



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

**MASTER'S DEGREE
IN
TELECOMMUNICATIONS ENGINEERING**

.....

**ALLOCATION AND FRAGMENTATION
POLICIES FOR ROUTING AND
SPECTRUM ASSIGNMENT IN ELASTIC
OPTICAL NETWORKS**

Master Thesis

in

Laboratory of Advanced Networking M

Supervisor

Prof. GIANLUCA DAVOLI

Defended by

VALENTINA ZHANG

Co-Supervisor

Prof. LUIZ H. BONANI

Graduation Session I / October / 2025

Academic Year 2024/2025



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

**MASTER'S DEGREE
IN
TELECOMMUNICATIONS ENGINEERING**

.....

**ALLOCATION AND FRAGMENTATION
POLICIES FOR ROUTING AND
SPECTRUM ASSIGNMENT IN ELASTIC
OPTICAL NETWORKS**

Master Thesis

in

Laboratory of Advanced Networking M

Supervisor

Prof. GIANLUCA DAVOLI

Defended by

VALENTINA ZHANG

Co-Supervisor

Prof. LUIZ H. BONANI

Graduation Session I / October / 2025

Academic Year 2024/2025

Contents

Abstract	vii
1 Introduction	1
2 Overview	3
2.1 Foundations and Limitations of Fixed-Grid WDM	3
2.2 The First Step Towards Agility: The ROADM Revolution	6
2.3 The Paradigm Shift: Architecting the Elastic Optical Network	8
2.3.1 Fixed vs. Flexible Grid Comparison	9
2.3.2 Sliceable Bandwidth-Variable Transponders and Superchan-	
nels	10
2.4 Lightpath Definition and Establishment	15
2.4.1 WDM lightpath	15
2.4.2 EON lightpath	17
3 Elastic Optical Network Simulator	21
3.1 Physical Layer Implementation	21
3.1.1 Node Class	22
3.1.2 Link Class	24
3.1.3 Spectrum Class	25
3.1.4 Topology Class	26
3.1.5 Connection Class	28
3.1.6 Route Class	29
3.1.7 Connection Setup and Release	30
3.2 Control Plane	31
3.2.1 Initialization and Routing Engine	31
3.2.2 Spectrum Management System	32

3.2.3	Spectrum Assignment Policies	37
3.3	Simulation Engine	63
3.4	Data Management and Simulator Structure	66
3.4.1	Plotting and visualization	67
3.4.2	Project Organization and Workflow	67
3.4.3	Simulation Workflow	69
4	Simulator Test	73
4.1	Simulation Setup	73
4.1.1	YAML configuration	73
4.1.2	Topology	76
4.2	First-Fit Allocation Policies Results	78
4.2.1	Experimental Setup	78
4.2.2	Blocking Probability Analysis	79
4.2.3	Validation Against Reference Implementation	83
4.3	Random-Fit Allocation Policies Results	84
4.3.1	Experimental Setup	84
4.3.2	Performance Comparison with First-Fit	84
4.3.3	Blocking Probability and Fragmentation Analysis	85
4.3.4	Cost-Benefit and Practical Implications	87
4.3.5	Practical Implications	87
4.4	Longest-Fit Allocation Policies Results	87
4.4.1	Experimental Configurations	88
4.4.2	Performance Analysis and Stability	88
4.4.3	Comparative Performance and Practical Recommendations	90
4.4.4	Practical recommendations	90
4.5	Validation on a Realistic Network Topology	91
4.5.1	Topology and Experimental Configuration	91
4.5.2	Key Findings and Correlation with Previous Results	93
4.5.3	Conclusion	96
5	Conclusion and future work	99

Abstract

The exponential growth of heterogeneous, bandwidth-intensive applications has revealed the limits of traditional fixed-grid Wavelength Division Multiplexing (WDM) networks and motivated a shift to flexible-grid Elastic Optical Networks (EONs). EONs offer much higher spectral efficiency but also create a challenging combinatorial problem: Routing and Spectrum Assignment (RSA). RSA must respect spectrum continuity and contiguity constraints while avoiding fragmentation in a dynamic traffic environment.

This thesis offers a comprehensive study of RSA strategies for EONs through a purpose-built discrete-event simulator. The simulator models dynamic lightpath setup and teardown under heterogeneous traffic conditions and implements key RSA policies — First-Fit, Random-Fit, and Longest-Fit — as well as important architectural options such as wavelength conversion and a software-defined break capability. Experiments were carried out on both toy and large-scale realistic topologies and validated against reference implementations.

Our results show that deterministic allocation policies generally outperform stochastic ones. First-Fit minimizes blocking probability, while Longest-Fit maximizes resource efficiency: it reduces wavelength converter usage by 18–24% while keeping blocking performance competitive. Enabling break capability proved to be a highly effective, low-cost software feature that significantly improves performance and reduces hardware requirements. By contrast, wavelength conversion yields only modest blocking reductions (2–6%) at substantial economic cost. Based on these findings, we recommend Longest-Fit with break capability for resource-conscious deployments and First-Fit where minimizing blocking is the priority. This work contributes a validated simulation tool and actionable insights for RSA design, laying the groundwork for future research into machine-learning driven, impairment-aware, and cost-aware optimization of elastic optical networks.

Chapter 1

Introduction

The relentless growth of bandwidth-intensive applications, such as cloud computing, ultra-high-definition video streaming, and real-time virtual reality, is pushing conventional optical transport networks to their fundamental limits. Traditional fixed-grid Wavelength Division Multiplexing (WDM) systems, which allocate spectrum in rigid 50 GHz or 100 GHz channels, increasingly reveal their limitations in efficiently accommodating dynamic and heterogeneous traffic patterns. This architectural constraint often leads to substantial underutilization of spectral resources, creating capacity bottlenecks even when theoretical spectrum availability remains.

Elastic Optical Networks (EONs) have emerged as a transformative solution to these challenges. By adopting a flexible grid architecture that divides the optical spectrum into narrow, closely spaced slots (typically 12.5 GHz or finer), EONs enable dynamic and efficient resource allocation tailored to specific service requirements. This paradigm shift from fixed to adaptive spectrum allocation fundamentally enhances network utilization while supporting diverse traffic characteristics. However, this flexibility introduces new complexities in the Routing and Spectrum Assignment (RSA) process, which must simultaneously satisfy three critical constraints: spectrum continuity (same slots along entire path), spectrum contiguity (adjacent slots within each link), and fragmentation management amid dynamic traffic patterns.

This thesis presents a comprehensive investigation of RSA strategies for EONs through the development and application of an advanced discrete-event simulation framework. Our research methodology combines theoretical analysis with extensive simulation experiments to evaluate the performance characteristics of various spectrum allocation policies under different network configurations and traffic con-

ditions.

To establish the necessary foundation for this work, Chapter 2 begins with a comprehensive overview of optical networking technologies, examining traditional WDM-based systems and their limitations before progressing to the architectural foundations of Elastic Optical Networks. This background sets the stage for understanding lightpath establishment processes and the RSA problem formulation that underpins our research.

Building upon this foundation, Chapter 3 details the design and implementation of our custom-developed EON simulator. This extensive chapter covers the physical layer implementation including node, link, spectrum, and connection classes; the control plane architecture featuring route computation and spectrum management systems; and the simulation engine core with its event processing and data management capabilities. The chapter provides thorough descriptions of implemented spectrum assignment policies and the complete simulation workflow.

Having established these theoretical and methodological foundations, Chapter 4 presents our experimental methodology and results analysis. Beginning with simulation setup details including configuration parameters and topology specifications, the chapter proceeds to systematic performance evaluation of three primary allocation policies: First-Fit, Random-Fit, and Longest-Fit. Each policy is analyzed under varying conditions of wavelength conversion and break capability. The chapter includes validation against reference implementations and concludes with testing on a realistic large-scale network topology to demonstrate the simulator's capabilities and confirm policy performance generalizations.

Finally, Chapter 5 synthesizes our findings and contributions, discussing the practical implications for network operators and identifying promising directions for future research. The conclusion affirms that intelligent RSA algorithm design, particularly through policies like Longest-Fit combined with features like break capability, can achieve significant performance and cost gains in elastic optical networks.

Through this structured investigation, this thesis aims to provide both theoretical insights and practical tools for the design and operation of efficient, high-capacity optical networks that can meet the escalating demands of modern digital applications.

Chapter 2

Overview

The digital universe is expanding at an incredible rate, driven by a relentless global demand for bandwidth. From the seamless streaming of 4K and 8K video to the vast, real-time data exchanges of cloud computing and the stringent low-latency requirements of 5G and emerging 6G networks, this growth exerts immense pressure on the global optical network infrastructure—the physical backbone of the internet. For years, the industry has relied on **wavelength division multiplexing (WDM)** technology, which multiplies capacity by transmitting multiple light signals (wavelengths) concurrently on a single fibre. However, the fundamental rigidity of its design has become its greatest liability in the face of dynamic, unpredictable traffic. This chapter will chronicle this evolution by first examining the limitations of fixed-grid WDM, then exploring the incremental improvement offered by ROADMs, and finally detailing the paradigm shift to Elastic Optical Networks (EONs), their enabling technologies, and the new research challenges they introduce.

2.1 Foundations and Limitations of Fixed-Grid WDM

Wavelength Division Multiplexing (WDM) technology represents the conventional approach to optical networking, enabling substantial transmission capacity through simultaneous transmission of multiple optical carrier signals over a single fiber. WDM systems operate according to the ITU-T G.694.1 standard, which specifies fixed channel spacings of 50 GHz or 100 GHz within the conventional band (C-band) ranging from 1,530 nm to 1,565 nm, preferred for its lower attenuation characteristics during data transmission [1, 2].

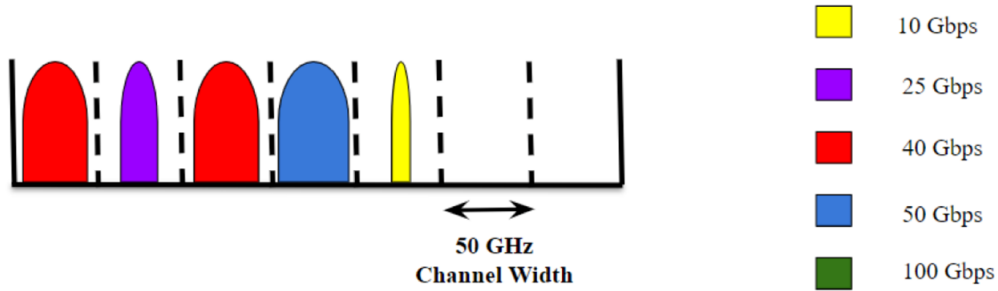


Figure 2.1: 50 GHz Fixed Grid Optical Network supporting mixed line-rate services [3].

An end-to-end optical connection in this system is referred to as a lightpath—a dedicated, unbroken circuit from a source to a destination node, traversing a physical route of fiber links and optical switches. The establishment and operation of these lightpaths rely on a set of core components that form the foundation of all optical networks [4, 2] (Figure 2.2). Furthermore, this process is governed by a set of constraints, which become even more complex and critical in the evolution to agile networks; these will be explored in detail in Section 2.4.

- **Transponders:** The terminal endpoints. These devices perform the crucial Optical-Electrical-Optical(O-E-O) conversion, transforming electrical data from client devices into precise optical signals on a specific ITU-T wavelength at the transmitter, and reversing the process at the receiver.
- **Multiplexers/Demultiplexers (Mux/Demux):** The on-ramps and off-ramps of the optical highway. The multiplexer combines individual wavelengths from multiple transponders onto a single fiber for transmission. At the destination, the demultiplexer separates the composite signal back into its constituent wavelengths, directing each to its intended receiver.
- **Optical Add-Drop Multiplexers (OADMs):** The interchanges. A Fixed OADM (FOADM) allows a specific, pre-defined wavelength to be dropped (received) or added (transmitted) at an intermediate network node, while all other wavelengths express through unchanged. This enables complex ring and mesh topologies but is a static, manually configured hardware solution.
- **Erbium-Doped Fiber Amplifiers (EDFAs):** The signal boosters. These optical amplifiers counteract signal attenuation (loss) in the fiber, boosting the optical power without the need for costly and complex O-E-O conversion,

thus enabling transmission over transcontinental distances.

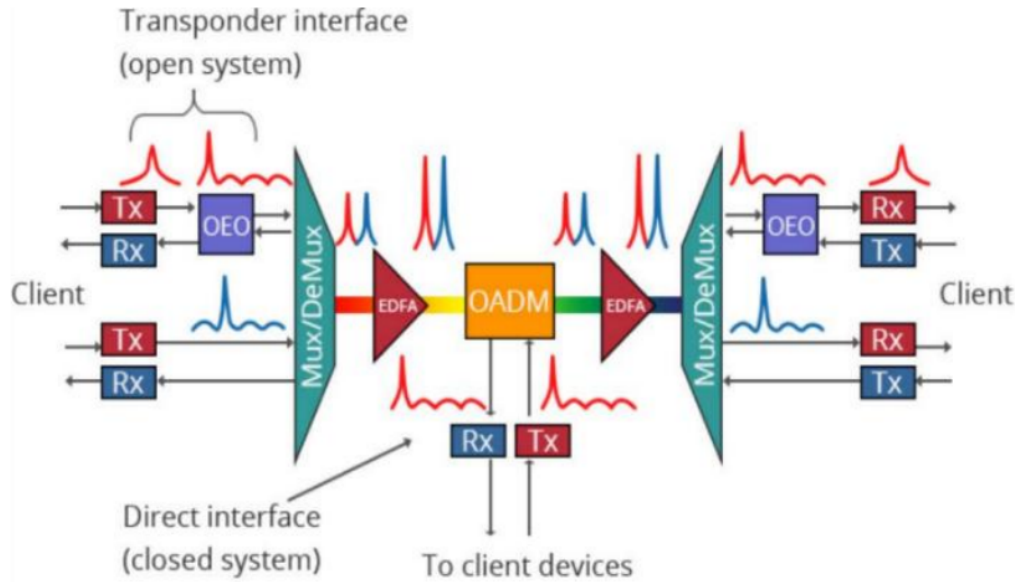


Figure 2.2: Multiplexing and demultiplexing of optical signals [2]

This entire architecture is governed by a critical constraint: the **wavelength continuity constraint** [5]. This mandates that the *exact same wavelength* must be available and reserved on every single fiber link along the entire route of the lightpath. This fundamental rigidity creates three profound inefficiencies:

1. **Spectrum Waste** (The "Empty Truck" Problem): A low-bit-rate service (e.g., 10 Gb/s [2.1]) occupies an entire 50 GHz channel, leaving most of its capacity unused. Conversely, a high-bit-rate service (e.g., 400 Gb/s) must awkwardly bond multiple 50 GHz channels, a complex process that leaves unused "guard bands" between them and consumes extra transponders.
2. **Operational Inflexibility**: The network cannot adapt to the specific bandwidth needs of a connection. A lane is a lane, whether you need to send a car or a convoy of trucks. Provisioning new services requires manual, time-consuming hardware reconfiguration.
3. **Higher Blocking Probability**: Even if the total unused capacity in the network is sufficient, a connection request is blocked if a single common wavelength isn't free on all links of the path. This is known as wavelength blocking.

The network was static, inefficient, and operationally costly, proving increasingly ill-prepared for the unpredictable, "bursty" traffic patterns of the modern internet.

These fundamental inefficiencies spurred the development of a more agile network node, the first step toward a flexible network: the Reconfigurable Optical Add-Drop Multiplexer (ROADM).

2.2 The First Step Towards Agility: The ROADM Revolution

The initial response to this rigidity was a crucial evolutionary step: the Reconfigurable Optical Add-Drop Multiplexer (ROADM). The key innovation was the Wavelength Selective Switch (WSS). This component allowed network operators to remotely and dynamically configure—via a software controller—which wavelengths were added, dropped, or expressed through at any network node [3, 6]

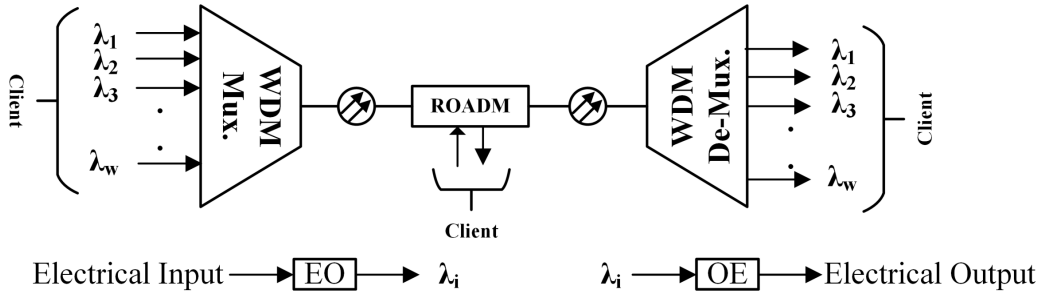


Figure 2.3: Wavelength Division Multiplexing (WDM)-based transmission employs a multiplexer (MUX), a reconfigurable optical add-drop multiplexer (ROADM), a demultiplexer (DEMUX), W wavelengths $\{\lambda_1, \lambda_2, \dots, \lambda_w\}$, electrical-to-optical (E/O) converters, and optical-to-electrical (O/E) converters at both ends of the link. [3]

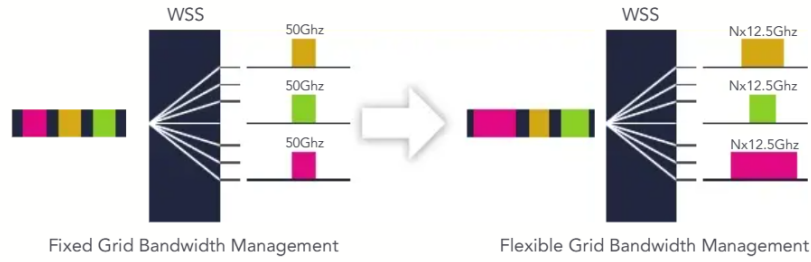


Figure 2.4: Evolution of ROADM with Flexible Grid [6]

This represented a monumental leap in operational flexibility. ROADMs introduced critical features:

- **Colorlessness:** Any transponder can connect to any ROADM port to transmit or receive any wavelength, breaking the fixed relationship between a physical port and a specific wavelength.
- **Directionlessness:** Any wavelength can be routed to any physical output port of the node, providing unprecedented flexibility in network design and traffic engineering.

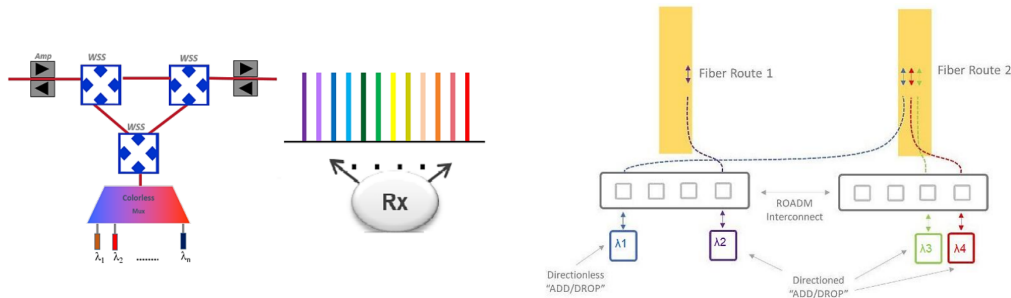


Figure 2.5: Reconfigurable Optical Add-Drop Multiplexer (ROADM) critical features [2]: (a) Colorless add/drop and (b) Directionless add/drop.

This innovation drastically reduced the manual intervention and operational expenditure associated with FOADMs and enabled robust, survivable, and easily reconfigurable mesh networks. However, this was a partial solution. These first-generation ROADMs were still built for the **fixed grid**. They could switch entire 50 GHz lanes with newfound agility, but they could not change the width of the lanes themselves.

The underlying problem of **spectral inefficiency** remained unsolved. The highway was now smartly managed, but the lanes were still rigid. This necessitated a more radical, paradigm-shifting solution: a move from a fixed-wavelength to a flexible-spectrum architecture, giving rise to Elastic Optical Networks.

2.3 The Paradigm Shift: Architecting the Elastic Optical Network

The true revolution was a fundamental shift in thinking: from managing wavelengths to managing spectrum. Elastic Optical Networks (EONs) abandoned the fixed grid for a flexible grid (also defined in ITU-T G.694.1), slicing the C-band spectrum into fine, granular slots of 12.5 GHz [7]. The central frequency of a slot is defined as

$$f_n = 193.1 \text{ THz} + n \times 0.00625 \text{ THz}, \quad n \in \mathbb{Z}, \quad (2.1)$$

where n is an integer index identifying the slot position [7] (see Figure 2.7). A connection now requests not a fixed wavelength, but a variable amount of capacity:

$$W = 12.5 \text{ GHz} \times m, \quad (2.2)$$

where m indicates the number of contiguous slots allocated to the connection as a positive integer [7]. This is the concept of **”right-sizing”** the spectrum, allocating precisely the amount of resource needed for the traffic demand, chosen modulation format, and transmission distance. This new paradigm necessitated a complete overhaul of the network’s components and underlying transmission techniques. A key enabling technology is Orthogonal Frequency Division Multiplexing (OFDM) applied in the optical domain (O-OFDM). Proposed by M. Jinno et al. [7], O-OFDM replaces the simple intensity modulation of traditional DWDM. It works by splitting a high-speed data stream into many orthogonal, lower-speed subcarriers that are closely spaced and transmitted in parallel. This orthogonality prevents interference between subcarriers, allowing them to be packed tightly together in the frequency domain. O-OFDM is the fundamental technique that makes the fine-grained 12.5 GHz granularity and the flexible aggregation of slots into superchannels not just a concept but a practical reality. (see Figure 2.6).

