The same happens for the second allocation where after the allocation of the longest block to cover the last link we use the first available

block so the one with starting slot index 3.

With the no_break logic instead the situation slightly changes because after picking the same longest block of the previous case $([2; 0; 3; 4])$ here we can't pick also the other one $[1; 3; 3; 1]$ because it would mean breaking the block $[1; 3; 3; 2]$ and by doing this we would go against our constraint, therefore the algorithm's choice in this case is the block $[1; 7; 3; 1]$ which is the first available block that has at least one slot occupied in both previous and next links as we can see from figure 3.11a. From figure 3.11b instead we can see that the second allocation is the same of the previous case since in there we didn't need to break any spectrum block.



(a) First allocation



(b) Second allocation

The stubborn version also in this case, in absence of a wavelength shift limitation constraint, shows the same results. The LF algorithm has been designed with the objective of minimizing the number of required wavelength conversions as at each iteration we look for the longest block that is available and this may lead to less conversion and a less fragmented spectrum, so an higher spectrum usage efficiency lowering the blocking probability of connection requests. In particular this type of algorithm may be used in scenarios in which we have a certain pools of converters that can be shared among different nodes that as the one proposed in section 4.5 where converters are collected in the so called Wavelength Conversion Centers that can be exploited instead of placing converters on each node in which is required to perform a frequency shift. This approach, even though with physical constraints that don't allow its immediate implementation on real networks, can lead to significant reduction of implementation costs since wavelength converters in this way can be shared among sets of nodes. This topic will be further investigated at the end of the following chapter.

Chapter 4

Simulator engine and results

This chapter presents the complete structure of the EON simulator showing how the core components presented in chapter 3 are actually implemented and the role they play in the different simulations. Moreover the results of two simulation scenarios are presented, the first over a simple topology in order to compare the performance of the simulator w.r.t. the Java reference simulator from which it was inspired to ensure that simulations actually provide consistent results, so in the only scenario that OfNetSim supports. On the same topology also a comparison of the performances of the different versions of allocation algorithms that we have designed is provided. The second scenario is instead performed on a real-world topology where the different algorithms are again compared showing how specific versions can bring significant advantages that could be further enhanced by introducing other factors into consideration such as alternative routing approaches.

4.1 Simulator structure

In figure 4.1 we can see how the simulator is composed. At first we can see that we have a directory called **components** which includes all the core elements presented in chapter 3 plus an additional class called **simulation** that contains all the simulation logic and that, by interacting with the **Control-Plane**, allows us to generate and manage connection requests, allocate and release connectivity resources and compute statistics that will be stored in the output files. The next section will further explain what that class does. Then we have the **main** script that runs a single simulation, so contains all the in-

```

WCaaS/simulator/
|-- campaign.py          % campaign manager
|-- main.py              % run a single simulation
|-- components/          % core classes (Node, Link, etc.)
|-- config/              % topology files (e.g., SNDlib)
|   |-- topology_1.yaml
|   |-- topology_2.yaml
|-- input/                % parameter files
|   |-- default_parameters.yaml
|   |-- scenario_1.yaml
|-- output/               % results
|   |-- plot/             % generated figures
|   |-- simulation_1.csv

```

Figure 4.1: Structure of the simulator's project

formation, e.g. file paths and parameters needed to launch a single instance of a simulation, while the **campaign** script is used in order to run simulation campaigns by calling the main script several times with different input parameters. We introduced this flexibility in order to make the make easier the usage of different parameters files for different simulation keeping the ones that are in common unchanged. Then we have the **config** folder that contains the YAML files related to network topologies. Those topologies as mentioned in the previous chapter can be both custom, so created by the user, or taken from real-world topologies providers like SNDlib [17]. In fact we added to the simulator a **topology converter** that takes the text file from SNDlib and converts it into a YAML file with the exact structure we need to work with the simulator's logic. An example of this type of files is the following:

```

nodes:
  Amsterdam: [4.90, 52.35]
  Athens: [23.73, 38.00]
  Barcelona: [2.18, 41.37]
  Belgrade: [20.50, 44.83]
  Berlin: [13.40, 52.52]
  ...

```

```

links:
  L1: [Amsterdam, Brussels]
  L2: [Amsterdam, Glasgow]
  L3: [Amsterdam, Hamburg]
  L4: [Amsterdam, London]
  L5: [Athens, Palermo]
  ...

```

We can see that all the topology nodes must be identified with a name, or an ID, and two geographical coordinates, while all the links must an ID plus a tuple containing the IDs of the nodes that the link connects. Then we have the **input** folder that contains the files related to the configuration parameters for each simulation. We decided to implement a structure in which we have a default parameters file containing all the parameters needed to run a generic simulation, and a set of simulation-specific files that contain only the parameters which for that simulation differ from the default ones. This increases efficiency in both simulation set-up, since we don't need to manually write each time all simulation parameters, and campaigns execution since from one simulation to the other we only need to load the file that contains its specific parameters. Finally we have the **output** folder that will contain the results of our simulations divided in folders where for each simulation that was run a CSV file is created with the data for each traffic load point and the related plots about blocking statistics and costs that are further discussed in the following section.

4.2 Simulation engine

The simulation engine is the heart of the simulation as it is responsible for the setup and management of the simulation. In fact when instantiated the simulation class starts loading the parameters, first from the default YAML file and then the simulation-specific one and starting from there some values are automatically computed. An example of that is the number of spectrum slots per link, in fact by defining as B_{link} the total available bandwidth on a link and B_{slot} the band required for a slot that in our case is $12.5GHz$ the

number of slots per link n_{slots} is equal to:

$$n_{\text{slot}} = \left\lceil \frac{B_{\text{link}}}{B_{\text{slot}}} \right\rceil$$

Then among the other parameters that are automatically computed we have: the **leaving rate**

$$\mu = \frac{1}{T_h}$$

where T_h is the average holding time. Then we have the **normalized service bandwidths**:

$$b_i^{\text{norm}} = \left\lceil \frac{b_i}{B_{\text{slot}}} \right\rceil$$

where: b_i^{norm} is the number of spectrum slots needed to serve the request, B_i is the bandwidth demand of service i in Hz and B_{slot} as above is the bandwidth of a single slot also in Hz. Then we have the **traffic probabilities** where we need to distinguish between the case in which the user directly specifies them and the case in which they are computed automatically, in fact if nothing is specified we have that the traffic probability of service i is computed as:

$$p_i = \frac{\frac{1}{b_i^{\text{norm}}}}{\sum_{j=1}^{N_s} \frac{1}{b_j^{\text{norm}}}}$$

where N_s is the total number of services that are used in the simulation. We can see that p_i is computed as the ratio between the inverse of the number of slots required to fulfill the service and the same value summed over all the service types. This makes the services that have a lower normalized bandwidth more probable w.r.t. to the ones requiring more slots. Then we have the **full load capacity**, a parameter that represents the maximum traffic load the network can theoretically handle in Erlangs which, if not specified by the user, is computed as:

$$L_{\text{full}} = \frac{n_{\text{slots}} \cdot n_{\text{links}}}{\bar{b} \cdot \bar{h}}$$

where n_{slots} is the number of slots per link, n_{links} is the total number of links in the topology, $\bar{b}$ is the average normalized service bandwidth and $\bar{h}$ is the average number of hops per route. The last parameter that is automatically computed is the **load step** which is very important for the simulation logic, in fact inside each simulation several traffic points are tested, each one with its own traffic load which increases at every iteration of an amount equal to the step load which is computed as:

$$\Delta L = \frac{L_{\text{full}}}{P}$$

where P is the number of simulation points that are run inside a simulation.

After initializing all the parameters a pandas DataFrame structure is prepared for each simulation point and each of them is executed exploiting multiprocessing, so a worker is spawned for each point in order to reduce the total execution time. For each point the connection requests are generated according to the related traffic load and the control plane class, that will be described in the following section, takes care of routing and spectrum assignment. Throughout the process performance metrics are updated such as then number of blocked requests and costs. When the simulation finishes the following performance metrics are computed. At first we have the **overall network blocking ratio** which is computed as:

$$P_{\text{block}}^{\text{net}} = \frac{N_{\text{blocked}}}{N_{\text{total}}}$$

where N_{blocked} is the total number of blocked connection requests and N_{total} is the total number of connection requests. Then we have **per-service blocking ratios** that indicate how often requests of a specific service type are blocked. For example for a service i the blocking ratio is:

$$P_{\text{block}}^i = \frac{N_{\text{blocked}}^{(i)}}{N_{\text{total}}^{(i)}}$$

where $N_{\text{blocked}}^{(i)}$ is the number of blocked requests of service i and $N_{\text{total}}^{(i)}$ is the total number of requests of that same service. Then we have the **bandwidth blocking ratio** that for a specific service i represents the fraction of the total

required bandwidth that is blocked and therefore it is computed as:

$$B_{\text{bw}} = \frac{\sum_{i=1}^{N_s} N_{\text{blocked}}^{(i)} \cdot b_i}{\sum_{i=1}^{N_s} N_{\text{total}}^{(i)} \cdot b_i}$$

where N_s is the number of service types that are present in the simulation, b_i is the bandwidth demand in Hz of service i while $N_{\text{blocked}}^{(i)}$ and $N_{\text{total}}^{(i)}$ are the same as above, so blocked and total requests of service type i respectively. As mentioned before we also take into account allocation costs, in fact the total cost is computed as some fixed resource costs multiplied by the times they are used. We have:

$$C_{\text{total}} = \sum_{r=1}^{N_{\text{req}}} \left(C_{\text{slot}} \cdot s_r + C_{\text{conv,placed}} \cdot n_{\text{placed},r}^{\text{conv}} + C_{\text{conv,used}} \cdot n_{\text{used},r}^{\text{conv}} + C_{\text{node}} \cdot (h_r + 1) \right)$$

where N_r is the total number of connection requests that have been generated, C_{slot} is the cost of using one spectrum slot and therefore it is multiplied by s_r which is the number of slots required to serve request r . $C_{\text{conv,placed}}$ is the cost of placing a new converter and it is multiplied by $n_{\text{placed},r}^{\text{conv}}$ which is a number ≥ 0 that represents the number of converters that have to be placed to serve that request. Then we have the same thing but for the converters that are actually used, so that also already placed ones are comprehended, and finally the C_{node} and $(h_r + 1)$ represent the cost of using a node and the number of used nodes, as number of hops associated to that connection plus one, respectively. The values that we have associated to those fixed costs have already been discussed in section 3.2.3.

4.3 Control plane

For the control plane we have used an approach based on the concept of Software Defined Networking (SDN) in which we can exploit a global view of the network topology that allows a centralized path computation and spectrum allocation. The control plane class contains all the functionalities that are typical of the logical control plane, in fact it handles path computations,

spectrum management (allocation and release) and computation of some performance metrics. When a connection request arrives at the control plane at first a random source-destination node pair is chosen and the corresponding pre-computed shortest path is selected. In fact, as mentioned in section 2.3.1 we have adopted a Fixed Routing strategy to increase efficiency without interfering with the simulation goal. Then for that request according to the probabilities a service type is also randomly selected after which everything is ready for spectrum management operations. The control plane contains all the functions that are used to perform allocations from functions that explore the links' spectrum status creating spectrum objects for the different algorithms, and the different versions of the allocations algorithms that have been presented in section 3.1 included their standard versions that don't exploit wavelength conversion. The metrics that are computed in this class comprehend the average hops count and the min/max hops in a route. We can see the control plane as the brain of the simulator where the real decisions are taken according to some pre-defined policies.

4.4 Results

This section provides a description of the results obtained from our EON simulator on a simple simulation testbed. In fact at first the goal of the project was to create an EON simulator that correctly simulates allocation of connection requests, so in order to check the fulfillment of this requirement we have used an already existing simulator written in Java that was already validated, so we used it as a benchmark. This Java simulator was very limited in flexibility and the only scenario that it could simulate is a simple First Fit with no wavelength conversion on a very simple topology composed of 6 nodes and 16 links where every node is at unitary distance w.r.t. the adjacent ones. The complete topology is described by the following YAML file:

```
nodes:
  node1: [ 0.0, 0.0 ]
  node2: [ 1.0, 0.0 ]
  node3: [ 1.0, 0.0 ]
  node4: [ 2.0, 0.0 ]
  node5: [ 2.0, 0.0 ]
  node6: [ 3.0, 0.0 ]
```



Figure 4.2: Simple topology testbed

```

links:
  link1: [ node1, node2 ]
  link2: [ node2, node1 ]
  link3: [ node1, node3 ]
  link4: [ node3, node1 ]
  link5: [ node3, node5 ]
  link6: [ node5, node3 ]
  link7: [ node3, node4 ]
  link8: [ node4, node3 ]
  link9: [ node2, node5 ]
  link10: [ node5, node2 ]
  link11: [ node2, node4 ]
  link12: [ node4, node2 ]
  link13: [ node5, node6 ]
  link14: [ node6, node5 ]
  link15: [ node6, node4 ]
  link16: [ node4, node6 ]

```

This particular topology was hardcoded inside the Java simulator while in our version we exploited the flexibility of our topology load implementation, in fact, as mentioned in section 3.1.5, we can use both real-world topologies from SNDlib or custom user-defined topologies as in this case. Using our plotting function we can plot those topologies and this particular simple testbed is represented in figure 4.2. The peculiarity of this scenario is that since adjacent nodes are all at distance 1 in a 2D graph representing real geographical coordinates two nodes seem to be overlapped. More in detail nodes 2 and 3 are overlapped and the same for nodes 4 and 5.

4.4.1 Benchmark comparison

The first simulation we performed is the simplest possible, so the one that replicates the only available scenario of the Java benchmark simulator, which means that the algorithm used is the First Fit base with no wavelength conversion and setting all the other parameters in a way they perfectly replicate the Java scenario. Therefore we started setting the default parameters in the following YAML file:

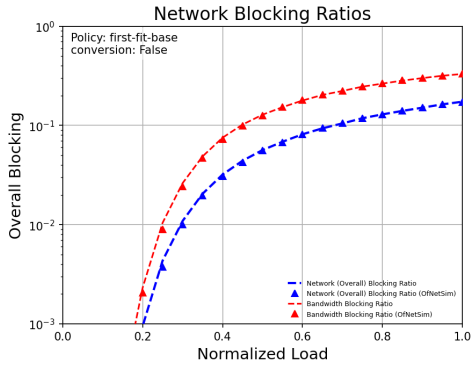
```
simulation:
  num_events: 200000 # Number of generated events (connections)
  traffic_probabilities_flag: False
  user_traffic_probability: []
  bandwidth: # Service bandwidths in Hz
    - 100000000000.0 # 100 GHz
    - 400000000000.0 # 400 GHz
  average_holding_time: 10.0 # Average holding time in seconds
  slot_bandwidth: 12500000000.0 # Grid spacing in Hz (12.5 GHz)
  available_bandwidth: 400000000000.0 # Links bandwidth
  maximum_network_load_flag: False
  user_maximum_network_load: 1000
  seed : 635 # Seed for the random number generator
  points: 20 # Number of simulated load points
  topology_file: "topology_java.yaml"
  conversion: True
  policy: "first_fit_best_effort"
  can_break: True
```

From there we set the simulation-specific parameter in order to use the correct allocation algorithm with its related settings. In order to do so we have another YAML file called `sim_1` that contains the parameters that we need to overwrite w.r.t. the default ones, therefore we have:

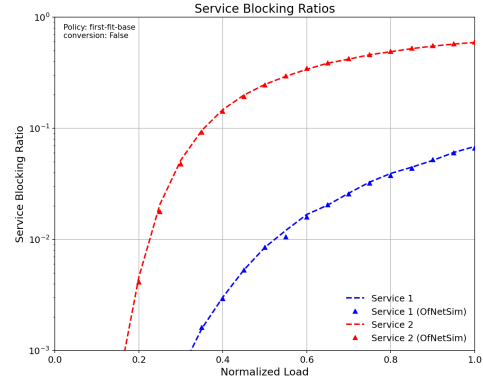
```
simulation:
  conversion: False
  policy: "first_fit_base"
```

As we can read from those files in this simulation we are generating 200 000 connection requests between random topology endpoints and each link has

4THz of available spectrum which, by using 12.5GHz spacing, result in 320 spectrum slots. We have two service types, one requiring 100GHz (so 8 contiguous spectrum slots) and the other 400GHz (32 slots). The traffic load ranges from 13 to 261 Erlang, which is simulated in 20 load points, therefore 20 simulations are actually run, the first with a load of 13 Erlang (5% of the full load) and the last one with 261 Erlang of traffic (100% of the full load).