2.3. THE PARADIGM SHIFT: ARCHITECTING THE ELASTIC OPTICAL NETWORK9

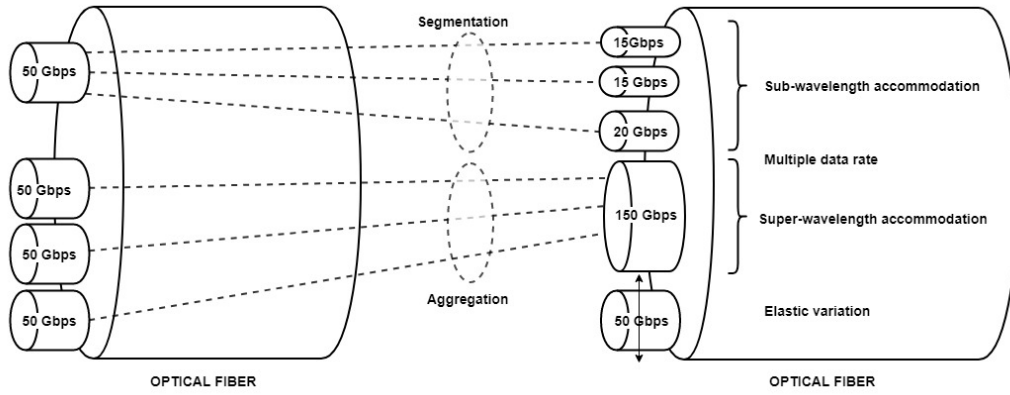


Figure 2.6: Characteristics of Elastic Optical Networks: Bandwidth Segmentation, Bandwidth Aggregation, Sub-wavelength and Super-wavelength Accommodation, and Energy saving (adapted from [3])

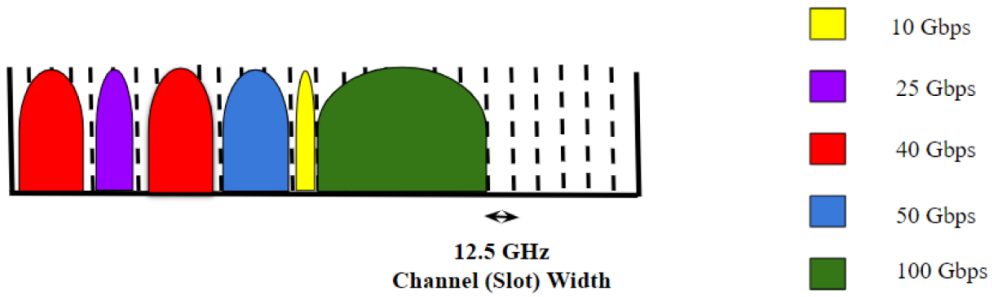


Figure 2.7: 12.5 GHz flexible grid in Elastic Optical Networks supporting mixed line rate services (from [3]).

2.3.1 Fixed vs. Flexible Grid Comparison

Figure 2.8 shows the key differences between fixed-grid WDM and flexible-grid EON spectrum allocation. Fixed grids use uniform, rigid spectral slots with set spacing. This often results in underused spectrum because of coarse granularity and the need for wide guard bands. In contrast, flexible grids enable the dynamic allocation of variable-sized, contiguous spectrum slices that closely match traffic bandwidth needs. This flexibility greatly improves spectral efficiency and network

scalability [8].

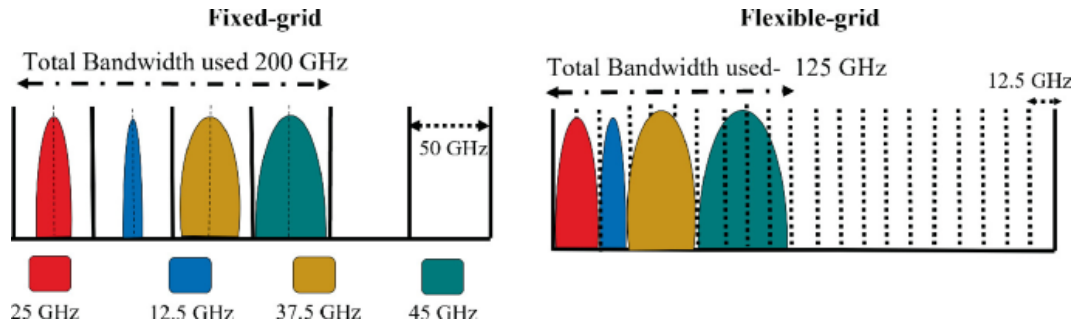


Figure 2.8: Comparison of spectrum allocation in Fixed-grid WDM vs. Flexible-grid EON architectures (adapted from [8]).

2.3.2 Sliceable Bandwidth-Variable Transponders and Super-channels

This new paradigm necessitated a complete overhaul of the network's components:

- **Sliceable Bandwidth Variable Transponders (SBVTs):** The intelligent end-points of the EON. Unlike fixed transponders, SBVTs are multiflow transponders that can generate multiple independent optical flows (subcarriers), dynamically tune the modulation format (e.g., from robust PM-BPSK for long distances to spectrally efficient PM-16QAM for short ones), adapt the forward error correction (FEC) overhead based on link conditions, and slice their total capacity, directing different subcarriers to different destinations simultaneously. They are the cornerstone of elasticity, making the transponder a software-defined resource. [9]
- **Flexible-Grid ROADMs / BV-WXCs:** The new intelligent interchanges. These nodes employ **Flexible-Grid Wavelength Selective Switches (WSS)**. They are no longer limited to switching fixed 50 GHz chunks; they can dynamically route any contiguous block of spectrum (e.g., 37.5 GHz, 62.5 GHz, 87.5 GHz). These are also called **Bandwidth Variable Wavelength Cross-Connects (BV-WXCs)** and are essential for supporting the variable-sized channels of EONs [10].

2.3. THE PARADIGM SHIFT: ARCHITECTING THE ELASTIC OPTICAL NETWORK

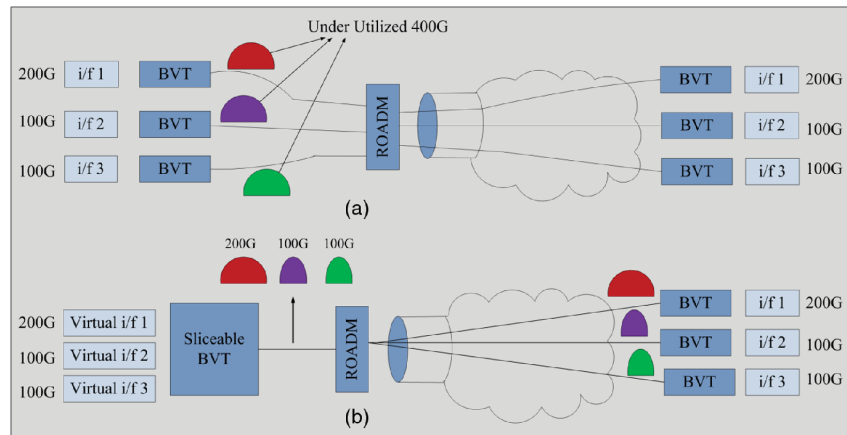


Figure 2.9: Conceptual difference between (a) a conventional Bandwidth Variable Transponder (BVT), which dedicates its entire capacity to a single source-destination pair, and (b) a Sliceable BVT (SBVT), which can partition its capacity into multiple, independent 'slices' or sub-flows that can be routed to different destinations simultaneously. This sliceability offers unparalleled flexibility and efficiency. [9]

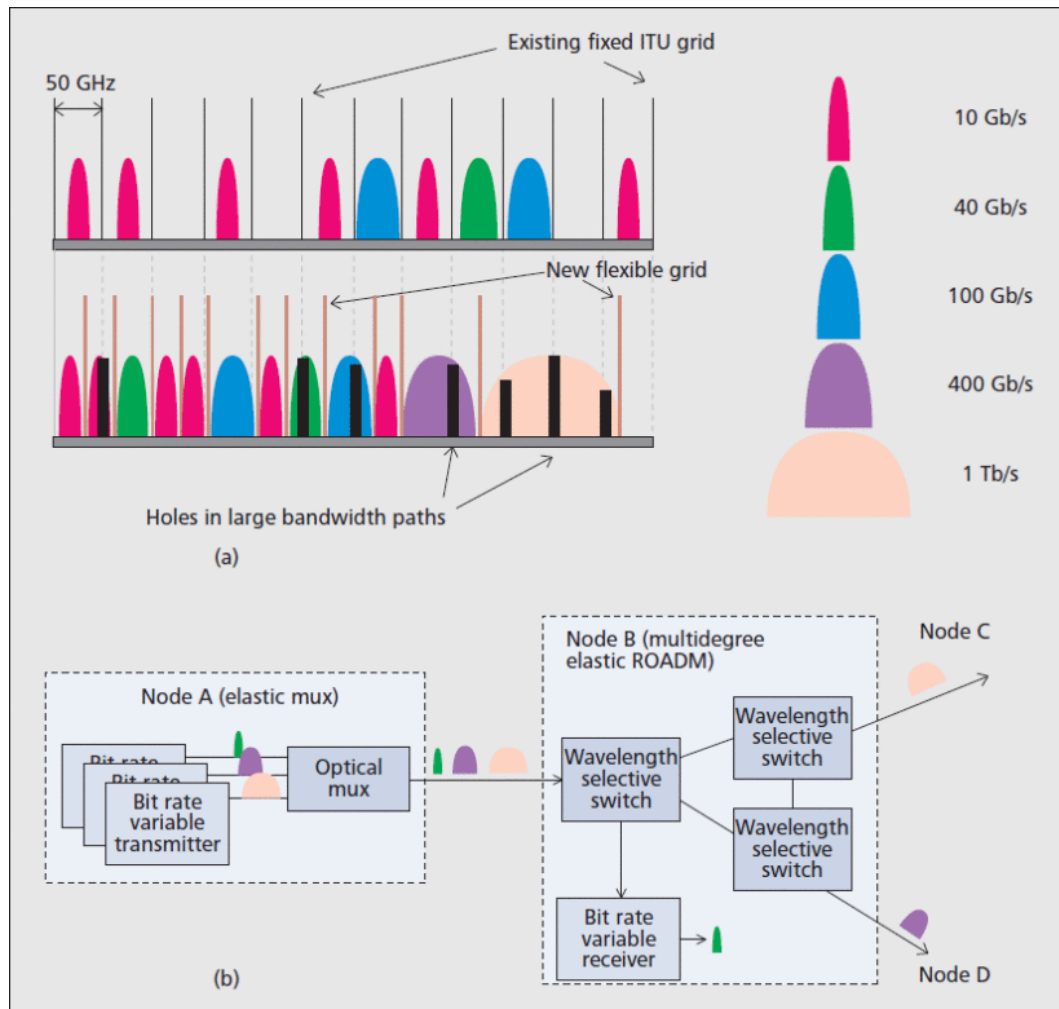


Figure 2.10: (b) Operation of a flexible-grid ROADM, demonstrating its ability to switch variable-bandwidth superchannels (e.g., from A to C or D) that would be blocked by the fixed-grid boundaries of a traditional ROADM (a). (adapted from [10])

The integration of these components forms a cohesive EON architecture, as illustrated in Figure 2.11. This architecture demonstrates how BVTs or SBVTs at the edge interface with the flexible-grid core, where BV-WXCs dynamically route variable-sized spectrum slices to establish end-to-end lightpaths.

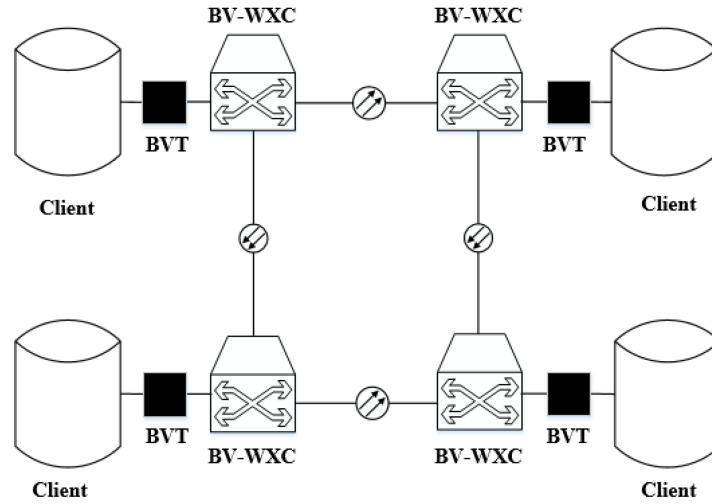


Figure 2.11: Architecture of an Elastic Optical Network (EON) node. The diagram shows Bandwidth Variable Transponders (BVTs) or Sliceable BVTs (SBVTs) at the network edge, responsible for generating and receiving optical signals. These connect to Bandwidth Variable Wavelength Cross-Connects (BV-WXCs) in the network core, which flexibly switch variable-sized spectrum slices between different fiber links to establish connections. (adapted from [3])

These components enable the most efficient transport unit in an EON: the **superchannel**. A superchannel is a high-capacity optical signal formed by densely aggregating multiple subcarriers (e.g., from an SBVT) that are generated, routed, and received as a single entity. By sharing guard bands only at the edges of the entire superchannel and not between internal subcarriers, they achieve remarkable spectral efficiency. A telling example from the literature shows that a 1.2 Tb/s signal requires approximately 600 GHz of spectrum in a fixed-grid DWDM system but only about 462.5 GHz when implemented as a superchannel, representing a 23% reduction in spectral footprint [9]. (see Figure 2.12).

The practical implementation of superchannels is commercially viable only through Photonic Integrated Circuit (PIC) technology. As shown in Figure 2.13, a PIC integrates dozens of discrete optical components (lasers, modulators,

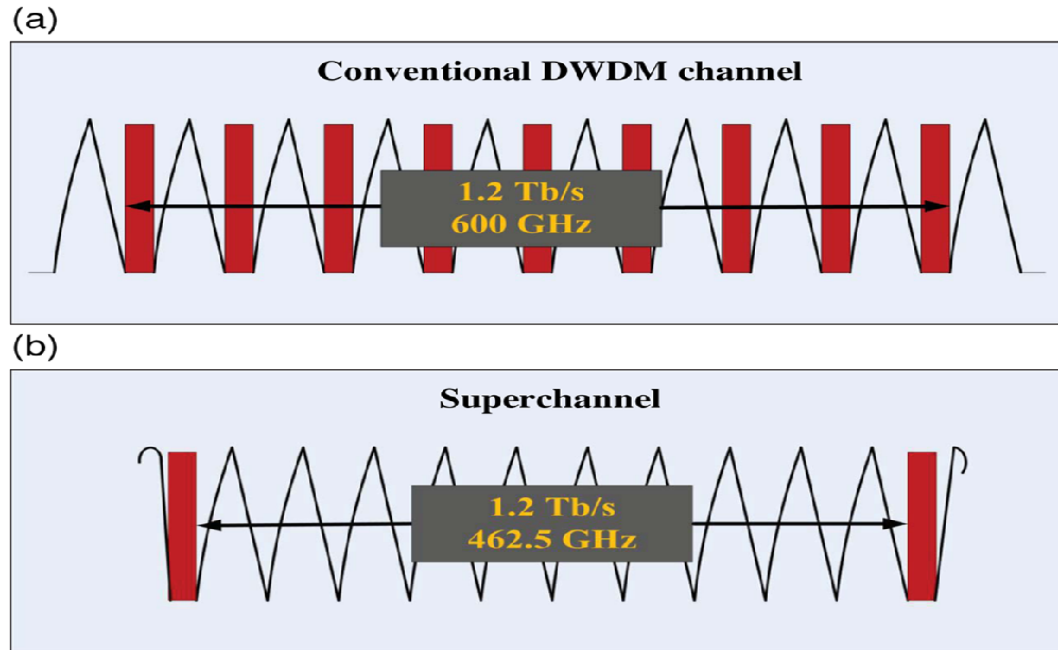


Figure 2.12: (a) Conventional channel and (b) superchannel. (adapted from [9])

multiplexers) onto a single chip, which is the only practical way to realize the complex superchannel transmitters required for EONs. PICs drastically reduce the size, power consumption, cost, and hardware complexity by an order of magnitude compared to discrete component implementations. Beyond 100 Gb/s, PIC technology is considered mandatory for feasible superchannel deployment.

Analysis of Efficiency Gains: The quantitative superiority of EONs is demonstrated in Figure 2.14, which models the efficiency gains over a point-to-point link. While EONs consistently deliver efficiency improvements, the magnitude of the benefit is highly dependent on the specific traffic scenario. For example, a 100 Gb/s signal using DP-16QAM can fit efficiently into a single 50 GHz grid, resulting in a modest gain for EON. This scenario represents the best-case scenario for fixed-grid systems, minimizing their inherent inefficiency.

However, the most significant advantages of EON become starkly evident for rates that are not a clean multiple of the fixed grid, such as 400 Gb/s or 1 Tb/s. Here, as the Figure 2.14 shows for 1 Tb/s DP-QPSK, gains can be dramatic—exceeding 150%—because fixed-grid DWDM requires inverse multiplexing—splitting the traffic into multiple 50 GHz channels (e.g., four channels for 400G DP-QPSK).

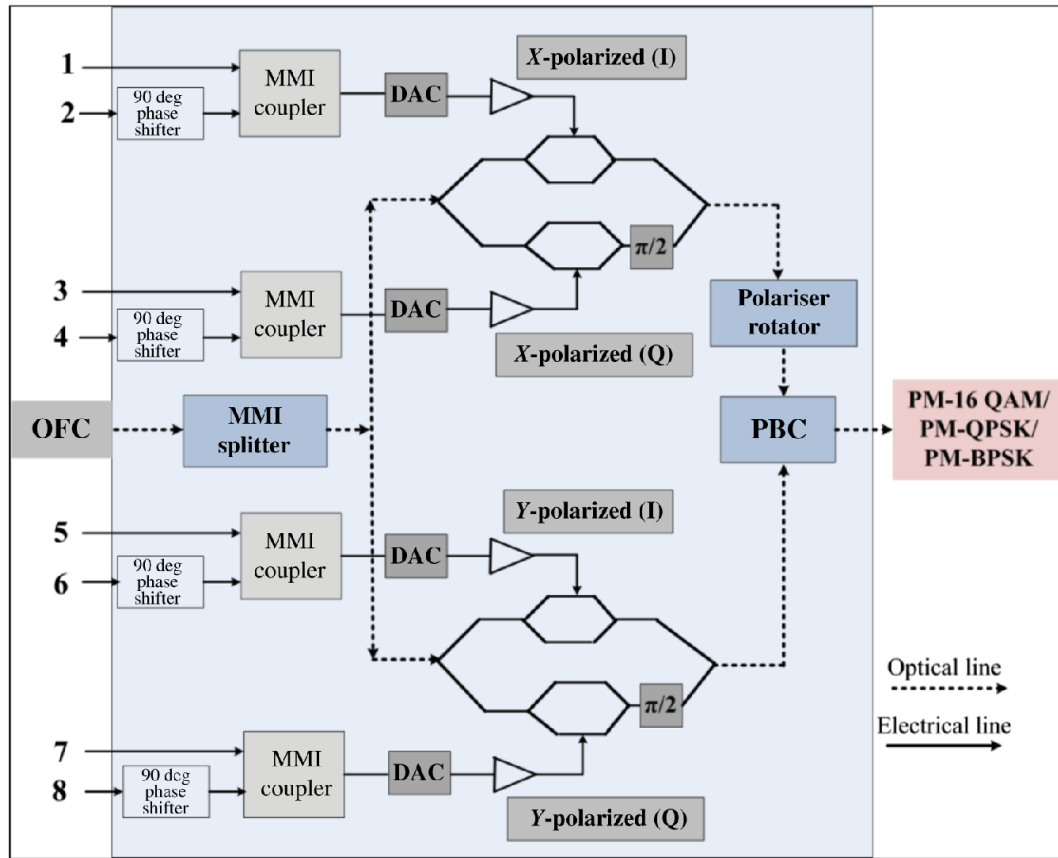


Figure 2.13: Conceptual diagram of a Photonic Integrated Circuit (PIC), showing the integration of multiple optical components onto a single chip.

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 2.14: Efficiency improvement for flexible spectrum over a point-to-point link, assuming a 50 GHz grid for fixed DWDM and 10 GHz channel guard band and superchannels for EONs.

This approach has two critical drawbacks: it consumes more transponders and, more importantly, it introduces a much higher blocking probability in a network-wide context. A connection request is blocked if all required inverse-multiplexed channels are not available on every link of the path. This dramatically amplifies the spectral inefficiency.

Thus, when modeled over an entire network—rather than a single point-to-point link—the efficiency benefits of EON are significantly amplified, often exceeding 50% due to this reduced blocking. EON avoids inverse multiplexing by allocating a single, optimally-sized spectrum block, drastically increasing the likelihood of finding a available path. Therefore, from both a pure spectral efficiency and a network utilization perspective, there is a compelling advantage to deploying EONs, assuming the traffic demand is sufficient and diverse enough to merit the advanced infrastructure.

2.4 Lightpath Definition and Establishment

As defined in Section 2.1, a **lightpath** is an end-to-end optical connection that allows data to be transmitted entirely in the optical domain, avoiding costly optical-electrical-optical (O-E-O) conversions at intermediate nodes [4]. While this core definition applies to both Wavelength Division Multiplexed (WDM) and Elastic Optical Networks (EONs), the process of establishing a lightpath differs fundamentally due to their distinct resource allocation constraints.

These constraints arise from the physical and technological limitations of optical transmission and switching and are the primary factors determining whether a connection request can be successfully fulfilled. In both paradigms, establishing a lightpath involves more than just finding a physical route; it requires allocating a specific spectral resource—a fixed wavelength in WDM or a contiguous set of frequency slots in EONs—that must be available along the entire path according to a strict set of rules.

2.4.1 WDM lightpath

In traditional fixed-grid WDM networks, two main constraints must be met.

Routing constraint The first of which is the **Routing constraint**. This constraint considers finding a viable physical route through the network topology which connects the source node to the destination node through a set of optical fiber links. Path selection typically performed using algorithms like Dijkstra's shortest path algorithm that often focus on minimizing metrics based on hop count (i.e., link count)

or total physical distance. While some algorithms may be concerned with additional physical pathing constraints such as traffic load balancing of network links, survivability of the path in case of network failure (utilizing physical path diversity), and signal quality constraints such as Optical Signal-to-Noise Ratio (OSNR) and the accumulation of Polarization Mode Dispersion (PMD). These circumstances are more suited for constraint-based routing algorithms or k-shortest path algorithms to achieve the pathing objectives.

Wavelength continuity constraint The second constraint is **Wavelength continuity constraint**. This constraint states that the specific wavelength must be available on each of the lightpath's fiber links along the path. For example, during the lightpath, if it entered the network on $\lambda_1 = 193.1$ THz, it must remain on λ_1 at every hop along the lightpath. This constraint is due to the way some network elements are designed like Reconfigurable Optical Add-Drop Multiplexers (ROADMs) and transparent Optical Cross-Connects (OXC). These types of network elements operate in the optical domain and usually do not support wavelength conversion. In fixed-grid WDM systems, ROADMs can add, drop or pass specific wavelength channels, but they do not have the functionality to shift signals from one frequency slot to another. Transparent OXCs also switch wavelength channels from one fiber to another fiber within the same wavelength channel and maintain the wavelength at its original optical frequency. Thus, when a lightpath enters the network utilizing a wavelength slot ($\lambda_1 = 193.1$ THz), it must exit the same λ_1 at each intermediate node. ROADMs and OXCs are unable to convert it to another available slot ($\lambda_2 = 193.2$ THz), regardless of available slots.

The inability to convert between wavelength slots introduces the problem of wavelength blocking. A lightpath request may be rejected despite sufficient capacity on individual links, simply because no single wavelength is free on all of them. For example, suppose link 1 has λ_1 available but not λ_2 , while link 2 has λ_2 available but not λ_1 . In such a case, the request cannot be accommodated, even though both links have capacity. This limitation can be overcome by introducing wavelength converters at intermediate nodes. These devices allow a signal to be shifted from one wavelength to another, thus relaxing the continuity constraint. If in every node of the network, a wavelength converter is placed, the network behaves similarly to a circuit-switched system, where blocking is determined only by capacity availability rather than wavelength alignment. The presence of wavelength converters significantly reduces blocking probability and improves spectral utilization [11].

2.4.2 EON lightpath

The shift from fixed-grid Wavelength Division Multiplexing (WDM) to flexible-grid Elastic Optical Networks (EONs) brings new opportunities and constraints for lightpath setup. While keeping the main routing constraint from WDM networks, EONs introduce additional spectrum-related constraints that change the resource allocation issue.

Spectrum Contiguity Constraint The **spectrum contiguity constraint** expresses the requirement that the frequency slots assigned to a lightpath must be contiguous (adjacent) on each fiber link. For example, a 37.5 GHz lightpath (3×12.5 GHz slots) might occupy slots [21, 22, 23], which corresponds to the frequency range 193.2625–193.3000 THz. Contiguity is important for signal integrity in all-OFDM-based transmission systems, where the orthogonality of subcarriers depends on very precise alignment related to the spectral locations. Without it, spectral leakage and interference degrade the signal [7]. In contrast, WDM networks do not require contiguity, since each wavelength (e.g., $\lambda_1 = 193.1$ THz, $\lambda_2 = 193.2$ THz) is treated independently and carries a separate fixed-bandwidth signal.

Spectrum Continuity Constraint This second constraint expands the wavelength continuity principle of WDM by requiring that the same contiguous block of frequency slots be available on all links of the lightpath. As shown in Figure 2.16, Chatterjee et al. present a lightpath request needing two adjacent slots from node A to node E. The shortest path A–C–E fails because even though each link has two free contiguous slots, they are not aligned across links, violating the continuity constraint. The request can be satisfied through the alternate path A–C–D–E, where slots [5, 6] are available and aligned on all links, thus meeting both continuity and contiguity constraints.

Together with Figure 2.16, Chatterjee et al. [12] further illustrate how fragmentation leads to call blocking in Figure 2.15. Consider a lightpath request that needs two slots, as shown in Figure 2.15(a). Even though two free slots are available throughout the network, the request fails because those slots are either not synchronized across links (Figure 2.15(b)) or not next to each other on individual links (Figure 2.15(c)).

These examples show two main types of fragmentation:

- *Non-aligned fragmentation:* Required slots are available on each link but do not match along the path, breaking continuity.
- *Non-contiguous fragmentation:* Free slots on individual links are scattered,

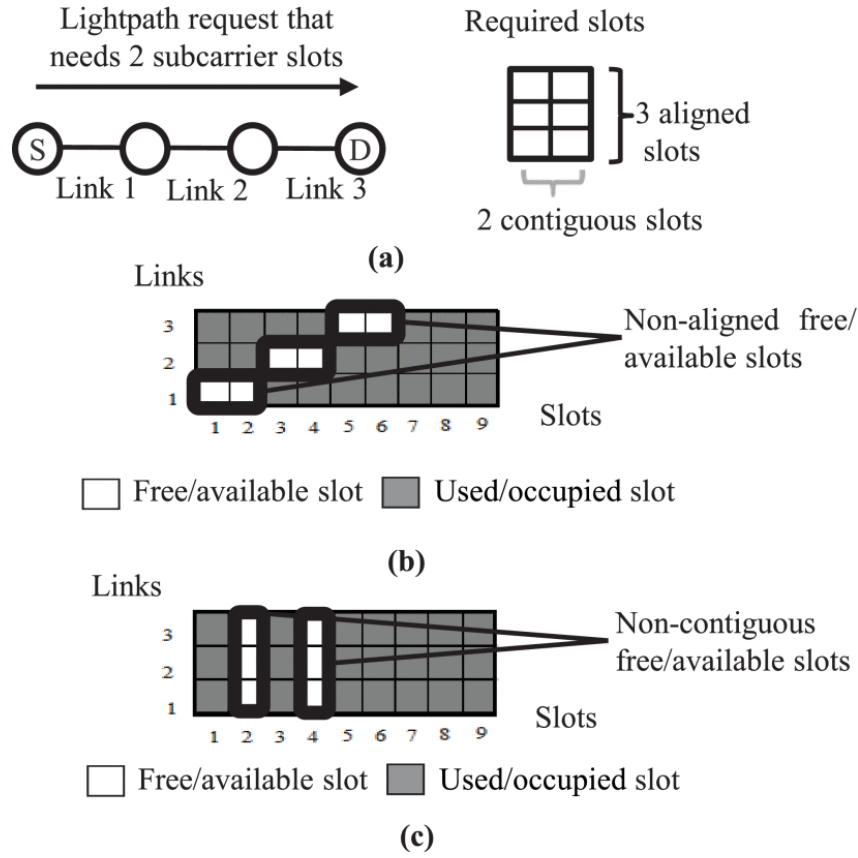


Figure 2.15: (a) Slot requirement for spectrum allocation, (b) fragmentation due to non-aligned free slots, and (c) fragmentation due to non-contiguous free slots [12].

disrupting contiguity.

Both types of fragmentation raise the blocking probability by decreasing the usable spectrum, even when there seems to be enough capacity available.

When **spectrum converters** are used at intermediate nodes, the spectrum continuity constraint can be relaxed. This allows a lightpath to use different contiguous slot blocks on each link. For example, a lightpath might use slot 21 on link 1, slot 22 on link 2, and slot 23 on link 3. The criterion for spectrum continuity would likely still apply because of the nature of coherent superchannel and O-OFDM transponders. Because of this, spectrum fragmentation is still a major problem with EON control-plane design. [7]

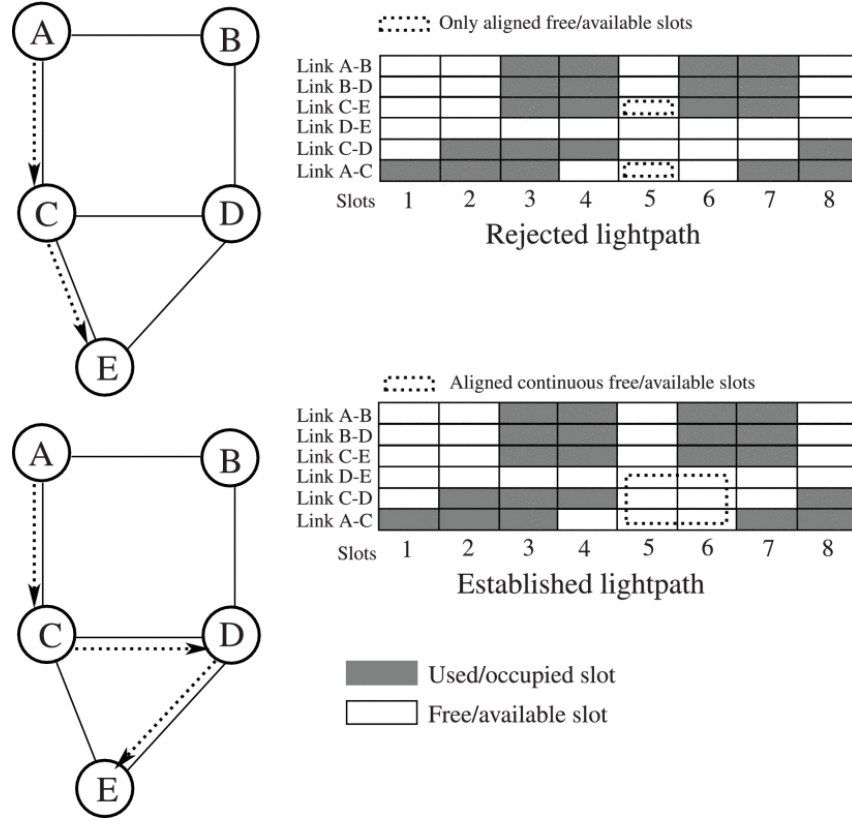


Figure 2.16: Concept of continuity and contiguity constraints in Elastic Optical Networks [12].

Non-overlapping constraint This **non-overlapping criterion** also removes spectral interference, by ensuring that no two lightpaths can use the same slot on the same fiber. If a lightpath uses slots [21, 22, 23], it cannot then have another lightpath make use of [22, 23, 24] on the same link. This ensures some integrity with signals, whether it be EON or WDM, while in WDM it is wavelengths rather than slot ranges being used. [10].

Guard bands **Guard bands** are also needed to separate adjacent lightpaths to prevent interference relative to either filter roll-off or spectral leakage. For example, if one lightpath is occupying slots [21, 22, 23], the second lightpath must begin in slot 25 or later; therefore, slot 24 is reserved as a guard band. The width of the guard band is based on the transponder and modulation format. While WDM systems had

fixed grid spacing (e.g., 50 or 100 GHz), wherein guard bands were already placed, EONs must reserve an explicit guard band, thus sacrificing spectral efficiency to maintain the quality of the signal [7, 3].

In the next chapter, a modular Python simulator for Elastic Optical Networks (EONs) will be described. **The flexibility of EONs, while solving the problems of WDM, introduces the complex and NP-hard Routing and Spectrum Allocation (RSA) problem. Managing spectrum fragmentation becomes the paramount challenge for efficient network operation.** This simulator, implemented as a discrete-event framework, supports flexible spectral allocation and configurable traffic scenarios, enabling systematic comparison of allocation strategies under varied network conditions. **This enables the systematic comparison of allocation strategies that is critical for unlocking the full potential of EONs.**

Chapter 3

Elastic Optical Network Simulator

This chapter introduces a modular Python-based simulator for Elastic Optical Networks (EONs), which is a translation and extension of the original Java-based simulator OfNetSim, developed by Prof. *Luiz Henrique Bonani (Federal University of ABC, Brazil)*. This tool is designed to analyze the effects of fragmentation and evaluate various spectrum assignment strategies. The full implementation is available in the project's GitHub repository at <https://github.com/giditre/WCaaS> (currently private, to be made publicly available upon publication). The simulator is highly configurable and supports the import of realistic network topologies from the SNDlib library. To enhance efficiency, it precomputes shortest paths between edge nodes using Dijkstra's algorithm with sparse matrix optimization. The simulation engine then dynamically manages connection request generation, spectrum allocation, and release events.

Our evaluation looks at spectrum allocation policies in various scenarios: with and without wavelength converters, and with or without the hypothetical `can_break` relaxation. We implement First-Fit, Random-Fit, and Longest-Fit policies and analyze their performance in terms of blocking probability, fragmentation, and spectrum utilization. By simulating realistic traffic patterns and service demands, we gain valuable insights into the trade-offs involved in achieving flexibility while minimizing fragmentation in EONs.

3.1 Physical Layer Implementation

The simulator is implemented in object-oriented Python. The physical layer models network elements and the connection lifecycle using compact, single-responsibility classes. The main EON entities are:

- **Node** – represents a reconfigurable optical switch with optional wavelength conversion capability;
- **Link** – represents a unidirectional fiber span that tracks slot-level spectrum availability;
- **Spectrum** – represents contiguous frequency-slot blocks allocated to light-paths;
- **Connection** – represents the lifecycle manager for a lightpath (setup, hold, teardown);
- **Route** – represents a path (sequence of links) together with allocation helpers;
- **Topology** – represents the container of all nodes and links and handles path computation and visualization.

Each class plays a specific role in simulating the network’s behavior. In the following subsections, each class will be explained in detail, describing its attributes, methods, and how it interacts with other components in the simulator.

3.1.1 Node Class

The **Node** class models a network node (an optical switch) that maintains input and output ports and handles wavelength-conversion requests. The complete implementation can be found in the project’s GitHub repository.

Each **Node** corresponds to one graph vertex and encapsulates the local state and capabilities needed by the control plane. In particular, a node records the costs of incoming links, instantiates **Link** objects for its outgoing edges, tracks available wavelength converters and their usage, and provides simple APIs to allocate and release converters when connections require wavelength shifts.

Main attributes The node stores its adjacency-related and operational data in a few core attributes.

1. The **input_ports** array (a NumPy integer array) represents the cost of the incoming link from each neighbour (a zero value indicates no incoming link);
2. The **output_ports** array (a NumPy object array) represents the **Link** instances for directed outgoing edges; absent outgoing links are represented explicitly by **None** to avoid uninitialized entries;
3. The **node_id** represents the node identity;
4. The **debug** flag represents a boolean option that enables optional runtime tracing;
5. The **converters** attribute represents the converter resources, stored as a **defaultdict(int)** keyed by pairs (**shift**, **converter_id**), where each value counts the currently active conversions handled by that converter instance;

6. The **max_conversions** represents the maximum concurrent conversions per converter;
7. The **placement_cost** represents the CAPEX for installing a new converter;
8. The **usage_cost** represents the OPEX per converter use, controlling converter behaviour and accounting.

Initialization The constructor receives the network adjacency (topology) matrix, the local **node_id**, and the number of spectrum slots for outgoing links. During initialization the node iterates over all vertices i : if **topology[node_id, i] != 0.0** a **Link** object is created for the outgoing edge and stored in **output_ports[i]**; if **topology[i, node_id] != 0.0** the incoming link cost is recorded in **input_ports[i]**. In this way each node maintains immediate knowledge of neighbouring links and their costs while delegating per-link spectrum state to the corresponding **Link** objects.

Converter Management Converter instances are tracked per frequency **shift** and per instance identifier. The **converters** dictionary stores, for each key (**shift, converter_id**), the number of active conversions currently handled by that instance.

When a conversion is requested for a given **shift**, existing converters with that shift are scanned. If any instance has free capacity (i.e., its active count is below **max_conversions**), its counter is incremented and only the **usage_cost** is charged. If all existing instances for the shift are saturated, a new converter instance is created (its active count initialized to one), and both the placement and usage costs are charged.

This design keeps converter bookkeeping simple while clearly separating placement CAPEX from per-use OPEX.

The node exposes a small, clear API used by the control plane.

The enable_debug(enable=True): Toggles runtime tracing. When enabled, users will see some debug lines about the output of converter usage.

The use_converter(shift, debug header): It tries to allocate a wavelength converter for the requested frequency shift and returns a (**placement, usage**) cost pair: *usage* (OPEX) is always charged, while *placement* (CAPEX) is non-zero only when a new converter must be instantiated. It scans existing converters keyed by (**shift, id**) and, if it finds one with spare capacity, increments its active counter, optionally logs the event, and **returns (0, usage)** to indicate reuse. If no existing converter has free slots, it creates a new converter id (using the current count of converters for that shift), initializes its counter to one, sets **placement = placement_cost**, logs the placement if debugging is enabled, and **returns (placement, usage)**.

The `release_one_converter(shift, debug_header)`: It is the method used to free resources, so it decrements the active counter of the first matching converter instance.

The `reset_all_converters(shift, debug_header)`: It is the method used for the final cleanup, which zeros the active counts for all converters of that shift.

3.1.2 Link Class

The **Link** class models a unidirectional optical fiber span between two nodes, providing the fundamental infrastructure for lightpath connections. This class is responsible for storing both the physical characteristics of the fiber link and maintaining the dynamic state of spectrum availability, providing the low-level data structure essential for routing and spectrum assignment algorithms. The complete implementation is available in the project's GitHub repository.

Main attributes The class encapsulates the following core attributes:

1. **link_cost**: Floating-point value representing the metric used for path computation (typically physical distance or administrative cost);
2. **input_node**: Integer identifier of the source node;
3. **output_node**: Integer identifier of the destination node;
4. **spectrum_status**: NumPy array of type `uint8` and length **num_slots**, where each element represents the availability of a frequency slot (**1** = free, **0** = occupied)

Initialization During initialization, the constructor receives three essential parameters: the numerical link cost, the source and destination node identifiers, and the total number of spectrum slots available on the fiber. The initialization process involves assigning the provided cost value to **link_cost**, storing the node identifiers in their respective attributes, and creating the spectrum status array initialized to all ones, representing a completely available spectrum upon creation.

Spectrum management As mentioned before, spectrum resources on each link are represented by a NumPy array named **spectrum_status**, where each element corresponds to a frequency slot. A slot value of **1** indicates that the slot is free, while **0** indicates that it is currently occupied by a connection. This binary vector allows efficient bitwise operations to identify available contiguous slot blocks, essential for spectrum assignment algorithms like First-Fit or Longest-Fit.

3.1.3 Spectrum Class

The **Spectrum** class serves as a fundamental abstraction for representing contiguous blocks of frequency slots allocated to optical lightpaths within the network. This class encapsulates the essential properties of spectrum segments that are candidates for allocation or have been assigned to specific connections, playing a critical role in both spectrum assignment algorithms and resource management operations. The complete implementation details are available in the project's GitHub repository.

Main attributes The class maintains three core attributes that define the characteristics of a spectrum block.

1. **size**: The number of consecutive frequency slots contained within the block, represented as an integer value;
2. **position**: The starting index of the block within the spectrum grid, following a zero-based indexing convention that aligns with the **Link.spectrum_status** array representation;
3. **consecutiveness**: An integer which is equal to **size - 1** and it represents the number of slots that follow the initial starting position, effectively defining the block's continuity span.

Initialization During initialization, the constructor accepts the **size** and **position** parameters and automatically computes the **consecutiveness** value. The implementation includes validation to ensure that the size parameter meets the requirement of being greater than or equal to one $\text{size} \geq 1$, maintaining the integrity of spectrum block representations throughout the system. This design supports efficient management of spectrum blocks during allocation and release operations by providing a standardized representation that can be easily manipulated and evaluated.