(a) Network blocking



(b) Service blocking

The results of the simulation are graphically represented in figures 4.3a and 4.3b where the dashed lines represent the results obtained through our python EON simulator while the triangles are the results of the Java simulator. As we can see the results perfectly match one with each other with very small differences that may be due to the fact that the same random seed used in different programming languages may not lead to the exact same generated number, but this surely proves that the simulator properly works and can be therefore used to compare the performance of different allocation algorithms which was our second goal of the project.

4.4.2 Comparison of First Fit algorithms

Once confirmed that the python EON simulator provides consistent results we compared the performances of the different algorithms we developed, as reported in table 3.1. The first block of simulations regards the First Fit policy algorithms with all the possible parameters combinations which are summarized in table 4.1.

Sim-specific file	Policy	Conversion	Can Break
sim_1	Base	no	yes
sim_2	Base	yes	yes
sim_3	Best-Effort	yes	yes
sim_4	Best-Effort	yes	no
sim_5	Stubborn	yes	yes
sim_6	Stubborn	yes	no

Table 4.1: Simulations with First Fit algorithms

The topology used was the same of the previous section, so the one in figure 4.2, and also the traffic load parameters, links' bandwidth and services remain the same. In order to compare the results we look at the blocking probability of connection requests for each algorithm which are collected in figure 4.4.

The graphs show the overall blocking of the requests and the bandwidth blocking w.r.t. the normalized network load where 1 corresponds to the 261 Erlang as previously mentioned. By comparing the first two graphs we can see the benefits of implementing wavelength conversion which reduces network blocking and as a consequence also bandwidth blocking. In fact if we look at the output CSV files containing the precise values of those performance metrics we can see that for point 20, so when we generate 200 000 connection requests (full load) in the first simulation bandwidth blocking is 33.2% while by allowing conversion it falls to 31.1%. This means that we increase the spectrum usage of 2.1% which is exploited to serve more connection requests, in fact the graph shows that less requests are blocked when using conversion. By means of the best effort version of the First Fit algorithm, sim 3 and 4, we don't get a relevant improvement, this means that by introducing the spectrum blocks logic we don't really save spectrum since in this particular case spectrum fragmentation is not the limiting factor. In simulation 4 there is a significant increment in the blocking probabilities for low traffic conditions. This happens because we are using the no break logic that doesn't allow the algorithm to pick spectrum blocks that don't have at least one of the slots occupied both above and below along the path. This condition in fact, in presence of just a few already occupied blocks we have situations in which we can only partially complete the path and after the conversion the algorithm can't find a suitable block to complete the allocation and therefore it blocks the request. Simulations 5 and 6 show that in case of can break conditions

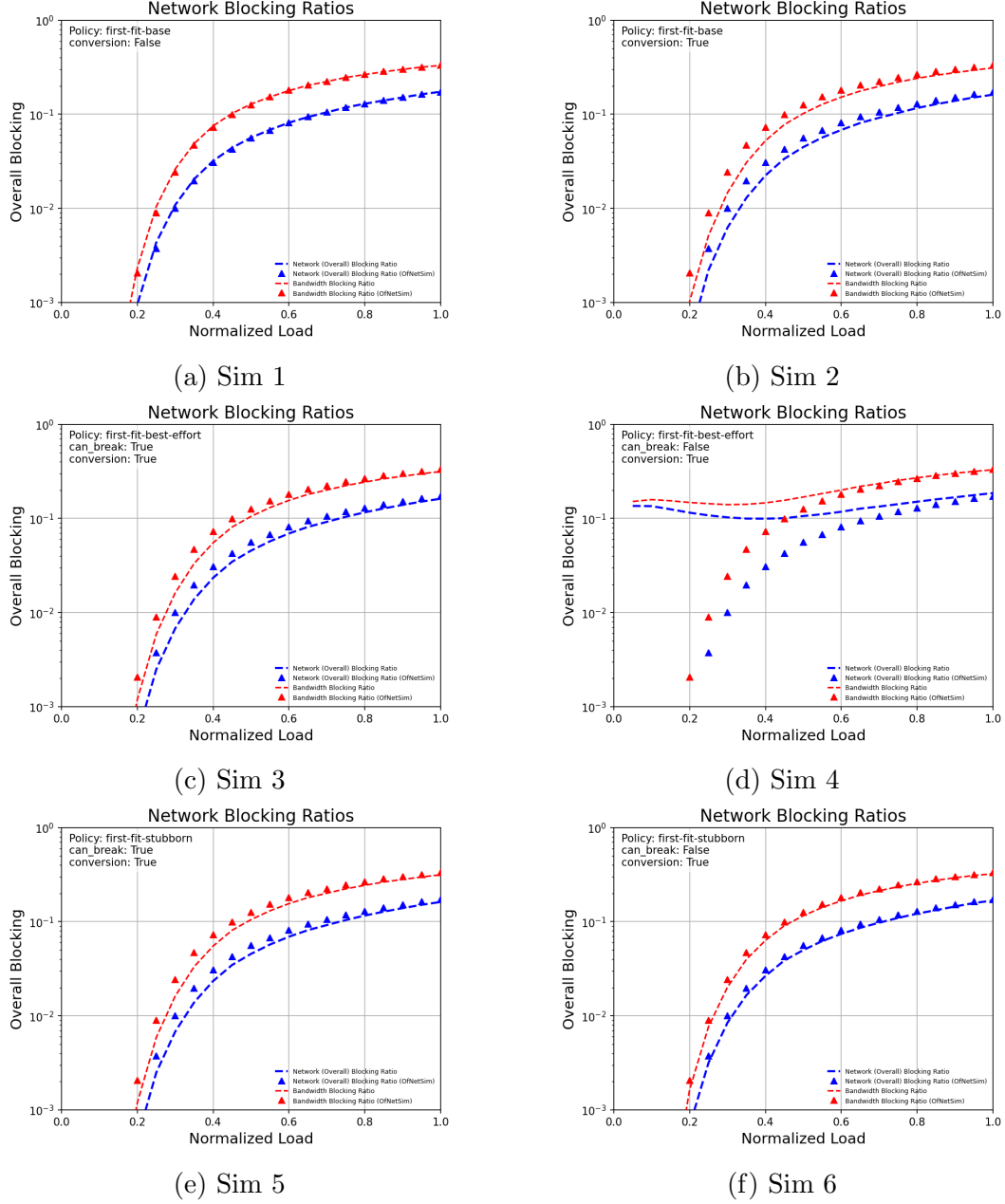


Figure 4.4: Network blocking of First Fit algorithms.

the performance of the stubborn version of the algorithm performs exactly the same of the best effort version since as mentioned in section 3.1 when we don't have wavelength conversion range limitation there is no gain in performance by using the stubborn version, but in case of using the no break logic conditions change. The stubborn version solves the issue of the no break that we had in the best effort algorithm because when a path is only partially completed at the first attempt, instead of blocking the algorithm goes back to the first chosen block and tries a different entry until it finds a suitable allocation, therefore the only condition in which it blocks is that there is no possible valid allocation for the whole path.

For what concerns the costs we can say that implementing wavelength conversion slightly increases the costs, in fact during the simulation we collect metric about average cost per request using as reference costs the ones in section 3.2.3 and using the equation in section 4.2. Table 4.2 shows the comparison in terms of average event costs of the base version of the algorithm with and without conversion for different traffic load points.

Point	No conversion	Conversion
1	22.72	24.72
3	22.71	25.16
5	22.52	25.21
10	20.30	23.95
15	18.30	21.83
20	17.00	20.07

Table 4.2: Average event cost with and without conversion.

From the table we can also see that there is a trend in the cost to decrease as the network load increases. This is due to the fact that we have blocking, in fact we are talking about average cost per event and when we have blocked connection requests those requests don't have a cost since they are not using any resource, therefore lowering the cost on average. The same thing happens in both conversion and non conversion scenarios, but when we use conversion we can also notice that before decreasing due to blocking the costs slightly increase. This is due too the fact that we deploy (and use) wavelength converters that have costs associated, so at the beginning the converters that are placed are averaged over a lower number of connections, than comes the increase in blocking that lowers again the average cost.

4.4.3 Comparison of Random Fit algorithms

This section provides an overview of the results of simulations performed using the different versions of the Random Fit algorithm with the different parameters combinations as reported in table 4.3. The simulations are run with the exact same topology and traffic loads of the previous section in order to allow a fair comparison between the allocation strategies.

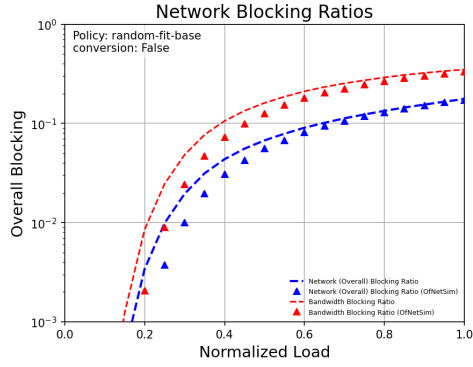
Sim-specific file	Policy	Conversion	Can Break
sim_7	Base	no	yes
sim_8	Base	yes	yes
sim_9	Best-Effort	yes	yes
sim_10	Best-Effort	yes	no
sim_11	Stubborn	yes	yes
sim_12	Stubborn	yes	no

Table 4.3: Simulations with Random Fit algorithms

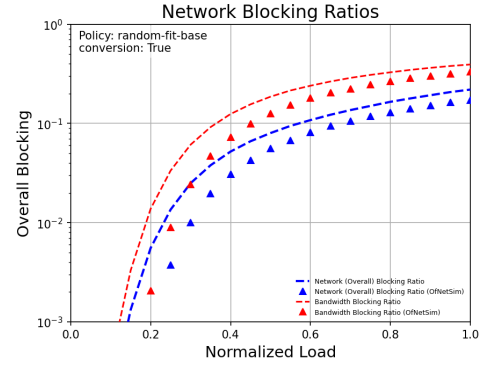
Graphs reporting simulations results are shown in figure 4.5, where we can notice similar behavioral patterns w.r.t. the the First Fit simulations. The first thing that can be noticed is that assigning the spectrum in a random manner increases in all cases the blocking probability and that in this case introducing wavelength conversion does not decrease it. In simulation 10 where we have the best effort version implementing the no break logic we have the same problem we noticed with the respective First Fit version for low traffic loads, even though in this case its impact is lower. The problem is fixed by adopting the stubborn version which is able to recursively find a suitable allocation in case of blocking.

In this case also in terms of costs the simulation that provided the lowest average event cost is the base version without wavelength conversion which suggests that when a random policy is adopted neither the adoption of spectrum block logic and wavelength conversion help improving network's performance. Table 4.4 shows the cost difference between the base versions of the Random Fit algorithms with and without conversion.

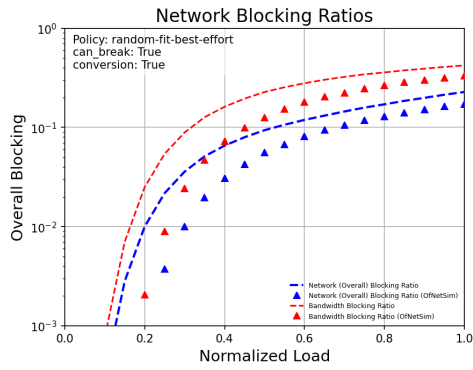
From the comparison between tables 4.2 and 4.4 it may seem that using a Random Fit algorithm is convenient in terms of average costs per connection w.r.t. First Fit implementations, and this is true, but it comes with a cost in terms of blocking. In fact since blocked requests don't consume resources



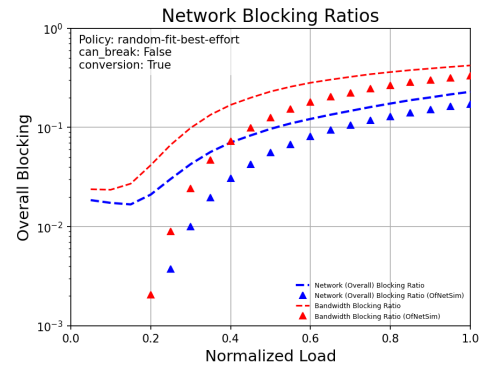
(a) Sim 7



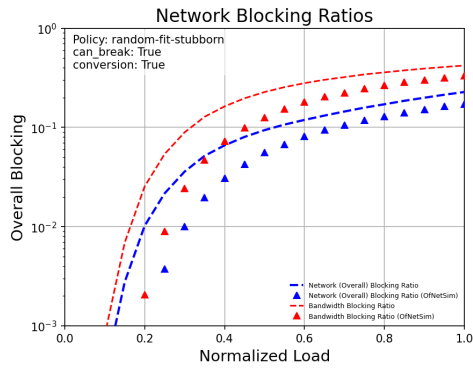
(b) Sim 8



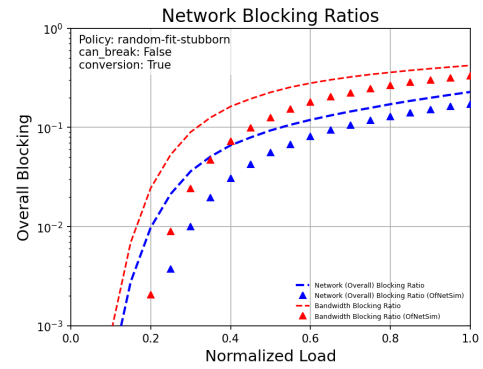
(c) Sim 9



(d) Sim 10



(e) Sim 11



(f) Sim 12

Figure 4.5: Network blocking of Random Fit algorithms.

Point	No conversion	Conversion
1	22.72	26.31
3	22.52	26.56
5	22.18	25.70
10	19.53	22.98
15	17.69	21.10
20	16.61	20.06

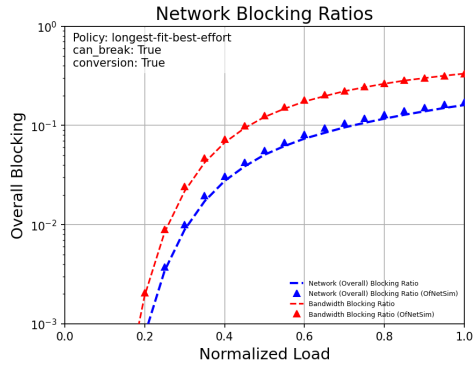
Table 4.4: Average event cost with and without conversion.

they lower the average of cost, therefore we can see it as a trade-off between utilization costs and blocking probability.