The Spectrum class facilitates the verification of contiguity constraints that are essential for end-to-end spectrum assignment in wavelength-routed networks. By encapsulating the mathematical relationship between size and position, the class enables efficient computation of block boundaries and overlap detection. Furthermore, it serves as a fundamental data structure for representing candidate spectrum segments during routing and resource assignment procedures, providing algorithms with a consistent interface for evaluating spectrum availability and making allocation decisions. The implementation's simplicity and focus on computational efficiency make it particularly suitable for the intensive processing requirements of dynamic spectrum allocation in elastic optical networks.

3.1.4 Topology Class

The **Topology** class serves as the foundational component for managing the complete network graph within the simulation environment. This class handles the loading, representation, and visualization of network topologies, providing the structural framework upon which all routing and spectrum allocation operations are built. The complete implementation is available in the project's GitHub repository.

Main Attributes The class maintains several key attributes that collectively define the network structure:

1. **yaml_file**: The path to the topology configuration file.
2. **num_nodes**: The total number of nodes determined during the loading process.
3. **topology**: The central data structure, an $N \times N$ NumPy array of floating-point values representing link costs, where zero values indicate the absence of a connection between nodes.
4. **nodes**: The ordered list of node identifiers.
5. **links**: The collection of endpoint pairs defining physical fiber connections, e.g., [(src, dst), ...].
6. **node_positions**: A dictionary mapping node names to their respective (x, y) coordinate tuples.

Initialization Upon instantiation, the class initializes core attributes from the provided YAML file path: **yaml_file** stores the configuration path, while **num_nodes** is determined post-loading.

The topology representation includes an $N \times N$ NumPy adjacency matrix **topology** storing inter-node link costs (typically physical distances), a list **nodes** containing node identifiers, and a list **links** of endpoint pairs defining fiber connections. Nodal spatial positions are captured in **node_positions** as a dictionary mapping node names to 2D coordinate tuples. These attributes collectively define the network graph structure and enable distance-based routing computations.

Loading Topology from YAML The **load_topology()** method implements the complete pipeline for parsing YAML configuration files and constructing the network graph. The method expects YAML files conforming to a specific schema that defines nodes with their spatial coordinates and links between node pairs.

```

nodes:
  node1: [x1, y1]
  node2: [x2, y2]
links:
  link1: [node1, node2]
  ...

```

Figure 3.1: Topology yaml file configuration nodes and links.

The simulator parses a YAML topology file, which can represent a topology get from SNDlib [13] (for example, see Fig. 3.2), to extract nodes, their positions, and link definitions. A mapping of node names to matrix indices is created, and **topology** is initialized as a zero-valued **float** matrix. For each link definition, the method calculates the Euclidean distance between connected nodes using the formula:

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This distance metric is assigned symmetrically to both directions in the adjacency matrix, effectively modeling an undirected graph with consistent link costs in both transmission directions.



Figure 3.2: SNDlib — USA topology [13] This is a U.S. backbone-style network topology, with major city POPs (Points of Presence).

Visualizing the Topology The class provides the `show_topology()` method for generating visual representations of the network structure. This implementation utilizes NetworkX for graph manipulation and Matplotlib configured for headless rendering environments. The method plots nodes at their specified coordinates, draws edges between connected nodes, and annotates these edges with their corresponding distance values.

The visualization functionality serves multiple purposes: it enables rapid validation of topology configurations during development, facilitates debugging of

network structures, and provides intuitive understanding of network connectivity patterns. The generated diagrams are automatically saved to temporary storage and opened using the system's default image viewer, ensuring compatibility across different operating environments.

The **Topology** class thus provides a comprehensive solution for managing network infrastructure within the simulation, offering both computational efficiency for pathfinding algorithms and intuitive visualization capabilities for analysis and verification purposes.

3.1.5 Connection Class

The **Connection** class encapsulates the complete state of an active lightpath established through the Elastic Optical Network, serving as the central entity for managing connection lifecycle from establishment to teardown. This class aggregates all necessary information to maintain a lightpath during its operational lifetime and enables efficient scheduling within the simulation's event management system. The complete implementation is available in the project's GitHub repository.

Main attributes The class maintains several critical attributes that collectively define the connection's characteristics and resource requirements.

1. **time_stamp (float)**: The release time of the connection (used for scheduling release time);
2. **route (List[int])**: The ordered list of node indices that define the path traversed by the lightpath through the network infrastructure;
3. **spectrum (Spectrum)**: The allocated spectrum slice or block ensuring proper spectrum resource management;
4. **service_type (int)**: The service or traffic class identifier.
5. **id_conn (int)**: The unique connection identifier.
6. **total_cost (float)**: The cost metric for the connection (distance + converter costs).
7. **used_converters (list)**: This records converters used along the path during the connection's lifetime to ensure proper resource release. Defaults to an empty list if not provided.

Initialization During initialization, the constructor accepts mandatory parameters including release time, route specification, spectrum allocation, service type classification, and connection identifier. The implementation incorporates optional parameters for **total cost** and **converter usage tracking**, employing defensive programming practices to avoid mutable default arguments through explicit None checks

and conditional list initialization.

Comparison method The class implements the `__lt__` special method to enable comparison based on release timestamps, making `Connection` objects compatible with priority queue structures such as Python’s `heapq` module. This ordering capability allows the simulation engine to efficiently manage connection lifetimes by processing teardown events in chronological order. When a connection reaches its scheduled release time, the simulation engine automatically triggers the release of all associated spectrum resources and wavelength converters, effectively modeling the dynamic resource allocation patterns characteristic of real-world optical networks. This comprehensive implementation ensures that each connection maintains complete state information necessary for accurate simulation of optical network behavior, including resource utilization tracking, cost accounting, and temporal management within the discrete-event simulation framework.

3.1.6 Route Class

The **Route** class serves as a fundamental data structure for representing predetermined network paths within the optical network simulation. This class encapsulates complete information about a specific path through the network infrastructure, including its nodal composition, cumulative cost metric, and unique identification. **Route** object are instantiated during the network initialization phase and subsequently utilized by both the control plane for path computation and the simulation engine for connection establishment procedures. The complete implementation is available in the project’s GitHub repository.

Main attributes The class maintains three essential attributes that comprehensively define a network path.

1. **route (List[int]):** The ordered list of node IDs defining the path from source to destination node;
2. **route_cost (float):** The total cost of the route, accumulated along the path (for example, sum of individual link distances or other relevant cost parameters);
3. **route_id (int):** The unique identifier that distinguishes each route instance within the system, enabling efficient reference and management of multiple precomputed paths.

Initialization During initialization, the constructor receives the complete node sequence, total path cost, and unique identifier as parameters. These values are stored within the object’s internal state without modification, preserving the integrity of the path information throughout the simulation lifecycle. The initialization pro-

cess ensures that each Route object maintains immutable path characteristics once instantiated, providing consistent behavior for all components that utilize route information.

Public methods The class exposes a single public method, `getRoute()`, which returns the complete ordered list of node identifiers defining the path. This method provides controlled access to the path information while maintaining encapsulation of the internal data representation. **The simulation engine's ignition system and control plane algorithms** utilize this method to retrieve path specifications when establishing connections, ensuring that all network components operate with consistent path information. The minimal interface design reflects the class's specialized purpose as a data carrier rather than an active component, optimizing both performance and reliability within the simulation framework.

3.1.7 Connection Setup and Release

The connection lifecycle management within the EON simulator follows a structured process that mirrors real-world optical network operations. When a connection request arrives, the control plane (as detailed in Section 3.2) initiates a multi-stage provisioning procedure. First, it selects an appropriate **Route** object (as detailed in Section 3.1.6) that satisfies the requested source-destination pairing and quality parameters. The route selection considers factors such as path length, existing congestion, and available resources.

Following route selection, the spectrum allocation phase starts, where the control plane performs a comprehensive search for contiguous spectrum slots along all **Link** objects comprising the selected route. This process employs specific spectrum assignment algorithms (First-Fit, Random-Fit, or Longest-Fit, as detailed in Section 3.2) to identify available frequency blocks that meet the bandwidth requirements while adhering to continuity and contiguity constraints. The allocation mechanism also accounts for wavelength conversion capabilities when enabled, allowing different spectral segments to be used on different links along the path.

Upon successful spectrum allocation, the system instantiates a **Connection** object (as described in Section 3.1.5) that encapsulates the complete connection state. This object incorporates the selected route, assigned **Spectrum** blocks, service type classification, calculated total cost, and precisely computed release time based on the holding time distribution. The connection is immediately inserted into a priority queue structure, ordered chronologically by release time, enabling efficient event scheduling and management.

The connection teardown process activates automatically when the system time

reaches the connection's scheduled release time. The simulator extracts the connection from the priority queue and systematically releases all allocated resources. This includes freeing the previously occupied spectrum slots on every traversed link in the path, effectively updating the spectrum status arrays to mark these slots as available for future allocations. Additionally, the implementation performs precise resource accounting by releasing any wavelength converters utilized along the path, ensuring that converter resources on the corresponding **Node** instances are properly returned to the available pool.

This comprehensive setup and release mechanism accurately models the dynamic resource management characteristics of elastic optical networks, demonstrating how network resources are allocated, utilized, and reclaimed based on traffic demand patterns. The process emphasizes the critical relationship between route selection, spectrum allocation, and connection lifecycle management, providing a realistic simulation of optical network behavior under varying traffic conditions and resource constraints.

3.2 Control Plane

The Control Plane serves as the core intelligence system of the network simulator, orchestrating both routing decisions and the allocation of spectral resources. It determines the optimal paths for connections and manages the efficient assignment and release of optical spectrum throughout the simulation lifecycle.

3.2.1 Initialization and Routing Engine

During initialization, the Control Plane executes several optimization-oriented preparation steps. The physical network topology is first converted into a compressed sparse row (CSR) matrix representation, a structure that significantly reduces computational overhead during path calculations. This representation is created once and reused throughout the simulation, avoiding the repeated conversion cost that would otherwise occur in every shortest-path computation.

The system precomputes all valid edge-to-edge routes using Dijkstra's algorithm. This is specifically limited to edge node pairs, based on the fundamental network assumption that traffic originates and terminates at network edges. This deliberate restriction avoids unnecessary processing of the intermediate core nodes. During this phase, the Control Plane also collects essential network statistics such as the average, minimum, and maximum hop counts directly during route discovery. This eliminates the need for a separate post-processing step, accelerating the initialization process.

The initialization stage also establishes probabilistic models for connection generation. The probabilities of the source-destination pairs are derived from the set of valid edge-to-edge connections and stored in cumulative form to enable constant-time random sampling. In parallel, traffic class probabilities are prepared, ensuring that subsequent connection requests follow realistic service-type distributions. This precomputation strategy delivers substantial performance benefits. By retrieving routes directly from the precomputed matrix rather than recalculating them on demand, the Control Plane can process connection requests with minimal delay. Restricting path discovery to edge nodes further reduces complexity, while the sparse-matrix optimization ensures that Dijkstra's algorithm executes efficiently, even for large and sparsely connected optical topologies.

Once initialization is complete, the **routing engine** operates in real time. For each incoming connection request, it randomly selects a source and destination node pair according to the established probability distribution and retrieves the corresponding shortest path from the precomputed route set. The path is returned as an ordered sequence of nodes encapsulated within an immutable **Route** object, which also records the total path cost. This separation of routing logic from later spectrum assignment processes preserves modularity and allows the simulation to scale without introducing computational bottlenecks.

3.2.2 Spectrum Management System

The Control Plane implements a comprehensive spectrum management framework that handles the allocation, release, and querying of spectral resources along network routes. This system provides the foundation for establishing and maintaining lightpaths while ensuring efficient spectrum utilization.

Core Operations

Spectrum Allocation The `assign_spectrum` method marks spectrum slots as occupied along each link of a specified route for multiple spectrum blocks. It requires an array of network nodes, each containing port and spectrum information, a list of node indices that define the path of the lightpath, and a list of spectrum objects, each representing a continuous spectrum block to be assigned on a link. For each spectrum block, the method iterates over the route links and marks the corresponding slots in the link's spectrum status as occupied (value 0). The indexing ensures that each spectrum block corresponds to its respective link in the route. This per-link assignment supports scenarios where wavelength conversion is allowed, enabling different spectrum blocks per link. Then, it returns the updated nodes array with

modified spectrum slot statuses reflecting the assigned spectrum.

Spectrum Release The `return_spectrum` method reverses the assignment by marking previously occupied spectrum slots as free. As input, it needs the network nodes, the path where the spectrum was assigned, and a list of *Spectrum* objects to release. Similar to assignment, it maps each spectrum block to the corresponding link on the route and sets the slots from occupied (0) back to free (1). Finally, it returns the nodes array with the specified spectrum slots freed, making them available for future connections.

Spectrum Availability Assessment

End-to-End Availability Checking The `get_available_boolean_spectrum` method, given as input the **network nodes** and the **route** over which availability is checked, computes the intersection of available spectrum slots across all links of a route, determining which slots are free end-to-end. Starting from the first link's spectrum status, the method performs a logical AND operation with subsequent link spectra. This intersection ensures that only slots free along the entire route are marked as available (1). It returns a binary list where each element indicates if the corresponding slot is available (1) or occupied (0) end-to-end.

Per-Link Spectrum Analysis

get_per_link_spectrum_matrix: The `get_per_link_spectrum_matrix` method is used to find a binary matrix representing the availability of spectrum slots per individual link along the route, so as input, it requires the network node and the path. Each row of the matrix corresponds to a link in the route; each column corresponds to a spectrum slot. Values are 1 if the slot is free on that link, or 0 otherwise. As output, it returns a 2D NumPy array (*num_links x num_slots*) showing per-link spectrum availability, useful for allocation algorithms that consider per-link constraints or wavelength conversion.

Contiguous Block Identification The `create_vector_spectrum` method converts a binary spectrum vector into a list of continuous free spectrum blocks. It needs as input a binary vector `spectrum_status` representing spectrum availability (usually the output of `get_available_boolean_spectrum`). It scans the list to identify continuous runs of free slots and creates corresponding *Spectrum* objects representing their size and starting position, then returns a list of *Spectrum* blocks available for allocation along the route.

Advanced Block Extraction with Numba Optimization The `extract_available_blocks_from_matrix` method is designed to *extract all valid spectrum blocks of*

a specified required bandwidth from the per-link spectrum matrix. This extraction is a crucial step in spectrum assignment algorithms that incorporate wavelength conversion, as these algorithms must consider the availability of contiguous free spectrum blocks on each link individually.

Given the per-link spectrum matrix and a required bandwidth, the method identifies all contiguous blocks of the required size starting at every possible slot index on each link. For each such block, it then determines how many consecutive links along the route also have this block available. The output of the method is a **dictionary** that maps tuples of the form (link_index, start_slot) to tuples containing the required bandwidth and the number of consecutive links covered. This mapping encodes both the position of the block and the extent of its availability along the route.

Two versions of this method have been developed. The first is a *pure Python* implementation that operates straightforwardly. It receives as inputs a two-dimensional binary array where each row corresponds to a link and each column corresponds to a spectrum slot; in this array, a value of 1 indicates a free slot while 0 indicates an occupied slot. Additionally, it takes the required bandwidth, which is the number of contiguous slots needed to accommodate a connection. The goal of the method is to find every possible starting slot index on each link where a contiguous block of free slots of the required size exists, and to determine how many consecutive links maintain this availability starting from that link.

The algorithm works by sliding a window of length equal to the required bandwidth across the spectrum slots for each link. For every candidate start slot, the method checks whether the corresponding block of slots is free on each link, resulting in a Boolean array that indicates availability per link. This Boolean array is then analyzed to detect consecutive runs of links where the block is continuously free. For each such run, the method records entries in the output dictionary that specify the starting link index, the starting slot index, the bandwidth, and the length of the coverage in terms of consecutive links. The method's ability to record partial coverage, where the requested block is only available on a subset of the consecutive links rather than the entire path, is important for flexible spectrum assignment policies such as partial allocation and wavelength conversion.

The second version is a *high-performance implementation that leverages the Just-In-Time (JIT) compilation capabilities of the Numba library*. By decorating the function with Numba's `@njit` decorator, the code is compiled into optimized machine code, which significantly improves runtime efficiency, especially for large-scale networks or scenarios that require frequent real-time computations. This version enhances performance by detecting continuous runs of free slots on each link

and sliding the fixed-length window only over these valid runs rather than the entire spectrum. It further improves efficiency by rapidly evaluating the coverage of each candidate block across consecutive links using compiled loops and by employing typed dictionaries to speed up data storage and retrieval. This Numba-accelerated implementation is particularly valuable in the control plane of elastic optical networks, where swift and repeated spectrum assignment decisions are critical for maintaining network performance under dynamic traffic conditions.

Implementation Example

Consider a network route consisting of three sequential links, each with a spectrum divided into eight slots. The availability of each slot on each link is represented by the following binary matrix, where a value of 1 indicates a free slot and 0 indicates an occupied slot:

$$\mathbf{S} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Here, each row corresponds to a link in the route, while each column represents a spectrum slot index from 0 to 7.

Use of `get_per_link_spectrum_matrix`: The matrix $\mathbf{S}$ is typically obtained using the method `get_per_link_spectrum_matrix`, which extracts the availability of spectrum slots for each link individually along the route. This per-link information is essential when wavelength conversion is permitted or when spectrum allocation decisions are made with consideration for link-specific constraints.

Use of `get_available_boolean_spectrum`: To assess which spectrum slots are free across the entire route simultaneously (i.e., end-to-end availability), the method `get_available_boolean_spectrum` computes the logical AND of all rows in $\mathbf{S}$. This intersection produces a one-dimensional binary vector $\mathbf{v}$ indicating slots that are free on every link along the route. Applying this operation to $\mathbf{S}$ yields:

$$\begin{aligned} \mathbf{v} &= (1 \wedge 1 \wedge 1, \quad 1 \wedge 1 \wedge 1, \quad 1 \wedge 1 \wedge 1, \quad 1 \wedge 1 \wedge 1, \\ &\quad 1 \wedge 1 \wedge 1, \quad 0 \wedge 1 \wedge 1, \quad 1 \wedge 1 \wedge 0, \quad 1 \wedge 0 \wedge 0) \\ &= (1, 1, 1, 1, 1, 0, 0, 0) \end{aligned}$$

This shows that only slots 0 through 4 are free on all three links simultaneously, matching the coverage analysis where blocks starting within these slots may have coverage 3.

Use of `create_vector_spectrum`: The vector $\mathbf{v}$ is then processed by `create_vector_spectrum`, which identifies continuous runs of free slots within the vector. These continuous blocks represent spectrum segments that can be allocated without interruption across the entire route. In this example, because free slots appear in continuous groups, the method detects multiple multi-slot free blocks rather than isolated single-slot blocks, as detailed in Table 3.1.

Use of `extract_available_blocks_from_matrix`: While the previous methods focus on end-to-end availability, scenarios that allow wavelength conversion or partial allocation require detailed per-link block availability. The method `extract_available_blocks_from_matrix` addresses this by sliding a window of a specified required bandwidth over each link's spectrum row in $\mathbf{S}$. For example, if a connection requires a block of three contiguous free slots, this method identifies all such blocks on each link individually and then determines how many consecutive links also have the block free starting from that link.

For the matrix $\mathbf{S}$ and a required bandwidth of 3, the method finds the following blocks (Table 3.1): The coverage indicates how many consecutive links, starting

(Link, Start Slot)	(Bandwidth, Coverage)
(0, 0)	(3, 3)
(0, 1)	(3, 3)
(0, 2)	(3, 3)
(0, 3)	(3, 2)
(1, 1)	(3, 2)
(1, 2)	(3, 2)
(1, 3)	(3, 1)
(2, 0)	(3, 1)

Table 3.1: Extracted 3-slot spectrum blocks with their coverage across links

from the given link, have the specified free block available. For example, the block starting at slot 0 on link 0 has coverage 3, meaning it is free on all three links. The block at slot 3 on link 0 has coverage 2, indicating it is free on link 0 and link 1 but not on link 2. Similarly, blocks with coverage 1 are only free on a single link. This detailed coverage information is crucial when wavelength conversion or partial spectrum allocation is allowed.

Together, these methods provide a layered approach to spectrum availability assessment, supporting both conservative (no conversion) and flexible (with con-

version) spectrum allocation strategies. The `get_per_link_spectrum_matrix` method offers detailed per-link slot availability, which serves as input to both `get_available_boolean_spectrum` and `extract_available_blocks_from_matrix`. The former aggregates availability into an end-to-end view, enabling the `create_vector_spectrum` method to extract continuous free blocks for whole-route allocation without wavelength conversion. The latter method, `extract_available_blocks_from_matrix`, supports more flexible allocation strategies by examining blocks on each link and their coverage, which is crucial when wavelength conversion or partial allocation is allowed.

3.2.3 Spectrum Assignment Policies

Spectrum assignment policies are essential in elastic optical networks as they determine how spectral resources are allocated to connection requests. These policies define the method for selecting contiguous free spectrum blocks along a route, influencing network performance, blocking rates, and resource efficiency. The choice of policy impacts the complexity of spectrum assignment algorithms and their ability to support features like *wavelength conversion* and *partial spectrum allocation (break capability)*.