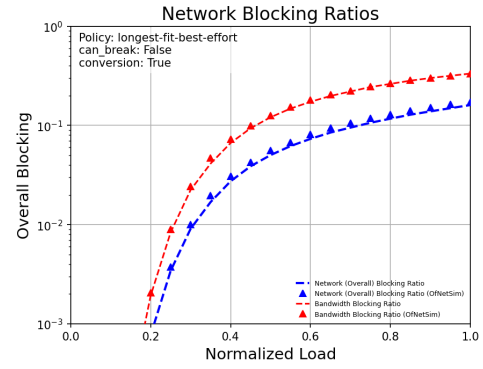
4.4.4 Comparison of Longest Fit algorithms

Now we give a closer look to the Longest Fit algorithms. This type of algorithms is a new path to explore that was enabled by wavelength conversion. It is based on the fact that the algorithm can scan all the spectrum and start allocating from the spectrum block that covers the highest number of links, then covering the rest of the path with the same logic on different slots. In case of no conversion all the spectrum blocks have the same length which is the length of the path, while in case of conversion we can chose to start allocating from the first available slot (FF), from a random available block (RF) or from the longest available block (LF). The LF algorithm has been designed with the idea of reducing spectrum fragmentation leading to an improved usage of the spectrum. As we have seen from previous simulation in a simple topology like the one in figure 4.2 fragmentation is not a limiting problem, therefore we didn't expect the algorithm in all its versions to bring a significant advantage on this particular scenario, while it may be a good choice for sparse networks in which we have longer paths and many more links. Table 4.5 summarizes the algorithm versions that have been designed and tested:

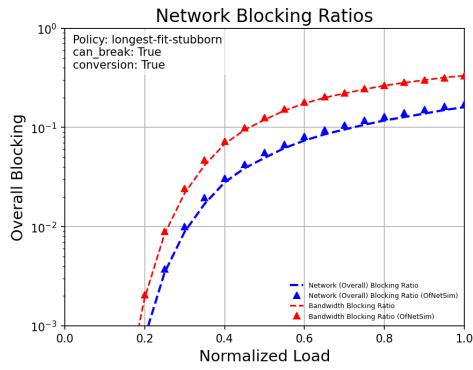
Figure 4.6 show the results of the simulations where we can that as expected there is no significant performance improvement w.r.t. the best case so fare since fragmentation is not a big problem in such a small network topology. The performance is very similar for all the versions of the algorithm. Can be noted that in this case both no break versions of the algorithm show any sign of increased blocking for low traffic loads.



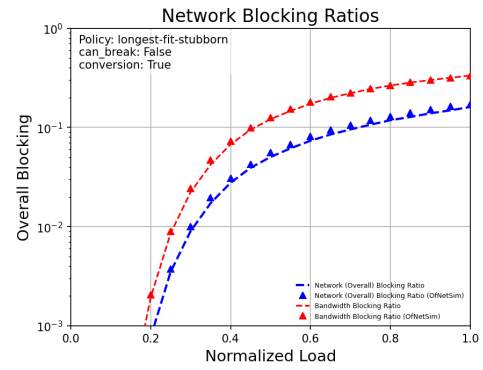
(a) Sim 13



(b) Sim 14



(c) Sim 15



(d) Sim 16

Figure 4.6: Network blocking of Longest Fit algorithms.

Sim-specific file	Policy	Conversion	Can Break
sim_13	Best-Effort	yes	yes
sim_14	Best-Effort	yes	no
sim_15	Stubborn	yes	yes
sim_16	Stubborn	yes	no

Table 4.5: Simulations with Longest Fit algorithms

Also from a cost viewpoint the four variants of the Longest Fit algorithm perform similarly. Table 4.6 shows the costs of the two can break logic versions, the first is best-effort and the second is stubborn.

Point	Best-Effort	Stubborn
1	22.72	22.72
3	22.72	22.72
5	22.62	22.62
10	20.79	20.81
15	18.88	18.84
20	17.74	17.72

Table 4.6: Average event cost best-effort and stubborn.

Among the different policies, FF, RF and LF, the LF is the one showing the best trade-off between blocking probability and costs in this specific simulation scenario, in fact w.r.t. FF its blocking probability is just slightly higher but still very similar and in terms of costs it is lower.

4.4.5 Test on a real-world topology

Since our implementation allows it we can perform simulation tests on a real-world network topology taken from SNDlib [17] and correctly converted with our topology converter script as described in section 3.1.5. For this simulation I picked the topology in figure 4.7 which represents a large scale optical network on which we can run the best-performing allocation algorithms based on the previous simulations.

Table 4.7 summarizes the approaches of allocation algorithms and parameters combinations that we picked for this simulation in order to assess a qualitative advantage of employing wavelength conversion and to confirm the presence

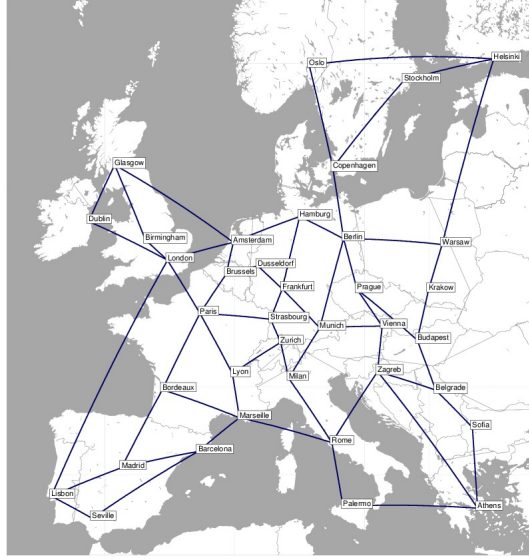


Figure 4.7: Optical Network Topology with 37 nodes and 57 links

of specific behavioral patterns that we noticed in the previous section performing simulations on a small custom topology.

Policy	Version	Conversion	Can Break
FF	Base	no	-
FF	Base	yes	-
FF	Best-effort	yes	yes
FF	Stubborn	yes	no
FF	Stubborn	yes	yes
LF	Best-effort	yes	yes
LF	Stubborn	yes	no
LF	Stubborn	yes	yes

Table 4.7: Simulations on the real-world topology

Also in this case we generate 200 000 connection requests as in the previous simulations but since we have an higher number of links and nodes the network load will be higher. In fact the full load on the new topology (point 20 of the simulations) is 717 Erlang. The plots in figure 4.8 show the results of the simulations in terms of network blocking.

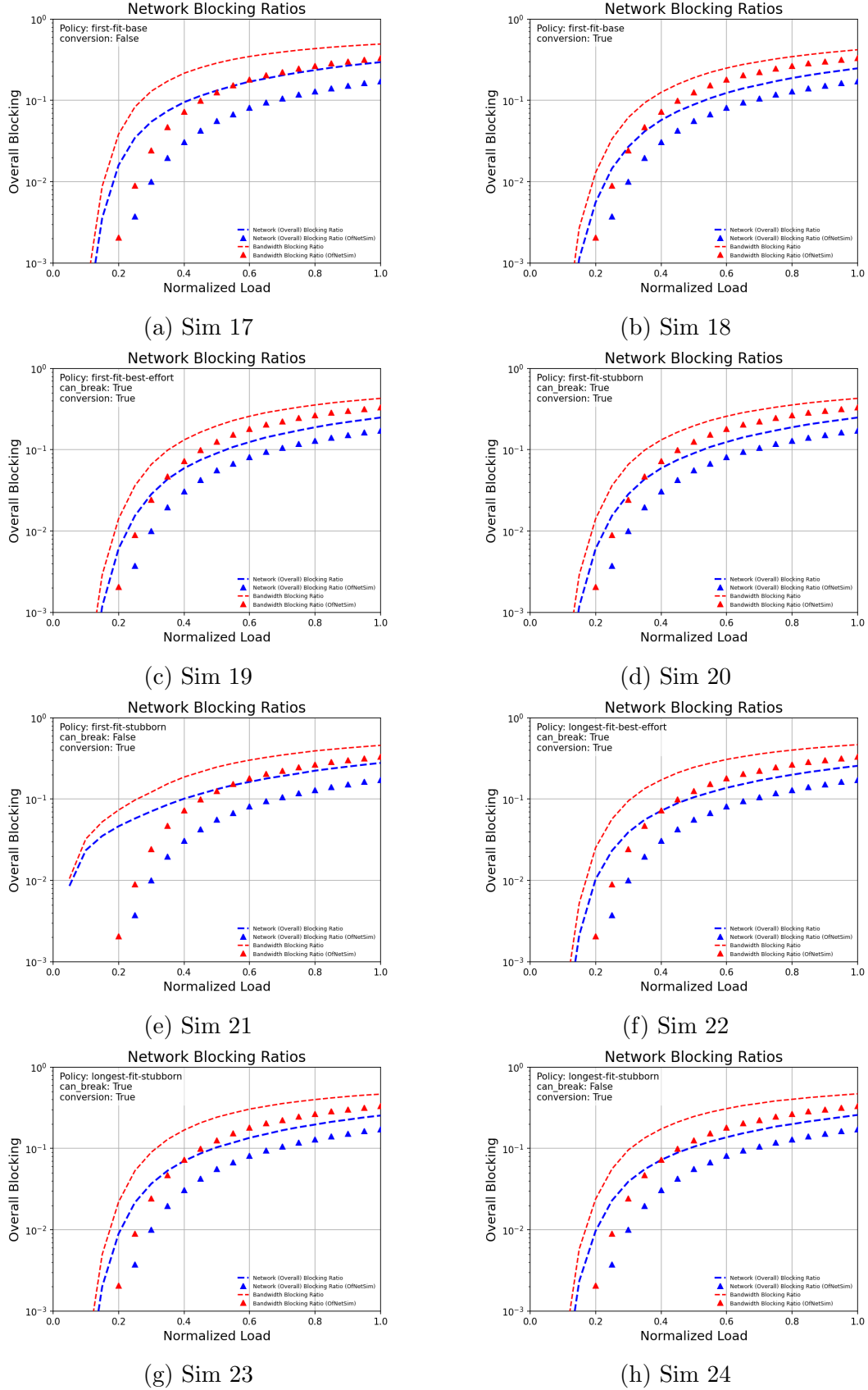


Figure 4.8: Network blocking for topology in image 4.7.

Is important to point out that the triangles in the pictures refer to the results of the Java reference simulator on the simple topology, so in this case they can simply be ignored, they are just a reference in order to be able to visually judge how good or bad an algorithm is performing. Table 4.8 reports the network blocking probabilities of all the simulations at load points 5, 10, 15 and 20 that corresponds respectively to 180, 360, 538 and 717 Erlang of network traffic.

Simulation	Load 5	Load 10	Load 15	Load 20
17	0.035	0.132	0.220	0.300
18	0.015	0.089	0.172	0.248
19	0.015	0.091	0.172	0.249
20	0.015	0.091	0.173	0.249
21	0.058	0.132	0.207	0.278
22	0.023	0.105	0.185	0.256
23	0.021	0.103	0.182	0.254
24	0.023	0.105	0.186	0.258

Table 4.8: Network blocking on the real-world topology

In this scenario we can better see the advantage that wavelength conversion brings. In fact among all the simulations the highest network blocking happens when we don't allow conversion where 30% of the incoming requests can't be served by the network and are therefore blocked. Among the algorithms that were tested the best performing one in this case is the FF base with wavelength conversion, so the one that allocates the first slice on each link without considering spectrum blocks. This algorithm shows in this scenario the lowest blocking for all the considered load points. While in the simple topology the stubborn version of the first fit brought better results in this case it performs slightly worse than the base version but still better than the no conversion case. Even though it doesn't seem a big gain in terms of blocking the 5.2% improvement it actually represents in our scenario 10 400 connection requests that can be now served by the network that in the absence of wavelength conversion was not able to admit, this means that in a huge congestion case such as the one simulated by generating 200 000 requests only by introducing the possibility to shift the wavelength of the signals inside nodes we can establish up to 10 400 more connections between topology endpoints which may be a significant advantage in many use cases.

4.5 Wavelength conversion as a service

While the First Fit and Random Fit policies have been already proposed in the literature, even though without considering wavelength conversion, the Longest Fit is a novel approach that was introduced in [19] specifically to address a scenario that we called wavelength conversion as a service (WCaaS). In particular this scenario refers to SDN-controlled optical networks where the hierarchical approach of Software-Defined Networking enables multi-domain orchestration and management which can be integrated with Network Function Virtualization (NFV) in order to provide more flexibility in lightpath setup and improve network's efficiency. In this type of networks thanks to the SDN framework we can create abstractions that describe the available connectivity resources that are provided to the SDN-controller in order to take smarter decisions with a complete view of what kind of resources are present and where. Following this paradigm and the concept of Everything as a service (XaaS), that in recent years facilitated the deployment of NFV technologies, we can theoretically treat wavelength conversion as a virtual network function (VNF) which is made available to the SDN-controller to serve user requests. This allows an optimized usage of wavelength conversion resources leading to a lower blocking probability during the setup of lightpaths for the provisioning of end-to-end network services. Under this assumption we denoted this scenario of wavelength conversion as VNF as Wavelength Conversion as a Service (WCaaS).

Chapter 5

Conclusion and future work

This chapter provides an overview of what can be learned from the simulations that we run providing a final comparison between the different allocation strategies that were tested and how the EON simulator can be improved in order to gain flexibility and lead to more significant improvements in terms of blocking probabilities and costs.

5.0.1 Conclusions

During the simulations that we run we could clearly see that the introduction of wavelength conversion in Elastic Optical Networks has an impact in terms of reducing network (and service) blocking. We have seen how depending on the scenario different algorithms may lead to a better result. We started by simulating a simple custom topology with 6 links and 6 nodes in which the best performing algorithms have been the First Fit and Longest Fit in their stubborn version with wavelength conversion, so the one that recursively tries to find a suitable allocation, even though since this topology is very small and therefore paths have a low average number of hops, the improvement in terms of blocking was very limited. The impact of conversion was more clear in a wider real-world network topology with 37 nodes and 57 links where instead the best performing algorithm resulted to be the base version of the First Fit, outscoring the case of no wavelength conversion by 5.2% which in numerical terms in our case, since the generated connection requests were 200 000, with conversion the network can allow 10 400 more connections which in very congested scenario such the one we simulated may represent a significant improvement for many

use cases. We can now look how this increment can be improved in future developments of the simulator.

5.0.2 Future work

The simulator that we designed and built is of course a starting point to what can become an holistic representation of an Elastic Optical Network that provides a complete flexibility over network management functionalities and control plane logic. In fact the goal of the project was to design a reliable EON simulator with which we could compare the performance of several spectrum allocation algorithms and investigate potential benefits of wavelength conversion in a reliable way. Currently the implementation allows the user to run simulation campaigns selecting a network topology that can be directly created by the user or converted from SNDlib where real-world optical network topologies are present. Moreover the user can already customize the simulation by means of several parameters, in fact can be currently selected: the number of connection requests that we want to generate, the probabilities that those requests are of a service type or another, the bandwidth required by the different network services, the average holding time of requests, the bandwidth of a spectrum slot (even though in order to match the Flexible Grid standard it has to be kept equal to 12.5GHz), the bandwidth that is available on each link, the maximum network load in Erlang that can be offered to the network, the seed for random numbers generation for reproducibility purposes, the number of different loads that will be tested on the network inside one simulation (also called simulation points) and finally the allocation algorithm to be used and its related parameters, so availability or not of wavelength conversion and can break or no break logic.