Policy Families

The spectrum allocation policies developed in this work are organized into three main families, each representing a different trade-off between computational efficiency and allocation effectiveness.

Base Family The **Base** family represents the simplest and fastest category. These policies operate either on a per-link basis (with wavelength conversion) or by relying on an aggregated end-to-end view of the spectrum (without conversion). Decisions are made using straightforward heuristics such as first-fit or random-fit, which select the first or a random suitable block from the available options. The key advantage of this family lies in its very low computational overhead, making it highly scalable and practical for real-time use in large networks. However, this simplicity comes at the cost of increased spectrum fragmentation. Because allocations are made greedily and without considering long-term continuity, the spectrum tends to fill unevenly, leaving behind small unusable gaps and potentially increasing blocking probability under higher loads.

Best-Effort (BE) Family The **Best-Effort (BE)** family represents an intermediate level of complexity. Unlike the Base family, BE policies adopt a broader perspective by analyzing candidate spectrum blocks across the entire path before committing to

an allocation. This path-wide enumeration enables smarter choices, such as prioritizing blocks that extend over multiple consecutive links to preserve continuity. In this way, Best-Effort policies attempt to reduce fragmentation and improve acceptance ratios, while still avoiding the computational explosion of full backtracking. They therefore provide a balanced trade-off: lower blocking probability and better spectrum utilization than Base methods, with only moderate increases in runtime. These properties make the BE family well-suited for dynamic, online provisioning scenarios where both efficiency and performance are important.

Stubborn (ST) Family The **Stubborn (ST)** family is the most advanced and computationally intensive. These policies introduce recursive backtracking and retry mechanisms, enabling an exhaustive exploration of the allocation space. Rather than committing to the first viable option, Stubborn policies evaluate multiple alternatives, undoing earlier allocations if they prevent the successful completion of the path. This tree-search approach guarantees completeness: if a feasible allocation exists within the attempt limit, the algorithm will eventually find it. The trade-off, however, is a significant increase in computational cost, with runtimes that can grow exponentially in the worst case. As such, Stubborn policies are best suited to offline planning, design studies, or scenarios where maximizing acceptance and minimizing blocking probability are critical objectives, and where computational resources and execution time are less constrained.

Taken together, these three families offer a progression of strategies: from the lightweight and highly scalable Base approaches, through the balanced and path-aware Best-Effort methods, up to the exhaustive and robust Stubborn algorithms. This layered categorization allows network operators or researchers to select the policy family that best aligns with their application scenario, balancing the trade-off between runtime performance and allocation optimality.

Operational Constraints and Features

Spectrum assignment policies operate within a well-defined set of physical constraints and can leverage specific network capabilities that significantly influence their operation and performance.

The fundamental constraints governing all spectrum allocation decisions are **contiguity** and **continuity**. The contiguity constraint requires that spectrum slots allocated to a single lightpath on any given fiber must be adjacent in frequency, preventing fragmented allocations within a single link. The continuity constraint states that, unless wavelength conversion capabilities are available, the same contiguous block must be reserved across all links of the route, ensuring the optical signal

can traverse the entire path without electronic processing. These constraints ensure the physical viability of optical transmissions but significantly restrict allocation flexibility.

Wavelength conversion **Wavelength conversion** is a crucial network capability that relaxes the continuity constraint. When enabled (through the Boolean `conversion` parameter), intermediate nodes equipped with wavelength converters can shift optical signals to different spectral segments across links. This allows a connection to use different spectral blocks on different links of its path, dramatically increasing allocation flexibility and reducing blocking probability. The implementation cost of this capability includes both capital expenditure for converter hardware and operational expenditure for their utilization, creating an important economic trade-off against performance improvements.

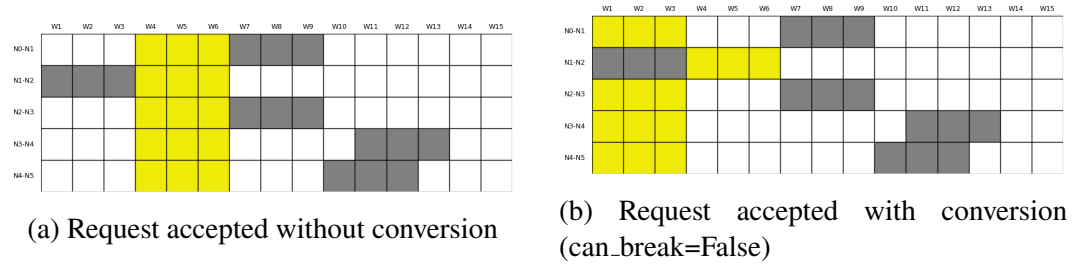


Figure 3.3: Wavelength conversion

Figures 3.3b and 3.3a illustrate the effect of wavelength conversion on spectrum assignment. Without wavelength conversion the entire lightpath must occupy the same frequency slots on every link; accordingly, the allocator reserves the contiguous block starting at slot 3 (i.e., slots 3–5) across the whole path. When wavelength conversion is allowed, each link can use different slot positions to satisfy the same connection. For example, in Figure 3.3b link 0 selects block (0,3), link 1 selects (3,3), and links 2–4 select (0,3). This per-link flexibility increases the probability of accepting the current request but may also produce more fragmentation on individual links.

Break Capability The **Break capability** (controlled by the Boolean `can_break` parameter) determines whether allocation algorithms can partition large free spectral regions during assignment operations. When disabled, algorithms preserve large contiguous free blocks by avoiding fragmentation, even if this requires searching for alternative spectral segments. When enabled, algorithms may split large free regions

to maintain continuity with previous allocations, increasing immediate connection acceptance rates at the potential cost of increased future fragmentation. This capability primarily affects Best-Effort and Stubborn policies, as Base policies typically lack the path-wide perspective necessary to make such determinations.

For example, consider a path of 5 links with 7 slots each, and the current occupancy pattern:

$$\mathbf{S} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Suppose the allocator assigns slots 0–2 on links 0–2. For links 3–4, these slots cannot be reused because continuity is required. The allocator now considers the next contiguous candidate: slots 3–5. Even though slots 3–5 are free on links 3–4, this selection would fragment a large free block that exists upstream (on links 1–2), so it is rejected. The allocator must continue searching and finds slots 4–6, which are an available solution because using them does not split a contiguous free block.

When `can_break` is `true`, the allocator is allowed to split large free blocks on the current link in order to maintain continuity with the previous allocation. This means that even if selecting a particular contiguous block would fragment a larger free region upstream, the allocation is permitted. Using the same example as before (5 links with 7 slots), suppose links 0–2 have slots 0–2 assigned. For links 3–4, slots 3–5 are now considered valid even if this would break a large unfragmented free region that exists upstream, because `can_break=true` removes the restriction. This increased flexibility allows the allocator to find a valid path more easily, reduces blocking probability, and improves resource utilization, but it comes at the cost of creating fragmented free spectrum on the link, which could make future allocations slightly harder to fit.

Implementation Heuristics

Within each policy family, spectrum assignment employs specific low-level heuristics to select individual spectrum blocks from available candidates:

- **First-Fit:** Scans candidate slot positions in ascending frequency order and selects the first block meeting the bandwidth requirement. This approach offers simplicity and speed but may lead to suboptimal spectrum utilization over time. Applied in Base, BE, and ST variants.

- **Random-Fit:** Selects randomly from all suitable candidate blocks of sufficient size. This approach promotes spectral load balancing and helps avoid pathological allocation patterns that can occur with deterministic methods. Implemented for Base, BE, and ST families to provide variability and fragmentation mitigation.
- **Longest-Fit (Largest-Fit):** Selects the largest contiguous block available to preserve large regions of spectrum for future high-capacity requests. This approach aims to reduce fragmentation by leaving larger contiguous free blocks available for future allocations. Is used primarily in BE and ST families where block enumeration is available.

These heuristics are implemented across all policy families, with each family employing them in different contexts and with varying levels of sophistication. The Base family typically applies these heuristics to immediate availability information, while Best-Effort and Stubborn families apply them to enumerated candidate sets with path-wide visibility.

Unified Policy Framework

The spectrum allocation framework implemented in this work can be understood as a multi-dimensional policy space where any specific allocation strategy is defined by the combination of:

1. **Policy Family:** The fundamental approach (Base, Best-Effort, or Stubborn)
2. **Wavelength Conversion Setting:** The continuity requirement across links
3. **Break Capability Setting:** The permission to fragment large free blocks
4. **Selection Heuristic:** The method for choosing individual blocks (First-Fit, Random-Fit, or Longest-Fit)

This combinatorial approach creates a rich design space for spectrum allocation strategies. For example, a `first_fit_best_effort` policy with `conversion=True` and `can_break=False` will perform a first-fit search on a path-wide basis, allowing different slots per link but avoiding the splitting of large free blocks. Conversely, a `longest_fit_stubborn` policy with `conversion=False` and `can_break=True` will exhaustively search for the largest possible contiguous block that can be allocated end-to-end, even if it requires fragmenting existing free regions.

This unified framework allows network operators to select the precise combination of policies and parameters that best aligns with their specific network

characteristics, traffic patterns, and performance requirements, ranging from simple and fast allocations to complex and optimal ones.

The logical progression from policy families to operational constraints to implementation heuristics provides a structured understanding of spectrum allocation, moving from high-level strategies to specific implementation details while preserving a clear relationship between all system components.

In the following sections, these families and heuristics are described in detail, together with the helper functions that the control plane provides to (i) compute per-link and end-to-end availability, (ii) extract candidate blocks, and (iii) perform allocation attempts (including conversion and optional `can_break` behavior where supported).

First-Fit Allocation Policy

The First-Fit (FF) policy is a simple and widely used greedy spectrum-allocation strategy: it scans frequency slots in ascending order and assigns the first contiguous block that satisfies the requested bandwidth. Although conceptually straightforward, practical implementations vary depending on whether wavelength conversion is allowed and on additional constraints such as slot reuse or partial-block handling. In this work we implement three First-Fit variants — *base*, *best-effort* and *stubborn* — that share the same greedy principle but differ in flexibility, computational cost and robustness in fragmented or constrained scenarios. The **base** variant provides a minimal, low-complexity assignment; **best-effort** relaxes certain alignment constraints to exploit fragmented spectrum; and **stubborn** augments the greedy search with backtracking to recover from unlucky early choices.

First-Fit Base The First-Fit Base policy serves as the foundational algorithm for spectrum allocation and operates in two distinct modes, depending on the availability of wavelength conversion capabilities.

In the mode **without wavelength conversion** the algorithm requires that the same contiguous slot block be available on every link of the path. The algorithm first computes a single binary spectrum list where a slot is marked as available (1) only if it is free on every link of the path. This is achieved via a logical AND operation across all link spectra (`get_available_boolean_spectrum`). This aggregated spectrum is then converted into a list of contiguous free blocks (`create_vector_spectrum`), which serves as the input for the allocation routine. It scans that list for the first block, if no common block satisfies the bandwidth requirement the request is rejected. In this mode, the algorithm enforces a strict spectrum continuity constraint across the

entire optical path. Let

$$P = \{l_1, l_2, \dots, l_n\}$$

be a route consisting of n links. The set of all spectrum blocks that are commonly free on every link $l_i \in P$ is denoted as

$$F_{\text{common}}$$

Each block $b \in F_{\text{common}}$ is characterized by its starting slot index $s(b)$ and its size $z(b)$ (i.e., the number of contiguous slots it spans).

For an incoming connection request with bandwidth requirement B (expressed in number of slots), the algorithm selects:

$$b^* = \arg \min_{b \in F_{\text{common}}} s(b) \quad \text{subject to } z(b) \geq B.$$

If such a block b^* exists, it is allocated on every link $l_i \in P$. Otherwise, the request is blocked. This mode has a time complexity of $O(|F_{\text{common}}|)$ and is straightforward to implement, but it can lead to higher blocking probabilities because of the stringent requirement that the same contiguous slots must be available across all links of the path.

In the mode **with wavelength conversion**, each link is treated independently: the input is a binary availability matrix (one row per link), the algorithm identifies contiguous free blocks on each link and assigns the first block that is large enough; if any link lacks a suitable block, the allocation fails. The conversion-enabled mode is more flexible (because allocations may differ between links), but it requires wavelength converters at the nodes. When wavelength conversion is available, the spectrum continuity constraint is relaxed. The input is a binary availability matrix A of dimension $n \times F$, where F is the total number of frequency slots, and $A[i, j] = 1$ indicates that slot j is free on link l_i . The algorithm processes each link l_i independently. For each link, it identifies the set of contiguous free blocks F_i . It then selects for each link:

$$b_i^* = \arg \min_{b \in F_i} s(b) \quad \text{subject to } z(b) \geq B$$

If a valid block b_i^* is found for every link $l_i \in P$, the allocation is successful. The time complexity is $O(n \cdot F)$. This mode is more flexible and can achieve lower blocking probability, but it requires the presence of wavelength converters at network nodes, increasing capital expenditure.

Numerical example To illustrate the baseline First-Fit behaviour, consider the linear topology composed of six nodes and five serial links:

Route: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$,

the path traverses 5 links (links between node pairs (0, 1), (1, 2), (2, 3), (3, 4), (4, 5)). Each link provides 15 spectrum slots indexed from 0 to 14. The requested lightpath requires a contiguous block of $\text{required_bandwidth} = 3$ slots on every link.

The initial per-link spectrum availability (1 = free, 0 = occupied) for the five links on the route is:

Initial spectrum (rows = links 0–4, cols = slots 0–14):

$$\mathbf{S}_0 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

	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 3.4: Initial available matrix

First assignment attempt (baseline, no conversion): Under the baseline First-Fit-based policy without wavelength conversion, the same contiguous slot indices must be available on every link. The algorithm scans from the lowest slot index upward and selects the first common block of size 3. In this example, the earliest such common block is the slots {0, 1, 2} on all links. Thus, the first assignment allocates position 0 with size 3 on every link. After allocation, the spectrum becomes:

After first assignment no conversion (slots 0–2 set to occupied on all links):

$$\mathbf{S}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

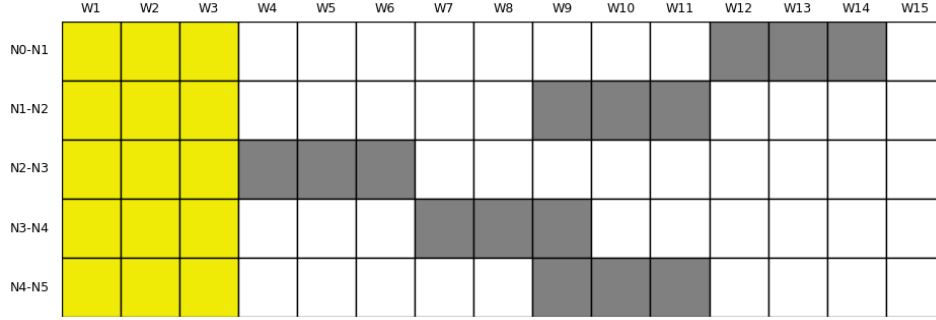


Figure 3.5: After first assignment (slots 0–2 set to occupied on all links).

Second assignment attempt (baseline, no conversion): A second connection request with the same bandwidth (3 slots) arrives. The baseline no-conversion policy again looks for a block of three contiguous slots that is *identically* free on all links. Scanning S_1 from the lowest index upward reveals that there is no common contiguous block of size 3 that is free on every link (the previously free slots are fragmented differently across links). Therefore, the allocation fails under the no-conversion baseline: no additional identical end-to-end block exists, and the spectrum matrix remains S_1 .

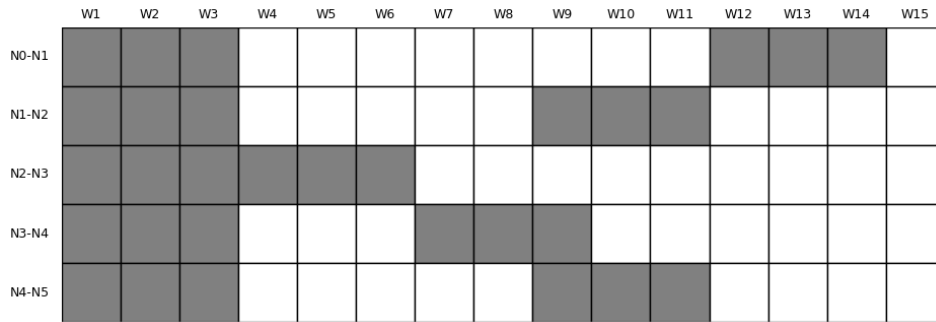


Figure 3.6: Second assignment attempt (baseline, no conversion): allocation fails

Baseline with wavelength conversion: For comparison, if wavelength conversion is permitted, the algorithm may choose independent contiguous blocks per link (i.e., different slot indices on different links). Applying the First-Fit rule per link (scan each link separately and pick the earliest fitting block) to the same post-first-assignment availability S_1 yields a successful allocation for the second request. In the observed execution, the chosen per-link blocks (position, size) were:

link index	chosen block (position,size)
0	(3, 3)
1	(3, 3)
2	(6, 3)
3	(3, 3)
4	(3, 3)

Applying these per-link allocations produces the following spectrum after the second assignment (with conversion enabled):

After second assignment (with conversion):

$$\mathbf{S}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

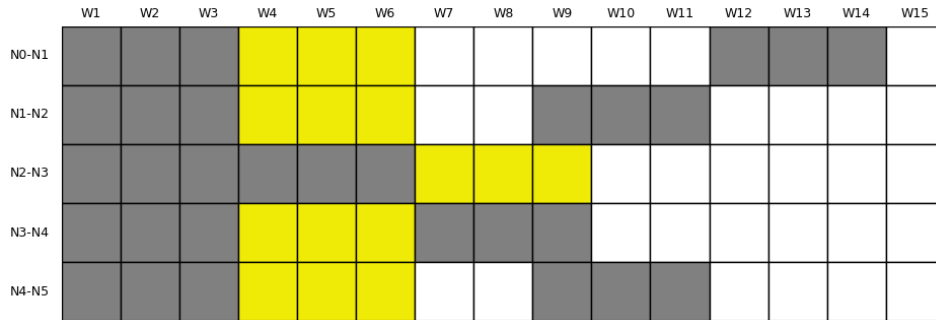


Figure 3.7: After second assignment (with conversion)

This numerical example highlights the principal difference between the two baseline modes:

- *Without wavelength conversion*, allocations must use identical slot indices across every link; fragmentation and misaligned free regions across links quickly prevent additional allocations.
- *With wavelength conversion*, independent per-link assignment allows the algorithm to exploit local free segments and often admits requests that would otherwise be blocked, at the expense of requiring wavelength converters at intermediate nodes.

First-Fit Best-Effort The First-Fit Best-Effort policy is a greedy allocation designed for networks with wavelength conversion. It introduces the boolean parameter `can_break`, which controls an alignment/adjacency constraint across neighboring links: with `can_break = false` a stricter adjacency rule is enforced (certain overlaps with the immediately preceding link are rejected), while with `can_break = true` that alignment restriction is relaxed and each link may be assigned any contiguous block that fits the request even if adjacent links start at different slot indices.

The routine scans candidate contiguous blocks in ascending slot order (first-fit), and for each candidate it assigns that block to as many consecutive links as the candidate covers. The assignment is final (no backtracking). If, after the scan, any link along the path remains unallocated, the request is rejected.

Operationally, the algorithm first inspects the availability matrix

$$A \in \{0, 1\}^{n \times F},$$

where n is the number of links on the path and F is the number of frequency slots; $A[i, j] = 1$ means slot j is free on link l_i . From this matrix the algorithm extracts all contiguous free blocks of length at least B and represents each one as a candidate described by its starting slot, the link where it begins, and how many consecutive links it covers. Candidates are considered in the natural first-fit order, i.e. sorted by increasing starting slot within each starting-link group.

Allocation proceeds in a single forward pass along the path. A pointer called `current_link` begins at the first link and the allocator repeatedly selects the earliest (first-fit) candidate that is valid for the current context and assigns it to the maximum number of consecutive links it covers. When a candidate is accepted, the corresponding starting slot is recorded for each of those links and the pointer advances past the covered links; the choice is final and the algorithm does not backtrack. If the scan reaches a link for which no valid candidate exists, the request is blocked.

The meaning of “valid” depends on the `can_break` setting. When `can_break = false` the allocator enforces a stricter adjacency constraint: it avoids selecting

a candidate that would break up a large, unfragmented free region on the current link in a way that conflicts with upstream availability. Concretely, this means some candidate placements that would be locally free are rejected because they would fragment a clean block that upstream links still consider available. When `can_break = true` this adjacency restriction is relaxed and any candidate that covers the current link may be chosen regardless of alignment with previously selected blocks; the allocator can therefore use different starting indices on adjacent links and accept local fragmentation in order to complete the request.