What is still very limited is how the routes are computed in order to serve connection requests. In fact, as mentioned in section 2.3.1, currently we adopt a fixed routing strategy approach which means that, in order to reduce simulation times, the paths between each pair of connection endpoints is pre-computed using a shortest path algorithm which in our case is Dijkstra's and when a request arrives to the network only an endpoint-based lookup is needed in order to find the path instead of actively computing it. In order to improve the flexibility in this area other approaches can be tested which may lead to a reduction in blocking probability and costs. One of those approaches is to adopt a fixed-alternate routing strategy in which we don't pre-compute only

one shortest path but more than one, for example using a k -shortest path algorithm which enables the possibility to try the allocation of multiple paths in case the first ones fail, therefore opening the door to the discovery of a path that even though is not the shortest possible it still enables the connection to be established which is better than a blocked request. It is not granted that in every topology this translates into an improved network performance because in this way to serve some connections we use more resources than the minimum needed amount and therefore this may lead to the blockage of other requests. Another approach that can be implemented is the adaptive routing strategy which shifts from a proactive to a reactive approach to path discovery. In fact adaptive routing aims at finding the best suitable path by accounting for the current state of the links' spectrum when the requests arrive in the network and tries to allocate the less used links first in order to optimize the overall spectrum availability. This is a very interesting approach that is possible to implement in our EON simulator since we have assumed to have a global view of network's resources and therefore we are potentially able to look at the status of the links' spectrum for each connection request that has to be served. A potential drawback of this approach is that it may be very computationally demanding particularly in scenarios in which many requests are generated which may lead to extremely long simulation times.

Another flexibility improvement that can be discussed is the possibility for the algorithms to choose the modulation format that is used to serve each request based on parameters such as required service bandwidth and length of the path. As seen in section 2.3.2 in fact the modulation format can be a key enabler of future EONs allowing for the transmission of increasingly higher bitrates inside our networks. In this way we would reach a complete approach to the RMSA problem as described in section 2.3 where routing, modulation format choice and spectrum assignment have maximum flexibility.

Moreover our current implementations of the FF, RF and LF algorithms and the related functions to analyze the spectrum and individuate blocks can be optimized in order to reduce the simulation times with which in a few cases, especially the most complex versions of the algorithms (so the stubborn ones) we struggled a bit, so a possible work can be done in this sense to reduce the computational complexity of the algorithms and also to find even more efficient algorithms.

Appendix A

Simulator components

A.1 Core components

A.1.1 Node

```
class Node:
    def __init__(self, topology: np.ndarray, node_id: int
, num_slots: int) -> None:
        self.input_ports: np.ndarray = np.zeros(topology.
            shape[0], dtype=int)
        self.output_ports: np.ndarray = np.empty(topology
            .shape[0], dtype=object) # Array of Link
            objects
        for i in range(topology.shape[0]): # topology.
            shape[0] is the number of nodes in the
            topology
            # if there is a connection between this node
            and node i we initialize a Link object and
            add it to the output_ports list
            if topology[node_id, i] != 0.0:
                self.output_ports[i] = Link(topology[
                    node_id][i], node_id, i, num_slots)
            # if there is a reverse link (so an incoming
            one) we only store its cost
            if topology[i, node_id] != 0.0:
```

```

        self.input_ports[i] = topology[i, node_id
        ]
self.node_id: int = node_id
self.debug = False # Debug flag

# Conversione wavelength
self.converters = defaultdict(int) # {(shift,
    converter_id): active_conversions}
self.max_conversions = 5 # Massimo numero di
    conversioni contemporanee per convertitore
self.placement_cost = 100 # Costo installazione
self.usage_cost = 5 # Costo utilizzo

def enable_debug(self, enable=True):
    """Toggle debug output"""
    self.debug = enable

def use_converter(self, shift: int, debug_header: str
= "") -> tuple[int,int]:
    """Request wavelength conversion for a specific
    shift"""
    placement = 0
    usage = self.usage_cost

    # search all converters for this shift
    for converter_id in [k for k in self.converters
        if k[0] == shift]:
        # if converter has spare capacity, use it
        if self.converters[converter_id] < self.
            max_conversions:
            self.converters[converter_id] += 1
            if self.debug:
                print(f"{debug_header}␣[Node␣{self.
                    node_id}]␣Converter␣used:␣shift={
                        shift},␣id={converter_id},␣active
                        ={self.converters[converter_id]}/{
                            self.max_conversions}")
            return (0, usage) # reused converter:
                only usage cost

```



```

                                shift},{converter_id},{active
                                =0")
    return

```

A.1.2 Link

```

class Link:
    def __init__(self, link_cost: float, input_node: int,
                  output_node: int, num_slots: int) -> None:
        self.link_cost: float = link_cost
        self.input_node: int = input_node
        self.output_node: int = output_node
        self.spectrum_status: np.ndarray = np.ones(
            num_slots, dtype=int)

```

A.1.3 Spectrum

```

class Spectrum:
    def __init__(self, size: int, position: int) -> None:
        self.size: int = size
        self.position: int = position

```

A.1.4 Connection

```

class Connection:
    def __init__(self, time_stamp: float, route: List[int],
                  spectrum: Spectrum, service_type: int, id_conn:
                  int, total_cost: float = 0.0, used_converters=None
    ):
        self.time_stamp: float = time_stamp
        self.route: List[int] = route
        self.spectrum: Spectrum = spectrum
        self.service_type: int = service_type
        self.id_conn: int = id_conn
        self.total_cost = total_cost

```

```

        self.used_converters = used_converters if
            used_converters is not None else []

def __lt__(self, other: Any) -> bool:
    """
    Comparison method for sorting connections based
    on their release time.
    This is used by heapq and other sorting
    mechanisms to maintain a time-ordered queue.
    """
    return self.time_stamp < other.time_stamp

```

A.1.5 Route

```

class Route:
    def __init__(self, route: List[int], route_cost:
        float, route_id: int) -> None:
        self.route: List[int] = route
        self.route_cost: float = route_cost
        self.route_id: int = route_id

    def getRoute(self) -> List[int]:
        return self.route

```

A.1.6 Topology

```

class Topology:
    def __init__(self, yaml_file: str) -> None:
        self.yaml_file: str = yaml_file
        self.num_nodes: Optional[int] = None
        self.topology: Optional[np.ndarray] = None
        self.nodes: List[str] = []
        self.links: List[Any] = []
        self.node_positions: Dict[str, Tuple[float, float]] = {}

    def load_topology(self) -> None:

```

```

"""
Reads the YAML data, extracts nodes, links, and
positions.
Then creates a node_indices dictionary to map
node names to indices.
The adjacency matrix is initialized as a zero
matrix and updated for each link with the
Euclidean distance between connected nodes.
"""
with open(self.yaml_file, 'r') as file:
    data = yaml.safe_load(file)

"""
Expected YAML topology format:
nodes:
    node1: [x_node1, y_node1]
    node2: [x_node2, y_node2]
links:
    link_id: [source_node, destination_node]
"""

self.nodes = list(data['nodes'].keys())
self.links = list(data['links'].values())
self.node_positions = {node: tuple(coords) for
    node, coords in data['nodes'].items()}

node_indices: Dict[str, int] = {node: idx for idx
    , node in enumerate(self.nodes)}
N: int = len(self.nodes)

# Change dtype to float since distances will be
floating-point values
self.topology = np.zeros((N, N), dtype=float)

for link in data['links'].values():
    node1, node2 = link
    i: int = node_indices[node1]
    j: int = node_indices[node2]