First-Fit Best-Effort is a fast, greedy routine (typical implementation cost on the order of $O(n \cdot F)$ for candidate extraction and scanning) that trades optimality for simplicity. With `can_break=false` it prioritizes preserving large contiguous regions and limiting fragmentation; with `can_break=true` it prioritizes acceptance probability by allowing per-link flexibility. The choice of the flag therefore implements a practical policy trade-off between long-term spectrum compactness and immediate blocking reduction.

First-Fit Stubborn The **First-Fit Stubborn** policy is a hybrid allocation strategy that combines the simplicity of first-fit selection with the completeness of **recursive backtracking**. It is implemented as a first-fit allocation procedure that, when a local choice leads to a dead end, undoes that choice and explores alternative candidates until either a complete path allocation is found or a configurable attempt limit is reached. This implementation targets the wavelength-conversion mode only and represents the allocation result as a list of Spectrum objects, one per link. The routine begins by preparing a safe failure sentinel (`allocation_failure`) and a hard cap on the total number of candidate selection attempts (`MAX_ATTEMPTS`) across the entire recursive search tree. This cap limits the computational effort by counting every time a candidate is tried, regardless of the recursion depth, ensuring the algorithm terminates even in complex scenarios. All candidate spectrum blocks that satisfy the requested bandwidth are precomputed once via `extract_available_blocks_from_matrix_jit`; these candidate blocks are stored in a map keyed by `(start_link, start_slot)` with values describing the block bandwidth and how many consecutive links it covers. Precomputing candidates keeps the recursive search focused on feasible blocks and avoids repeated scanning of the full spectrum matrix. The heart of the algorithm is a **recursive function** that advances along links from the lowest index to the highest. At each recursion step it first enforces the maximum attempts guard and then checks the base case: if the current `link_idx` is equal to or greater than the number of links, a complete allocation has been built and is returned. The routine also skips links that were already assigned as part of a previous multi-link block so the search only

attempts allocation for truly free link positions.

Candidate selection depends on the **can_break flag**. When `can_break` is `true`, any candidate block that begins at the current link is considered valid; the algorithm allows allocating a contiguous subset of links from that candidate's coverage.

When `can` is `false`, the algorithm applies a stricter filter: only candidates that start exactly at the current link, that do not exceed the path length, and that do not conflict with the immediately preceding link are allowed. The adjacency check excludes candidates whose slots are entirely occupied in the previous link, ensuring alignment with already committed allocations—unless the previous link is unallocated, in which case the candidate remains valid.

After filtering, candidates are ordered for first-fit behavior by ascending starting slot. The function then attempts each candidate in sequence, incrementing the shared attempts counter before each trial so the overall search can be aborted when `MAX_ATTEMPTS` is reached. For a chosen candidate, the routine tentatively marks the corresponding links with Spectrum entries and recurses to continue allocation beyond the current block. If recursion succeeds and returns a full allocation, that solution is propagated upward; if it fails, the function implicitly backtracks by discarding the tentative marks and trying the next candidate. If all candidates fail at a given recursion level, the function returns `None` to signal a dead end and trigger backtracking in the caller. FF-Stubborn therefore behaves like a depth-first tree search where nodes correspond to candidate block choices and branches represent alternative allocations for a link segment. This makes the policy complete in the sense that it will find a feasible allocation if one exists within the configured attempt budget, but it also exposes the algorithm to exponential worst-case runtime. The `MAX_ATTEMPTS` parameter is a pragmatic control to trade off search thoroughness against runtime, guaranteeing termination even when no full allocation exists. In practical terms FF-Stubborn is most useful when simple greedy strategies (such as plain first-fit or Best-Effort) fail due to early choices that create later dead ends—particularly in highly fragmented spectra or under strict alignment constraints. It is more robust than greedy approaches but significantly more computationally expensive, so it is best applied selectively (for instance, as a fallback when faster heuristics cannot find a path) or with conservative attempt limits when used within large-scale simulations.

Best Effort vs Stubborn: numerical Example We start from an **initial spectrum allocation matrix** representing 5 network links (rows 0 to 4) and 15 frequency slots (columns 0 to 14). Each entry is either:

- 1 = slot **free/available**
- 0 = slot **occupied/unavailable**

Initial spectrum (rows = links 0–4, cols = slots 0–14):

$$\mathbf{S}_0 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Starting from the initial spectrum $\mathbf{S}_0$ (five links, 15 slots each), all four combinations (Best-Effort / Stubborn $\times$ can_break = true/false) produce the same, easy first allocation from the initial spectrum: link 0 is assigned the contiguous block starting at position 3 (slots 3–5) and links 1–4 are assigned the contiguous block starting at position 0 (slots 0–2).

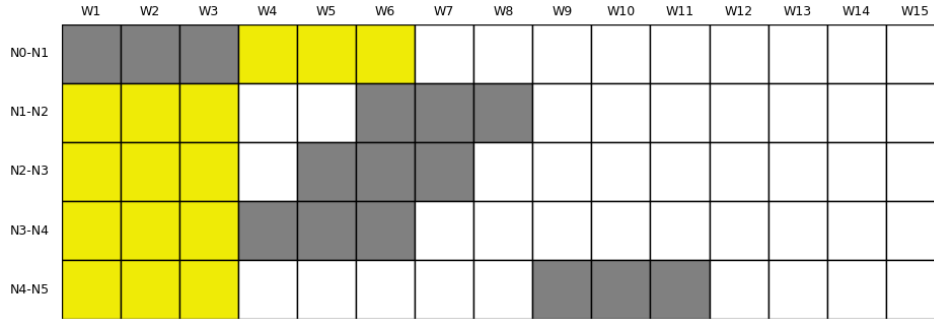
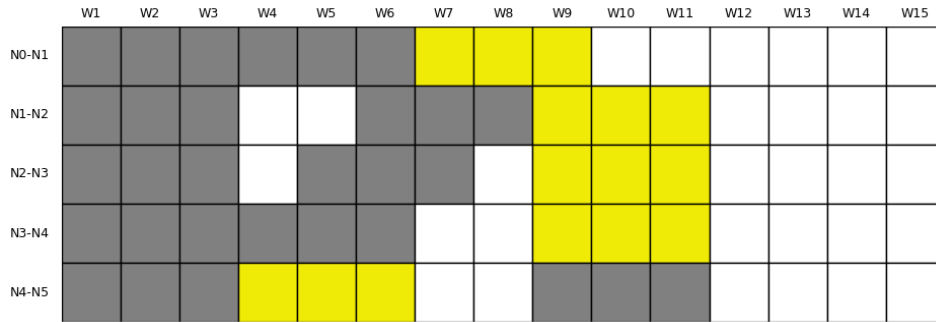


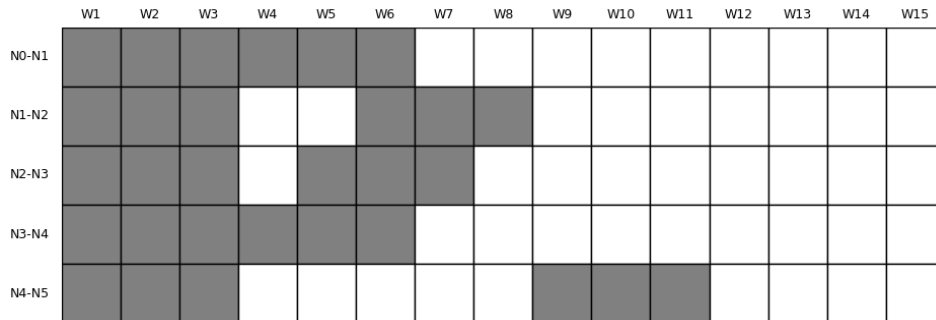
Figure 3.8: First Assignment (First-fit Best-Effort / Stubborn $\times$ can_break = true/false)

The divergence appears on the second allocation attempt:

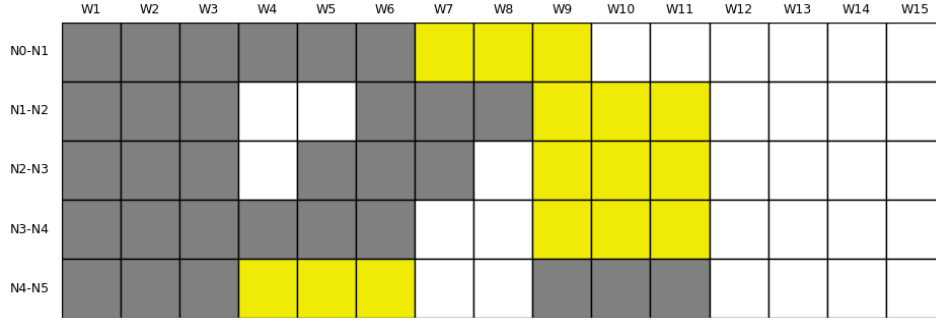
1. First-fit Best-Effort with can_break = true: it finds per-link contiguous blocks at different start indices (logged starts: 6 for link 0, 8 for links 1–3, and 3 for link 4) and updates the spectrum accordingly. Because alignment across links is not required, the greedy scan can exploit remaining fragmented pockets independently on each link.
2. First-fit Best-Effort with can_break = false fails on the second attempt: the strict adjacency checks combined with the first-fit choice prevent any valid

Figure 3.9: Second assignment in First-fit Best-effort `can_break=True`

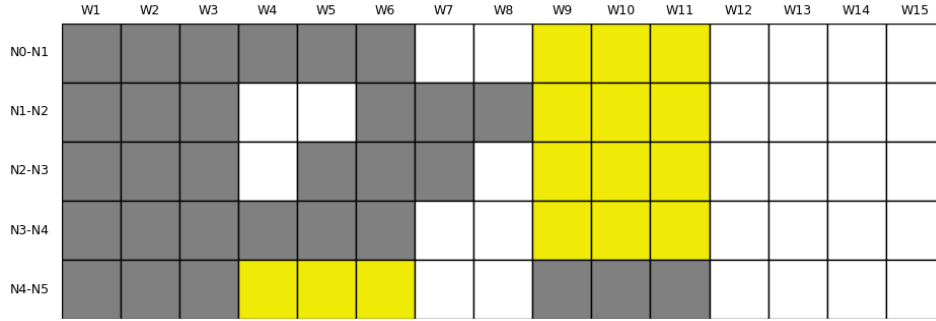
contiguous start positions that satisfy all links, so each link returns $(-1, -1)$ and the spectrum stays the same. Under this strict mode, the routine is limited in how it builds `valid_keys` (it searches for blocks starting at specific positions and enforces adjacency and overlap checks), and the greedy first-fit method can't find a single valid contiguous choice per link given the current fragmentation — it never considers later candidate starts once earlier options lead to dead ends.

Figure 3.10: Second assignment in First-fit Best-effort `can_break=False`

3. First-fit Stubborn with `can_break = true` behaves like the First-fit Best-Effort `can_break = true` case (it finds per-link varying starts such as 6, 8, 8, 8, 3) because best-effort already succeeds there.
4. First-fit Stubborn with `can_break = false` does not stop at the first failing candidate; it explores alternative candidate blocks recursively and backtracks

Figure 3.11: Second assignment in First-fit Stubborn with `can.break = true`

when a choice prevents future allocations. Here, the stubborn no-break second assignment succeeds by ultimately selecting start position 8 for links 0–3 and 3 for link 4 ((8,3),(8,3),(8,3),(8,3),(3,3)), a combination the first-fit best-effort never reached. In short, best-effort gave up; stubborn tried other combinations and found a feasible alignment.

Figure 3.12: Second assignment in First-fit Stubborn with `can.break=False`

So, `can.break` controls alignment across links (it allows per-link start indices to differ when true), while the search strategy (best-effort vs stubborn) controls whether the allocator will try alternate start positions when the obvious first choices lead to dead ends. In the example, the first allocation was easy and identical for all modes; the second allocation succeeds for any mode that either allows per-link shifting (`can.break=True`) or, if alignment is required, for a search that is willing to backtrack (stubborn `can.break=False`). Best-effort `can.break=False` is the one that fails. When running the Best-Effort allocator with `can.break = True`, the second

assignment succeeds by choosing different contiguous blocks per link (logged starts: 6, 8, 8, 8, 3). `can_break = True` allows each link to pick its own start index (blocks remain contiguous on each link but need not be aligned across links), so the greedy scan finds available pockets and succeeds.

Random-Fit Allocation Policy

The Random-Fit (RF) policy is a probabilistic spectrum-allocation strategy in which, instead of scanning slots in a fixed order, the algorithm randomly selects a contiguous free block from the set of all blocks that meet the bandwidth requirement. This randomization helps distribute spectrum usage more evenly across the frequency range, potentially reducing fragmentation caused by deterministic policies such as First-Fit. As with FF, the implementation can vary depending on whether wavelength conversion is allowed and on constraints like slot reuse or partial-block handling. In this work we implement three Random-Fit variants — *base*, *best-effort* and *stubborn* — that mirror the flexibility and search strategies of their First-Fit counterparts, but introduce randomness in block selection to balance load and improve spectrum utilization in the long term.

Random-Fit (base) The Random-Fit base routine implements the simplest idea: pick a suitable free block at random instead of always taking the first one. In the no-conversion mode the input is a list of precomputed contiguous blocks that are free on every link; the routine filters those blocks to the ones large enough and then selects one uniformly at random, assigning that identical block to every link. In the conversion mode the input is a per-link binary availability matrix; for each link, the routine enumerates all contiguous free blocks, filters them to those with *size larger or equal required_bandwidth*, and then selects one of these valid blocks uniformly at random. If the selected block is larger than the request, a random start position within that block is chosen so that the required number of contiguous slots fits. If any link has no valid block the whole request fails. The base policy therefore trades deterministic/fragmentation-aware placement for randomness that can help spread load or avoid worst-case patterns.

Random-Fit Best-Effort The Best-Effort variant extends the base per-link random choice to multi-link allocations and introduces the familiar `can_break` flag. The routine builds a set of candidate multi-link blocks (keyed by link index and start slot) and then walks the path: if `can_break = true` it accepts any candidate that covers the current link (so each link can take a contiguous block starting at a different position); if `can_break = false` it applies stricter adjacency checks and only accepts candidates that start exactly at the current link and do not produce forbidden overlaps with the

previous link. The key difference from First-Fit is that when there are multiple valid candidates the algorithm picks one uniformly at random rather than deterministically picking the earliest. If, after filling as many links as a chosen candidate covers, some link remains unallocated the routine continues scanning; if any link ends up without a block the request fails. Because choices are random, two runs on the same input can produce different allocations and fragmentation outcomes.

Random-Fit Stubborn (backtracking + randomness) The stubborn (backtracking) variant combines randomness with a recursive search. It extracts the same candidate set as Best-Effort, shuffles it (randomized ordering) and then attempts to allocate by recursion (at maximum *MAX_ATTEMPTS* times): pick a random candidate for the current link, assign it to the covered links, and recurse to allocate the remainder. If the recursion hits a dead end, it backtracks and tries the next random candidate. This preserves the advantage of randomness (diverse allocations across runs) while gaining the ability to recover from early unlucky choices by exploring alternatives. In short: Best-Effort = random greedy; Stubborn = random search with backtracking.

Considerations: The Random-Fit policy distributes allocations stochastically, thereby avoiding the systematic, pathological placement patterns that deterministic First-Fit may produce under repeated, similar requests. This randomization promotes spectral load balancing and increases diversity in slot utilization. The Best-Effort Random variant provides rapid allocations with complexity $O(n)$; however, it is more likely to fail in heavily fragmented spectra because an initially unlucky random choice can fall within unusable regions. By contrast, Stubborn Random applies backtracking and thus systematically explores permutation spaces, improving the probability of finding a valid allocation at the expense of higher worst-case complexity, typically $O(n^k)$ where k denotes the backtracking depth.

Policy selection should be driven by the fragmentation characteristics of the spectrum and by the relative importance of allocation success versus allocation latency. Under strict alignment constraints (`can_break=false`) and in strongly fragmented spectra, Stubborn Random maximizes the likelihood of successful allocation and is therefore the preferred choice despite its computational cost. For environments with moderate fragmentation where responsiveness is prioritized, Best-Effort Random combined with `can_break=true` yields a favourable trade-off between speed and success rate. In cases of low fragmentation or when long-term load balancing is the primary objective, the basic Random-Fit policy is appropriate due to its simplicity and its tendency to prevent pathological placement patterns.

Numerical Example: We start from an **initial spectrum allocation matrix** representing 5 network links (rows 0 to 4) and 15 frequency slots (columns 0 to 14). Each entry is either:

- 1 = slot **free/available**
- 0 = slot **occupied/unavailable**

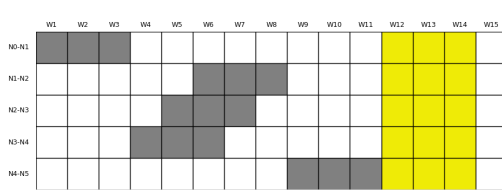
Initial spectrum (rows = links 0–4, cols = slots 0–14):

$$\mathbf{S}_0 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

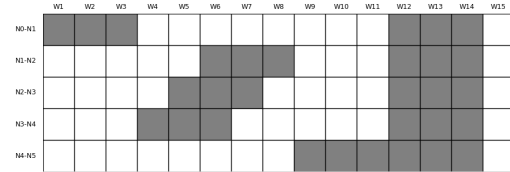
1. Random-fit base:

- (a) Random-Fit Base no conversion: the algorithm tries to find the same contiguous block of free slots across all links in the path. A valid candidate block must be free on every link at the same position. From these valid blocks, the algorithm picks one uniformly at random and applies it to all links.

In the first assignment, the only chosen block was (pos=11, size=3), so all links had these slots marked as occupied. When the second request comes, no contiguous block of the required size exists across all links at the same position, so the list of valid candidates is empty — all positions return as (-1, -1).



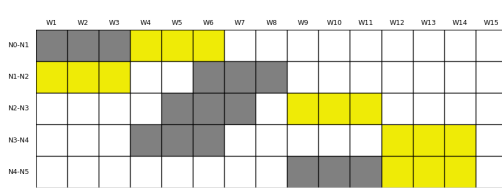
(a) First assignment RF base no conversion



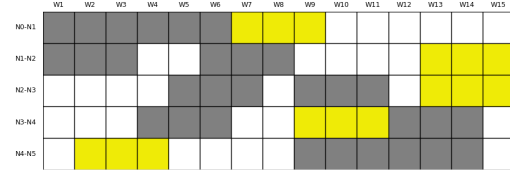
(b) Second assignment RF base no conversion

Figure 3.13: Random-Fit Base without conversion

- (b) Random-Fit Base with Conversion: here, wavelength conversion allows different slot positions to be chosen per link. The algorithm scans the link's row to find contiguous blocks of free slots (1 values), filters to blocks where the size is equal to or larger than the required bandwidth. Then, it chooses one valid block at random for that link. From within that block, it picks a random starting position so the required slots fit. In the first assignment, each link picks a suitable free block independently: positions 3, 0, 8, 11, and 11 with size 3. After marking these as occupied, the second assignment again selects different positions per link (6, 12, 12, 8, and 1), successfully allocating another set of blocks. The updated matrix shows a more scattered occupation, as slot alignment between links is not required.



(a) First assignment RF base with conversion



(b) Second assignment RF base with conversion

Figure 3.14: Random-Fit Base with conversion

2. Random-Fit Best Effort: the code generates a dictionary of candidate blocks from the matrix, keyed by (start_link, start_slot) with values (bandwidth, links_covered).

- (a) can_break=True: Any block that covers the current link is valid. The valid_keys list will include candidates starting before the current link if the block spans across it. Random choice from these keys allows partially overlapping allocations between links, so fragmentation is tolerated and gaps get filled faster. In the first assignment, different positions (3, 8, 8, 8, and 11) are chosen and marked. The second assignment again finds blocks at positions 6, 12, 12, 12, and 0, further filling scattered gaps. Note that, as Fig. 3.15(a) shows, links 1, 2, 3, 4 choose blocks starting from position 8, 8, 8, 11, even though those slots are free in previous links, this behavior is allowed since we enable the can_break mode.

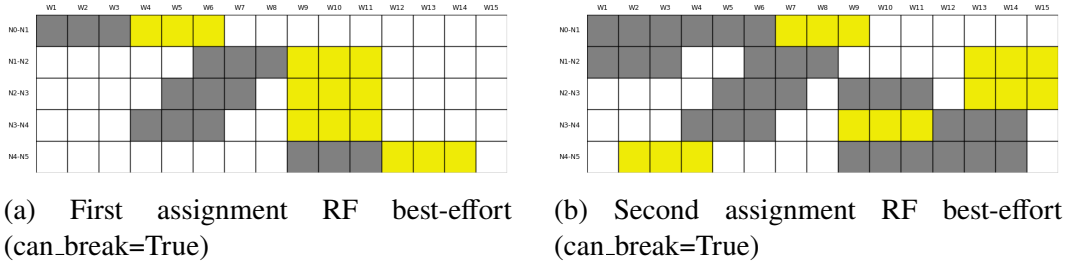


Figure 3.15: Random-Fit Best Effort with conversion (can_break=True)

(b) can_break=False: A valid block must start exactly at the current link and fit within the remaining hops. Additionally, the previous link must not have all these slots free to avoid overlapping free space. This helps keep the spectrum more continuous. Only contiguous, same-size blocks per link are chosen, leading to more compact occupation patterns. In both assignments, the yellow blocks (chosen blocks) start from the current link. For example, in Fig. 3.16(a), link 4, the chosen block starts at position 1 with size 3; it is allowed because, in link 3, slot 3 is not free, so we are not breaking contiguous blocks.