```

```

        # Calculate Euclidean distance between nodes
        x1, y1 = self.node_positions[node1]
        x2, y2 = self.node_positions[node2]
        distance = math.hypot(x2 - x1, y2 - y1)

        # Fill in the symmetric adjacency matrix with
        the distance as cost
        self.topology[i][j] = distance
        self.topology[j][i] = distance

def show_topology(self) -> None:
    """
    Visualizes the topology using a headless
    Matplotlib backend and opens the plot.
    """
    if self.topology is None:
        raise RuntimeError("Topology not loaded. Call
                           load_topology() first.")

    try:
        # Create figure with headless backend.
        fig = plt.figure(figsize=(14, 10))
        G = nx.Graph()

        # Add nodes and their positions.
        for node in self.nodes:
            G.add_node(node, pos=self.node_positions[
                node])

        # Add edges if nodes are connected.
        for i in range(len(self.nodes)):
            for j in range(i + 1, len(self.nodes)):
                if self.topology[i][j] > 0:
                    G.add_edge(self.nodes[i], self.
                               nodes[j])

        # Draw the network.
        pos = nx.get_node_attributes(G, 'pos')
        nx.draw(G, pos, with_labels=True, node_size
                =1000, node_color='lightblue',

```

```

        font_size=9, font_weight='bold',
        edge_color='gray', alpha=0.8)
edge_labels = {
    (u, v): f"{self.topology[self.nodes.index
        (u)][self.nodes.index(v)].1f}"
    for u, v in G.edges()
}
nx.draw_networkx_edge_labels(G, pos,
    edge_labels=edge_labels, font_size=14,
    font_color='darkred',
    alpha=0.7)
plt.title("Network_Topology_Visualization",
    fontsize=14, pad=20)
plt.axis('equal')
plt.grid(True, linestyle='--', alpha=0.4)
plt.gca().set_facecolor('whitesmoke')

# Save the figure to a temporary file.
temp_dir: str = tempfile.gettempdir()
temp_file: str = os.path.join(temp_dir, "
    topology_plot.png")
plt.savefig(temp_file, bbox_inches='tight',
    dpi=150)
plt.close(fig)

# Verify the file exists.
if not os.path.exists(temp_file):
    raise FileNotFoundError(f"Plot_not_saved_
        to:{temp_file}")

# Open the file using the platform's default
image viewer.
if platform.system() == 'Windows':
    subprocess.run(['start', temp_file],
        shell=True, check=True)
elif platform.system() == 'Darwin':
    subprocess.run(['open', temp_file], check
        =True)

```

```
        else:
            subprocess.run(['xdg-open', temp_file],
                           check=True)

            print(f"Plot opened at: {temp_file}")

    except subprocess.CalledProcessError as e:
        print(f"Failed to open image: {e}")
    except Exception as e:
        print(f"Visualization error: {str(e)}")
```


Bibliography

- [1] *ITU-Facts-and-figures*. URL: <https://www.itu.int/itu-d/reports/statistics/2024/11/10/ff24-internet-use/>.
- [2] Varsha Lohani, Anjali Sharma, Yatindra Nath Singh, Kumari Akansha, Baljinder Singh Heera, and Pallavi Athe. “Optical networks”. In: *arXiv preprint arXiv:2408.07478* (2024).
- [3] Gianluca Davoli, Raffaele Di Tommaso, Alessio Giorgetti, and Carla Raffaelli. “Impact of lightpath selection on end-to-end service orchestration in disaggregated optical networks”. In: *2024 International Conference on Optical Network Design and Modeling (ONDM)*. IEEE. 2024, pp. 1–6.
- [4] *ITU-T G.694.1 standard*. URL: <https://www.itu.int/rec/T-REC-G.694.1>.
- [5] Ori Gerstel, Masahiko Jinno, Andrew Lord, and SJ Ben Yoo. “Elastic optical networking: A new dawn for the optical layer?” In: *IEEE communications Magazine* 50.2 (2012), s12–s20.
- [6] Masahiko Jinno, Hidehiko Takara, Yoshiaki Sone, Kazushige Yonenaga, and Akira Hirano. “Multiflow optical transponder for efficient multilayer optical networking”. In: *IEEE Communications Magazine* 50.5 (2012), pp. 56–65.
- [7] Antonio Napoli, Marc Bohn, Danish Rafique, Alexandros Stavdas, Nicola Sambo, Luca Potì, Markus Nölle, Johannes Karl Fischer, Emilio Riccardi, Annachiara Pagano, et al. “Next generation elastic optical networks: The vision of the European research project IDEALIST”. In: *IEEE communications magazine* 53.2 (2015), pp. 152–162.
- [8] Kazuro Kikuchi. “Fundamentals of coherent optical fiber communications”. In: *Journal of lightwave technology* 34.1 (2015), pp. 157–179.

- [9] Nicola Andriolli, Alessio Giorgetti, Piero Castoldi, Gabriele Cecchetti, Isabella Cerutti, Nicola Sambo, Andrea Sgambelluri, Luca Valcarengi, Filippo Cugini, Barbara Martini, et al. “Optical networks management and control: A review and recent challenges”. In: *Optical Switching and Networking* 44 (2022), p. 100652.
- [10] Mehdi Habibi and Hamzeh Beyranvand. “Impairment-aware manycast routing, modulation level, and spectrum assignment in elastic optical networks”. In: *Journal of Optical Communications and Networking* 11.5 (2019), pp. 179–189.
- [11] Xiang Zhou, Wei Lu, Long Gong, and Zuqing Zhu. “Dynamic RMSA in elastic optical networks with an adaptive genetic algorithm”. In: *2012 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2012, pp. 2912–2917.
- [12] Xiaoliang Chen, Baojia Li, Roberto Proietti, Hongbo Lu, Zuqing Zhu, and SJ Ben Yoo. “DeepRMSA: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks”. In: *Journal of Lightwave Technology* 37.16 (2019), pp. 4155–4163.
- [13] Faezeh Yousefi and Akbar Ghaffarpour Rahbar. “Novel crosstalk, fragmentation-aware algorithms in space division multiplexed-Elastic Optical Networks (SDM-EON) with considering physical layer security”. In: *Optical Switching and Networking* 37 (2020), p. 100566.
- [14] Shrinivas Petale and Suresh Subramaniam. “An ML approach for crosstalk-aware modulation format selection in SDM-EONs”. In: *2022 International Conference on Optical Network Design and Modeling (ONDM)*. IEEE. 2022, pp. 1–6.
- [15] TeraFlow H2020 Project. *TeraFlow: AI-Driven Zero-Touch Network and Service Management*. 2020. URL: <https://www.teraflow-h2020.eu/>.
- [16] Eric A.M. Fagotto, Luiz H. Bonani, Sandro M. Rossi, Jorge D. Marconi, Marcelo L.F. Abbade, Ivan Aldaya, Uriel R.C. Miranda, and Marcos P. Belançon. “Impact of wideband wavelength conversion on the performance of optical networks”. In: *Optics & Laser Technology* 169 (2024), p. 109948. ISSN: 0030-3992. DOI: <https://doi.org/10.1016/j.optlastec.2023.109948>. URL: <https://www.sciencedirect.com/science/article/pii/S0030399223008411>.

- [17] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. “SNDlib 1.0–Survivable Network Design Library”. English. In: *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*. <http://sndlib.zib.de>, extended version accepted in Networks, 2009. 2007. URL: <https://opus4.kobv.de/opus4-zib/frontdoor/deliver/index/docId/958/file/ZR-07-15.pdf>.
- [18] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. “SNDlib 1.0–Survivable Network Design Library”. English. In: *Networks* 55.3 (2010), pp. 276–286. DOI: 10.1002/net.20371. URL: <http://www3.interscience.wiley.com/journal/122653325/abstract>.
- [19] Gianluca Davoli, Raffaele Di Tommaso, Gaetano Francesco Pittalà, Luiz H Bonani, and Carla Raffaelli. “Performance Evaluation of Cloud-native Wavelength Conversion in Disaggregated Optical Networks”. In: *2025 IEEE 26th International Conference on High Performance Switching and Routing (HPSR)*. IEEE. 2025, pp. 1–6.