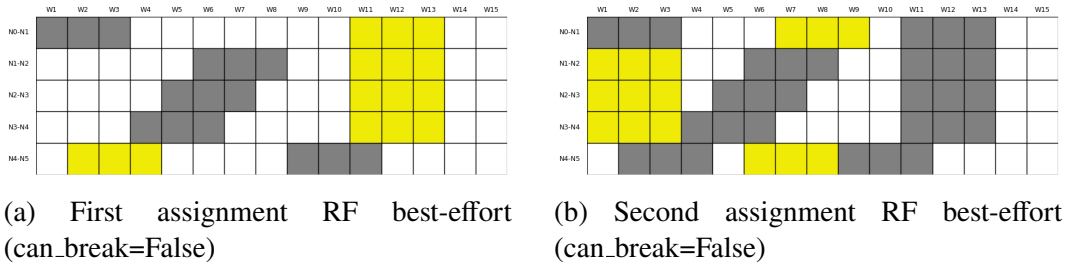


Figure 3.16: Random-Fit Best Effort with conversion (can_break=False)

3. Random-Fit Stubborn: This uses backtracking to find an allocation if one exists. Like Best Effort, it builds a candidate dictionary first, but instead of moving forward greedily, it randomly shuffles the candidate keys for the current link, tries one allocation, and recursively proceeds to the next unallocated link. If a dead end is reached, it backtracks and tries the next candidate. This method tends to produce more aligned patterns (especially when can_break=False), because the backtracking will explore allocations that keep spectral continuity before settling for fragmented ones.

- (a) `can_break=True`: all blocks starting at the current link are candidates, regardless of overlap, and the algorithm prefers keeping allocations aligned if possible, even when splitting is allowed. The first assignment picks positions 5, 8, 8, 8, and 1, marking them accordingly. The second assignment finds the same contiguous block at position 11 on all links, filling it uniformly.

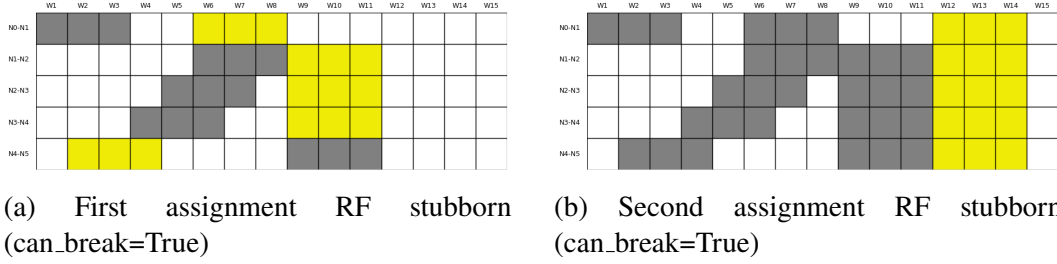


Figure 3.17: Random-Fit Stubborn with conversion (`can_break=True`)

- (b) `can_break=False`: stubborn mode still enforces alignment but only with contiguous blocks, producing a denser and more uniform occupation pattern, applying the same non-overlap rule as Best Effort.

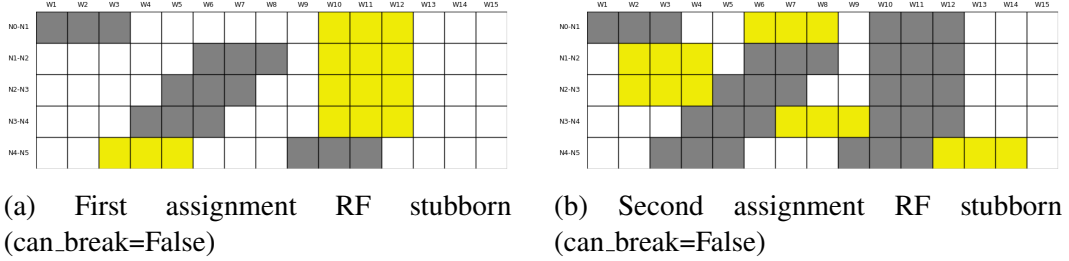


Figure 3.18: Random-Fit Stubborn with conversion (`can_break=False`)

Longest-Fit Spectrum Allocation Policies

The implementation provides two Longest-Fit variants: `longest_fit_best_effort` and `longest_fit_stubborn`. Both policies utilize a common helper function, `extract_available_blocks_from_matrix_jit`, to generate a candidate map. This map is keyed by tuples of `(start_link, start_slot)` and stores values of `(bandwidth, links_covered)`, representing the size of the contiguous free block and

the number of consecutive links it spans. The two variants diverge significantly in their search strategy for selecting from this candidate set.

Longest-Fit Best-Effort The Longest-Fit Best-Effort method implements a greedy, multi-link allocation that always prefers candidate blocks covering the largest number of consecutive links.

The procedure begins by extracting a global candidate map of contiguous free blocks that satisfy the requested bandwidth; each candidate is annotated by its starting link, starting slot, block size and the number of consecutive links it can cover.

If the `can_break` flag is `false`, the algorithm applies an additional filter that discards any candidate whose entire slot range is free on the immediately preceding link. This *no-break* rule prevents the algorithm from reusing the exact same spectral slice on adjacent links, thereby enforcing an alignment constraint that helps maintain larger contiguous free regions. The remaining candidates are then sorted by coverage length (descending) and by start slot (ascending). The allocator scans this sorted list and greedily assigns each longest non-overlapping candidate to all unallocated links it covers (assignments are truncated at the path end using a `min(..., total_links)` guard). Links covered by a chosen block are marked as allocated and skipped in subsequent checks; the routine stops early once every link has been assigned. If, after exhausting all candidates, any link remains unallocated, the function returns a uniform failure (a list of `Spectrum(-1, -1)` entries).

From a complexity standpoint the dominant costs are candidate extraction and the sort: if m denotes the number of candidates, sorting is $O(m \log m)$ and assignment is linear in the number of candidates and links. In practice the number of candidates grows with fragmentation and path length, so runtime depends on the network state.

Functionally, Longest-Fit Best-Effort favours continuity by assigning long, multi-link blocks first, reducing local decisions and improving spectrum utilization when wavelength conversion is not available. The default `can_break=true` permits identical slot ranges to be reused across adjacent links (useful when continuity is desired), while `can_break=false` enforces stricter per-hop distinctness at the cost of reduced flexibility and potentially higher blocking. Compared with exhaustive (backtracking) variants, the Best-Effort approach is fast and simple but may fail in situations where a greedy longest choice blocks a later feasible arrangement; in such cases a stubborn/backtracking variant can recover by exploring alternative candidates. Overall, Longest-Fit Best-Effort is appropriate when one wants a fast allocator that promotes long continuous allocations, and when occasional failures due to greedy choices are acceptable.

Longest-Fit Stubborn The Stubborn policy is a recursive backtracking strategy designed to allocate spectrum resources by prioritizing candidate blocks according to their coverage—the number of consecutive links they span—and, secondarily, by their starting slot. The process begins by extracting all candidate blocks of the required bandwidth from the spectrum matrix. These blocks are then grouped by their starting link and sorted so that those with greater coverage come first, and within equal coverage, those with lower starting slots are prioritized.

The algorithm operates recursively. At each step, it identifies the first unallocated link, which is the lowest-index link not yet assigned. From there, it collects all candidate blocks that begin at any free link and that fit within the spectrum matrix without exceeding the number of links or slots. If the `can_break` flag is set to `false`, the algorithm further restricts these candidates by excluding any block that would create an adjacency conflict with the previous link. An adjacency conflict occurs when the previous link already has all of the slots of the candidate block occupied. This condition ensures that, whenever possible, allocations are aligned with what has already been assigned. If the previous link is unallocated, however, the candidate is still considered valid.

Once filtered, the valid candidates are sorted again by coverage in descending order and by starting slot in ascending order. The algorithm then examines each candidate in this order. For each one, it checks how many links can be allocated. If `can_break` is `true`, it allows allocation of a contiguous subset of free links starting from the candidate's initial link up to its coverage. If `can_break` is `false`, it requires that the entire block of links defined by the candidate's coverage be free and allocated as a whole.

When a valid allocation is found, the algorithm updates the spectrum state and recurses to attempt further allocations. If a full allocation is achieved through recursion, the process returns the solution. If not, the algorithm backtracks and explores the next available candidate. Although the method is complete, meaning it is guaranteed to find a solution if one exists, it may take exponential time in the worst case. To prevent excessive computation, a parameter called `MAX_ATTEMPTS` is used to cap the number of recursive trials.

This method is complete and robust in highly fragmented spectra with strict alignment constraints, though it incurs higher computational time compared to greedy methods in simpler scenarios

Numerical Example: We start from an **initial spectrum allocation matrix** representing 5 network links (rows 0 to 4) and 15 frequency slots (columns 0 to 14). Each entry is either:

- 1 = slot **free/available**
- 0 = slot **occupied/unavailable**

Initial spectrum (rows = links 0–4, cols = slots 0–14):

$$\mathbf{S}_0 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The results between the `longest_fit_best_effort` and `longest_fit_stubborn` spectrum assignment policies reveal that their practical differences are negligible in this small example. Both methods consistently prioritize the allocation of the largest contiguous spectrum block available along the path, which means that in scenarios where a dominant block exists, they converge to the exact same allocation decision. This was particularly evident in the first evaluated request (position 11, size 3), where the four combinations (**best_effort** vs. **stubborn** $\times$ `can_break=True` or `False`) selected the same block, resulting in identical link occupancy states.

In more fragmented situations, such as the second evaluated request, the methods exhibited only minor divergences in slot positions when `can_break=True`. For example, the **best_effort** method selected blocks at positions 3 and 0, while the **stubborn** method selected positions 8 and 0 (or 1 in one case). However, the resulting link status matrices remained highly similar, with only marginal differences in slot allocation patterns. When `can_break=False`, both methods again behaved almost identically, typically choosing position 11 first and then a nearby longest block, such as position 8.

These observations highlight that the **longest fit policy itself dominates overall behavior**, rendering the choice between **best_effort** and **stubborn** of limited practical significance. The structural similarity in their results comes from the fact that both strategies rely on the same core logic: prioritizing the largest contiguous spectrum block, which strongly constrains possible variations in output.

In contrast, the `can_break` parameter has a more tangible effect on the allocation patterns. When set to `False`, it imposes stricter continuity constraints across the path, preventing block selection if it fragments free slots on adjacent links. This restriction can alter allocation decisions in highly fragmented spectrum states, occasionally reducing flexibility and increasing blocking probability. Therefore, within the family of **longest-fit** allocation strategies, the tuning parameters such as `can_break` appears

to be much more impactful than the choice between the **best_effort** and **stubborn** variants.

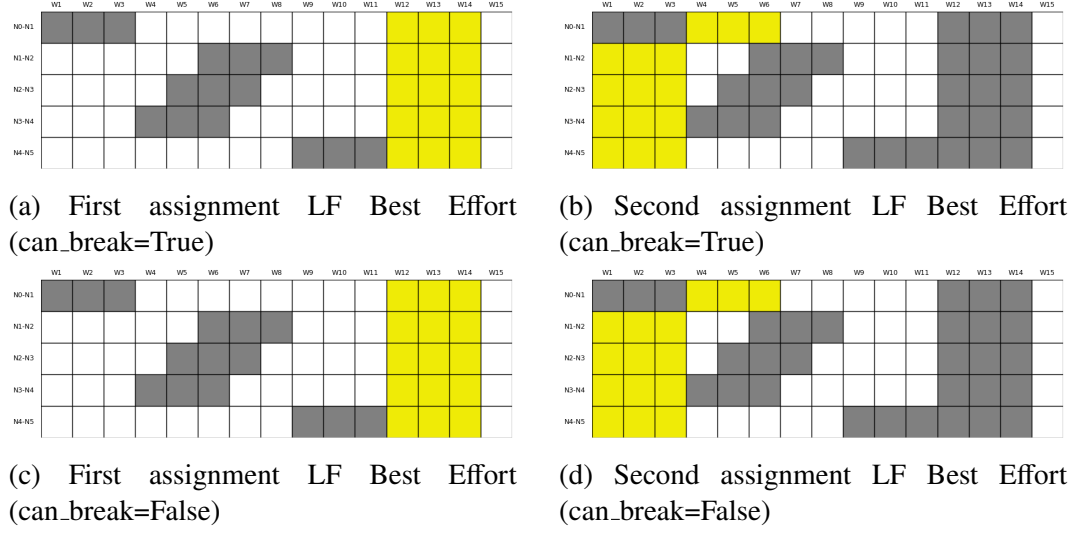


Figure 3.19: Longest-Fit Best-Effort with conversion (can_break=False)

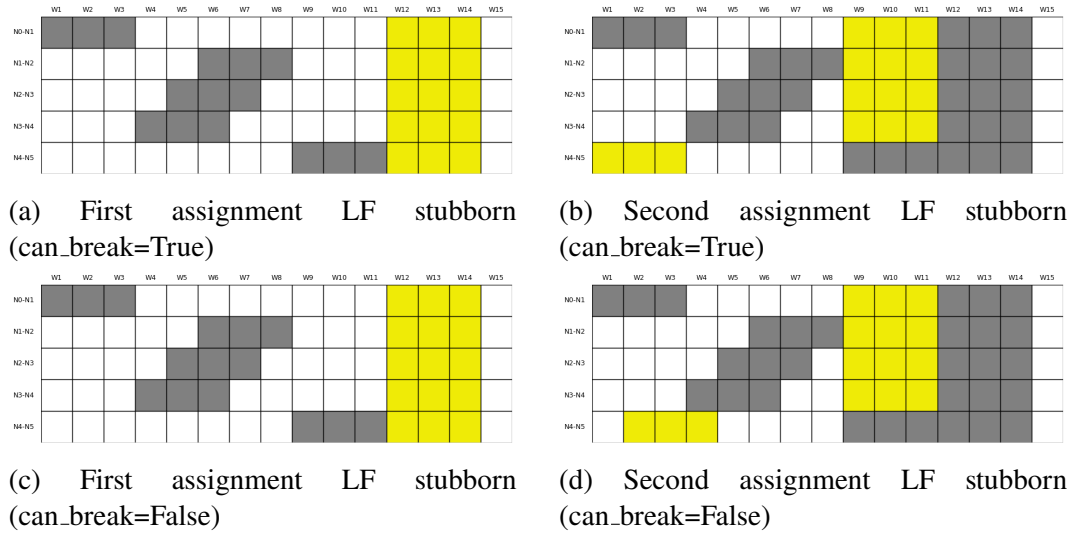


Figure 3.20: Longest-Fit Stubborn with conversion

3.3 Simulation Engine

The simulation engine, implemented within the **Simulation** class, orchestrates end-to-end experiments using a discrete-event simulation (DES) approach. Its operation is divided into distinct phases: configuration, initialization, parallel execution, and analysis.

Configuration and Initialization: At startup, the engine employs a layered configuration system. It first reads a *default_parameters.yaml* file to establish baseline values. Scenario-specific YAML files are then loaded, overriding or extending these defaults to define a unique experiment. This system ensures both consistency across runs and flexibility for specific tests.

From this configuration, the engine automatically calculates several derived parameters crucial for the simulation:

1. **Slots per Fiber:** The number of frequency slots is calculated as

$$N_{\text{slots}} = \left\lfloor \frac{\text{available spectrum bandwidth}}{\text{slot bandwidth}} \right\rfloor.$$

2. **Service Probabilities (p_k):** The probability p_k that a connection requests service class k is either taken directly from the configuration or derived using an inverse-bandwidth rule ($p_k \propto 1/s_k$). This models a realistic scenario where smaller, more common requests occur more frequently, inherently influencing network fragmentation.
3. **Maximum Network Load (Λ):** The theoretical maximum load the network can support is given by

$$\Lambda = \frac{N_{\text{slots}} \cdot N_{\text{links}}}{\bar{h} \sum_k p_k s_k},$$

where N_{links} is the number of physical links and $\bar{h}$ is the average hop count of shortest paths between all edge nodes (precomputed at initialization). This value Λ defines the range of loads (e.g., from 10% to 100% of Λ) evaluated in the experiment.

Parallel Execution: The evaluation of multiple load points is managed via Python's multiprocessing. *Apandas DataFrame*, with one row per load point, is constructed and passed to a multiprocessing pool. Each worker process receives a row, initializes an independent **ControlPlane** (containing routing tables and spectrum policies), and creates a pristine Node array. Each Node object contains output ports modeled as Link objects, which hold the spectrum status, a binary vector of length N_{slots} where 1 signifies a free slot and 0 an occupied one.

Critically, each worker's state is entirely isolated, and its random number generation is deterministic and reproducible: the *worker_seed* is derived from a global base seed plus the unique load point index (e.g., `worker_seed = base_seed + point_index`). This guarantees that runs are both reproducible and independent.

The Event Loop (`ignition()`): The core of the simulation is a heap-based priority queue that manages connection arrival and departure events. Connection arrivals follow a Poisson process. Inter-arrival times (t) and holding times (T_h) are generated using the inverse-transform method:

$$t = -\frac{\ln(U)}{\lambda}, \quad T_h = -\frac{\ln(U)}{\mu},$$

where $U \sim \mathcal{U}(0, 1)$ is a uniform random variate, λ is the arrival rate for the current load point, and $\mu = 1/\text{average_holding_time}$ is the departure rate. The service class for each arrival is selected by weighted sampling from the distribution p_k .

Upon a connection arrival, the engine invokes the configured spectrum assignment policy (e.g., `first_fit_best_effort`), passing it the required bandwidth and the current per-link spectrum availability matrix. The policies abstract the allocation complexity; they return either a list of Spectrum objects (one per link, defining a contiguous block) or a failure indicator. The key differentiating factors between policies are:

1. **Continuity Constraint:** Whether the allocated block must start on the same slot index on every link (no wavelength conversion) or can vary per link (with conversion).
2. **Alignment Constraint:** The `can_break` flag controls whether a block can be chosen even if it would *break* alignment with the previous link's free spectrum.
3. **Search Strategy:** Whether the policy uses a greedy first-fit/random-fit/longest-fit approach (`best_effort`) or employs backtracking to explore alternatives (`stubborn`).

Resource Management and Costing: A successful allocation triggers an update of the network state. `assign_spectrum()` marks the chosen slots as occupied ($1 \rightarrow 0$) on each link of the route. If wavelength conversion is enabled and the chosen slots differ between adjacent links, the `use_converter()` method is called on the intermediate node. This method manages a pool of converter resources, handling both the usage of existing converters and the placement of new ones if necessary. It returns a unique identifier for the converter used, which is stored within the connection object.

The total cost for a successful connection is calculated as:

$$C_{\text{total}} = \sum_{\ell} s_{\ell} c_{\text{slot}} + h c_{\text{hop}} + n_{\text{conv}} c_{\text{conv}} + n_{\text{new_conv}} c_{\text{placement}},$$

where s_{ℓ} is the number of slots used on link ℓ , h is the hop count, n_{conv} is the number of wavelength conversions performed, and $n_{\text{new_conv}}$ is the number of new converter installations required.

When a connection's holding time expires, it is popped from the priority queue and torn down. `release_spectrum()` frees the allocated slots ($0 \rightarrow 1$), and `release_converter()` is called with the stored converter IDs to release those specific resources.

Performance Metrics and Analysis: After processing all connection requests, each worker computes standard performance metrics:

1. **Network Blocking Ratio:**

$$P_b^{(\text{net})} = \frac{N_{\text{blocked}}^{(\text{total})}}{N_{\text{requests}}}.$$

2. **Bandwidth Blocking Ratio:**

$$P_b^{(\text{bw})} = \frac{\sum_k s_k N_{\text{blocked}}^{(k)}}{\sum_k s_k N_{\text{generated}}^{(k)}}.$$

3. **Per-Service Blocking Ratio:**

$$P_b^{(k)} = \frac{N_{\text{blocked}}^{(k)}}{N_{\text{generated}}^{(k)}}.$$

4. **Average Cost Metrics:** The mean and distribution of C_{total} are calculated, both overall and per service class.

The `run()` method collects results from all workers. Confidence intervals (e.g., 95%) for blocking probabilities are calculated using the standard error approximation for proportions ($P_b \pm 1.96 \cdot \sqrt{P_b(1 - P_b)/N}$). For cost metrics, which often follow non-normal distributions, bootstrapping methods are provided for robust interval estimation.

Optimizations: The engine incorporates several key optimizations to ensure computational efficiency, especially given the large number of required simulations.

- **Precomputation:** The ControlPlane precomputes all shortest-path routes between edge nodes at initialization using Dijkstra’s algorithm on a sparse matrix representation of the topology, avoiding expensive path calculations during the simulation loop.
- **JIT-Complied Candidate Extraction:** The critical function `extract_available_blocks_from_matrix_jit()` is decorated with `@njit` from the Numba library. This compiles the function to optimized machine code, providing near-native performance for the intensive task of scanning the spectrum matrix for contiguous free blocks. This optimization benefits all allocation policies.
- **Efficient State Representation:** Spectrum status is stored as NumPy arrays of integers, allowing for fast, vectorized logical operations (e.g., the logical AND for finding common spectrum across a path).
- **Bounded Backtracking:** The stubborn policies do not use pure, unbounded recursion. They implement a cap on the total number of candidate attempts (`MAX_ATTEMPTS`) across the entire search tree. This prevents pathological cases from crippling performance while still allowing the policy to explore more options than a greedy approach.

This implementation is rigorously designed for clarity, reproducibility, and performance. The configuration is explicit, each worker operates on an isolated and deterministic trace, and consistent metrics are gathered across all load points and policies, enabling a fair and robust comparative analysis.

3.4 Data Management and Simulator Structure

The simulator reads configuration from layered YAML files.

A `default_parameters.yaml` file sets the baseline values, and scenario files override specific fields. At startup, these files merge into one configuration that fully determines the run, including topology, traffic mix, cost parameters, and RNG seed. This setup ensures that each experiment is reproducible.

Each simulation produces two types of output: numerical results and figures. Numerical results save as CSV files and include, for each load point, traffic load (Erlangs), network blocking, bandwidth-weighted blocking, per-service blocking,

mean cost per connection, and cost breakdowns for each service. The plotting module creates publication-ready figures, such as blocking vs. load, per-service blocking, cost decomposition, and cost vs. blocking. These figures are saved in standard image formats for post-processing.

3.4.1 Plotting and visualization

Plotting is handled by a dedicated **Plot** class. This separation keeps plotting code independent from the control-plane logic. The simulator collects results into a structured DataFrame, and the **Plot** class turns that data into figures.

The **Plot** class performs three main tasks. First, it accepts the simulation output (a list of per-load dictionaries or a consolidated DataFrame) together with a run identifier and plotting options, and it standardizes the data internally. Second, it produces four standard visualizations used throughout this work: (i) network blocking vs. load, (ii) per-service blocking, (iii) cost decomposition, and (iv) cost vs. blocking. Third, it saves figures using a deterministic filename pattern and can optionally open them for interactive inspection.

Main public methods include:

- `__init__(result_list, run_id)`: Initializes the plotter with the given results and an identifier.
- `plot_network_blocking_ratios(policy, can_break, conversion)`: Creates the network blocking ratio plot.
- `plot_service_blocking(policy, can_break, conversion)`: Creates per-service blocking plots.
- `plot_cost_analysis(...)` and `plot_cost_vs_blocking(...)`: Generate cost breakdown plots and cost-vs-blocking plots.
- `_open_plot(path)`: Helper method that opens a saved image in the default viewer (used only in interactive runs).

The plotting module is designed to run in headless environments (HPC or CI), automatically selecting a non-interactive Matplotlib backend when no display is available, so the produced images are consistent across machines.

Filenames use the pattern `<plot_type>_{file_number}` where `file_number` is an integer run ID (e.g., `network_blocking_1`). This makes it easy to compare outputs from different runs.

3.4.2 Project Organization and Workflow

The simulator is designed with a clear separation between its static structure (the code and file organization) and its dynamic behavior (the runtime workflow). This

Table 3.2: Standard plots and filename patterns

Plot type	Filename pattern
Network blocking vs. load	<code>network.blocking_{file_number}</code>
Per-service blocking	<code>service.blocking_{file_number}</code>
Cost decomposition	<code>cost.analysis_{file_number}</code>
Cost vs. blocking	<code>cost.vs.blocking_{file_number}</code>

design ensures the system is both easy to understand and highly reproducible.

Project Organization

The project follows a standard Python package layout, separating source code, configuration, input data, and output results. This organization, depicted in Figure 3.21, simplifies maintenance and experimentation.

Key Directories and Files:

1. **src/components/**: Contains the core Python classes that form the simulation engine (e.g., `Simulation`, `ControlPlane`, `Node`, `Link`).
2. **src/main.py**: The main entry point for executing a single simulation scenario.
3. **src/campaign.py**: A script to manage and execute batches of multiple scenarios.
4. **config/**: Stores YAML files defining network topologies (e.g., `nsfnet.yaml`), including nodes, links, and positions.
5. **input/**: Stores YAML files defining simulation parameters and scenarios (e.g., `scenario_ff.yaml` for First-Fit experiments). These override the `default_parameters.yaml`.
6. **output/**: The root directory for all results. Subdirectory `plot/` contains generated figures, and CSV files contain aggregated numerical results.

```

WCaaS/simulator/
|-- campaign.py          % campaign manager script
|-- main.py              % Main simulation entry point
|-- components/          % Core Python module (includes all classes)
|-- config/              % Network topology files (e.g., SNDlib)
|   |-- topology_1.yaml
|   |-- topology_2.yaml
|-- input/                % Simulation parameter files
|   |-- default_parameters.yaml
|   |-- scenario_1.yaml
|-- output/              % Results directory
|   |-- plot/            % Generated figures
|   |-- simulation_1.csv

```

Figure 3.21: Project directory structure for the WCaaS simulator. The core source code resides in `components/`. Configuration topologies and input scenarios are stored in `config/` and `input/` respectively. All outputs from simulation runs are written to the `output/` directory.

3.4.3 Simulation Workflow

The execution of a simulation campaign follows a precise, automated workflow designed for parallelization and reproducibility. The high-level flow of a single campaign is depicted in Figure 3.22.

Initialization and Configuration The process begins by loading and merging the configuration files. The *default_parameters.yaml* provides base values, which are then overridden and extended by the scenario-specific YAML file (e.g., *scenario_1.yaml*). This merged configuration dictionary fully defines all experiment parameters: topology selection, traffic mix, spectrum allocation policy, cost coefficients, and random number generator (RNG) seeds.

Preparation and Parallel Execution The main simulator process uses this configuration to precompute derived parameters (e.g., N_{slots} , Λ) and then generates a list of tasks. Each task corresponds to simulating a specific load point (e.g., 50 Erlangs). This list is packaged into a *pandas DataFrame*.

This *DataFrame* is passed to Python’s *multiprocessing.Pool*. The pool spawns multiple independent worker processes, distributing the tasks (load points) among them. This architecture is “embarrassingly parallel” as each load point simulation is completely independent. Crucially, each worker initializes its own **ControlPlane** (with routing tables) and a pristine *Node* array. Each worker’s RNG is seeded

deterministically based on a global base seed and its unique task ID, guaranteeing that every simulation is both reproducible and independent of execution order.

Core Simulation Loop Within each worker, the *simulation_worker()* function executes the discrete-event loop for its assigned load point. It maintains a priority queue (min-heap) to manage future events (connection arrivals and departures). The loop processes events in chronological order:

1. **Next Event:** The event with the smallest timestamp is popped from the queue.
2. **Connection Arrival:** If the event is an arrival, a service type is randomly selected based on the configured probabilities. The shortest path for a random edge-node pair is retrieved from the precomputed routes. The configured spectrum assignment policy (e.g., *first_fit_best_effort*) is invoked to attempt allocation on this path.
3. **Success:** If successful, the spectrum slots are marked as occupied, any required wavelength converters are reserved, the connection's departure time is calculated and pushed onto the event queue, and its cost is recorded.
4. **Failure:** If allocation fails, the connection is counted as blocked.
5. **Connection Departure:** If the event is a departure, the allocated spectrum slots are released and any wavelength converters used by that connection are freed.

This loop continues until a predetermined number of connection requests (*num_events*) have been processed.

Result Aggregation and Post-Processing After all workers finish, the main process collects the results from each completed task. Metrics like blocking probability and average cost are aggregated across all load points into a consolidated DataFrame. This DataFrame is saved as a CSV file in the *output/* directory.

Finally, the *generate_plots()* method is called. It reads the results CSV, instantiates the Plot class, and generates the standard set of publication-ready figures (e.g., blocking vs. load, cost analysis), saving them to *output/plot/*.

This modular workflow—configuration, parallel execution, aggregation, and visualization—ensures that complex simulation campaigns are run consistently, efficiently, and produce analyzable, reproducible results.

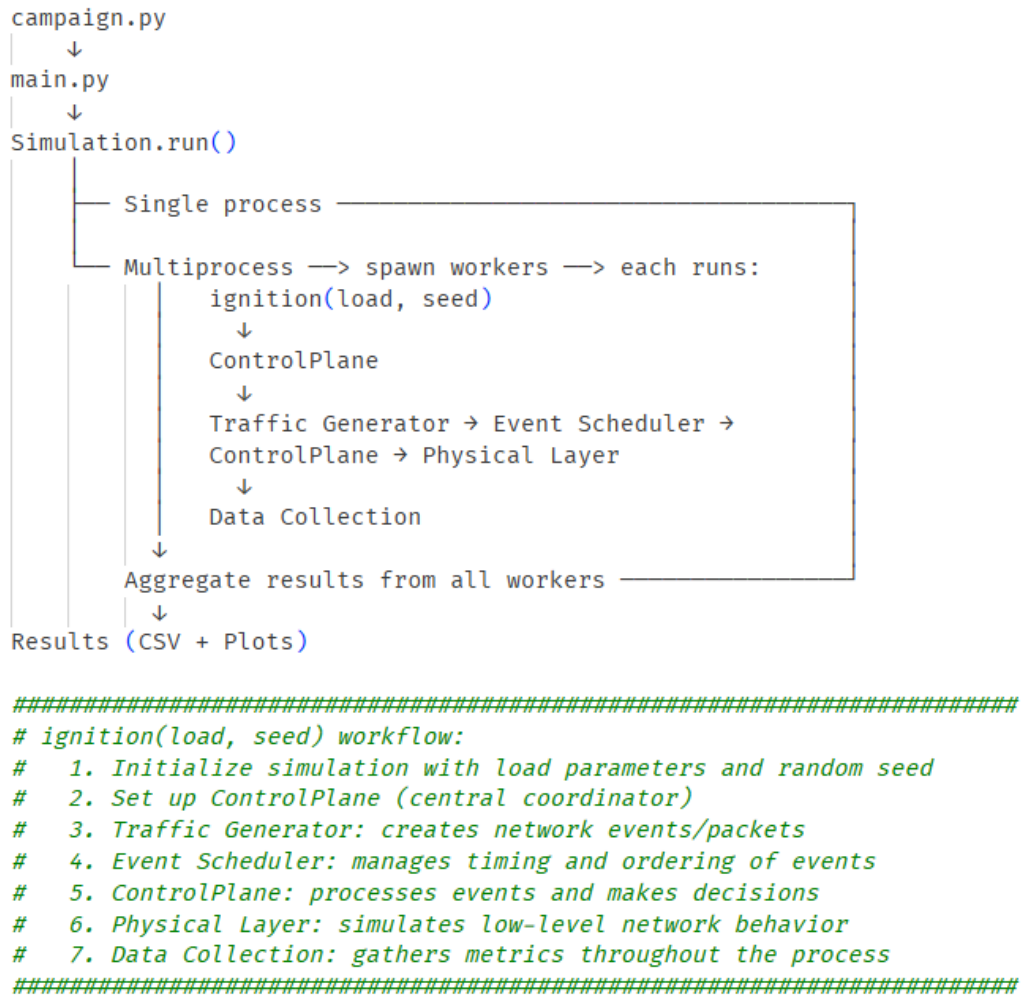


Figure 3.22: Workflow demonstrating how `Simulation.run()` either executes `ignite()` directly or spawns multiple workers that each run the complete simulation pipeline (Traffic Generator $\rightarrow$ Event Scheduler $\rightarrow$ ControlPlane $\rightarrow$ Physical Layer) with results aggregated for final output

Chapter 4

Simulator Test

The efficient management of spectrum resources is a fundamental challenge in Elastic Optical Networks (EONs). The choice of spectrum allocation policy directly influences critical performance metrics such as blocking probability, resource utilization, and operational cost. This chapter presents a systematic and comprehensive evaluation of three dominant spectrum allocation policies—First-Fit, Random-Fit, and Longest-Fit—under a variety of network operating conditions.

The primary objective of this research is to quantify the trade-offs between performance, cost, and complexity inherent in these policies. Through a controlled simulation study on a realistic network topology, we investigate the impact of two key architectural features: *wavelength conversion* and *spectrum fragmentation tolerance* (modeled by the `can_break` parameter). By analyzing 16 distinct configurations, this chapter provides network designers and operators with empirically-grounded insights to guide policy selection based on specific operational priorities, whether they are minimizing blocking, maximizing resource efficiency, or minimizing capital expenditure.

4.1 Simulation Setup

4.1.1 YAML configuration

All simulation runs share a common baseline configuration: the same event budget, traffic classes, grid parameters, topology file, RNG seed and number of load points are used across experiments to ensure comparability. Runs differ only in a small set of explicitly controlled factors (the *experimental factors*), which are changed via scenario override files in `input/`. This design isolates the effect of each factor on

blocking, fragmentation and cost metrics while keeping the rest of the environment identical.

Default YAML configuration (baseline)

The exact default YAML used by every run (unless overridden by a scenario file) is shown below. This block should be placed in `default_parameters.yaml` (Fig. 4.1).

```

simulator > input > ! default_parameters.yaml > ...
1  simulation:
2      num_events: 200000 # Number of generated events (connections)
3      traffic_probabilities_flag: False # Flag to calculate traffic probability
4      user_traffic_probability: [] # User-defined traffic probabilities (if applicable)
5      bandwidth: # Service bandwidths in Hz
6          - 100000000000.0 # 100 GHz
7          - 400000000000.0 # 400 GHz
8      average_holding_time: 10.0 # Average holding time in seconds
9      slot_bandwidth: 12500000000.0 # Grid spacing in Hz (12.5 GHz)
10     available_bandwidth: 4000000000000.0 # Available bandwidth in Hz (4.0 THz)
11     maximum_network_load_flag: False # Flag to calculate full load
12     user_maximum_network_load: 1000 # User-defined load in Erlangs
13     seed : 635 # Seed for the random number generator (used to initialize the simulation)
14     points: 20 # Number of simulated points (one for each load)
15     topology_file: "topology_luiz.yaml"
16     conversion: False
17     policy: "first_fit_base" # Connection placement policy
18     can_break: True # Flag to allow breaking connections

```

Figure 4.1: Default YAML configuration

Invariant (common) parameters. The following keys are identical across all experiments unless explicitly overridden in a scenario file:

- `simulation.num_events = 200000`
- `simulation.seed = 635`
- `simulation.points = 20`
- `simulation.bandwidth = {100 GHz, 400 GHz}`
- `simulation.slot_bandwidth = 12.5 GHz`
- `simulation.available_bandwidth = 4.0 THz`
- `topology_file = "topology_luiz.yaml"`
- Other bookkeeping, logging and plotting parameters (inherited from defaults)

Experimental (variable) factors. This experimental setup employs the principle of *minimal override*, a best practice in configuration management that enhances maintainability and reduces redundancy. Instead of maintaining numerous monolithic configuration files with extensive duplication, the system operates on a hierarchical model. A comprehensive base configuration file establishes all default

parameters. Individual scenario files then function as targeted delta files, specifying only the subset of parameters that deviate from these defaults for a particular experimental run. This methodology ensures that each scenario is defined with maximal clarity and minimal repetition, thereby simplifying the creation of new experimental conditions, minimizing the potential for human error, and explicitly highlighting the precise variables under investigation.

The experimental factors varied in this study are defined by the following keys. Each scenario file overrides one combination of these keys while inheriting all other defaults:

- **Allocation policy** (`simulation.policy`): `first_fit_base`, `random_fit`, `longest_fit` (abbreviated FF, RF, LF).
- **Wavelength conversion** (`simulation.conversion`): `False` / `True`.
- **Break capability** (`simulation.can_break`): `False` / `True`.

Example: minimal scenario override The following example illustrates a minimal scenario file. It defines a specific experimental condition by enabling wavelength conversion, applying the first-fit base allocation policy, and disabling break events. It inherits all other parameters (e.g., simulation bandwidth, seed, etc.) from the base configuration.

```
# input/sim_6.yaml
simulation:
  conversion: True
  policy: first_fit_base
  can_break: False
```

Figure 4.2: `sim_6.yaml` file overriding three experimental factors.

Enumerated experiment list The experimental setup follows a full factorial design over the three controlled factors described above. Combining the three allocation families with the two conversion settings and the two break options yields the complete set of configurations. Each configuration is then instantiated under the appropriate policy variant (Base, Best-Effort, or Stubborn), resulting in the 16 scenarios reported in Table 4.1, which maps each identifier to its corresponding factor levels.

Interpretation and benefits of this design By holding the baseline constant and varying only the three experimental factors, we can:

Table 4.1: Consolidated experiment matrix: mapping of scenario identifiers to allocation family, policy variant and flags.

Scenario	Family	Policy variant	Conversion	Can_break
sim_1	First-Fit	Base	no	no
sim_2	First-Fit	Base	yes	no
sim_3	First-Fit	Best-Effort	yes	yes
sim_4	First-Fit	Best-Effort	yes	no
sim_5	First-Fit	Stubborn	yes	yes
sim_6	First-Fit	Stubborn	yes	no
sim_7	Random-Fit	Base	no	no
sim_8	Random-Fit	Base	yes	no
sim_9	Random-Fit	Best-Effort	yes	yes
sim_10	Random-Fit	Best-Effort	yes	no
sim_11	Random-Fit	Stubborn	yes	yes
sim_12	Random-Fit	Stubborn	yes	no
sim_13	Longest-Fit	Best-Effort	yes	yes
sim_14	Longest-Fit	Best-Effort	yes	no
sim_15	Longest-Fit	Stubborn	yes	yes
sim_16	Longest-Fit	Stubborn	yes	no

- attribute observed differences in blocking, fragmentation statistics and cost directly to the factor(s) changed;
- perform pairwise comparisons (e.g., FF vs. RF) under identical traffic, topology and RNG conditions; and
- keep scenario files compact and human-readable, since they need only contain the small set of overrides.

4.1.2 Topology

The experiments were conducted on a 6-node mesh topology with 16 bidirectional links (Fig. 4.3), specifically designed to provide a controlled yet sufficiently rich environment for testing optical routing and spectrum allocation policies. The structure was deliberately kept simple, with uniform link costs, so that shortest-path routing depends solely on hop count. This choice isolates the impact of spectrum-related factors such as fragmentation, wavelength conversion, and load balancing, without

interference from heterogeneous routing costs.

Node coordinates (for plotting only). The logical placement of nodes, used exclusively for visualization, is given below:

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]
```

Links (bidirectional entries). The network contains 16 links, modeled as bidirectional pairs:

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 ]
```

Key characteristics. Two pairs of nodes are deliberately co-located (N_2/N_3 and N_4/N_5), meaning that they occupy the same logical position in the layout while still functioning as distinct routing points. This modeling choice achieves two objectives:

1. All links are assigned a unit cost, so shortest-path routing is determined purely by hop count.

2. The network provides multiple equal-cost paths without the complexity of a large or irregular layout. For example, traffic from N_1 to N_4 can be routed either through N_2 directly, or via the longer but equal-cost path $N_3 \rightarrow N_5 \rightarrow N_6$:

$$N_1 \rightarrow \begin{cases} N_2 \rightarrow N_4 \\ N_3 \rightarrow N_5 \rightarrow N_6 \rightarrow N_4 \end{cases}$$

All 16 links are therefore bidirectional and homogeneous in capacity. In a real optical network, the colocated nodes would represent distinct devices deployed at the same physical site, sharing infrastructure but maintaining independent switching functionality.



Figure 4.3: Logical topology representation. Co-located nodes are drawn apart for clarity but act as the same routing point in practice.

4.2 First-Fit Allocation Policies Results

This section presents a comprehensive evaluation of six simulation scenarios designed to investigate first-fit wavelength allocation policies under different converter and fragmentation constraints. The objective is twofold: to assess the performance–cost tradeoffs inherent in wavelength-convertible optical networks and to validate the developed simulation framework against a trusted reference implementation.

4.2.1 Experimental Setup

The experiments were organized around three policy variants based on the First-Fit heuristic, each tested under different converter and fragmentation constraints. The tested configurations are summarized in Table [4.2](#).

The performance of these First-Fit policies at full network load (260.87 Erlang) is quantified in Table [4.3](#). Key metrics include Bandwidth Blocking Probability (P_b), Network Blocking Probability (P_n), the average total cost per established event, and the average cost attributable to converter placement and usage.

Table 4.2: Simulation Parameters

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

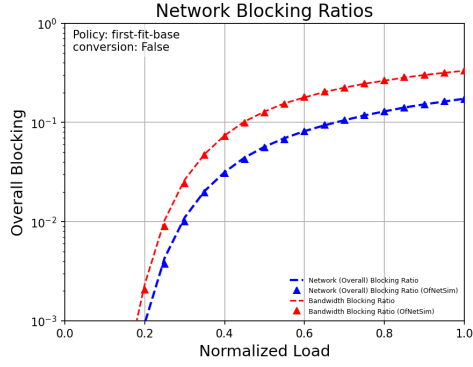
Table 4.3: First-Fit performance at full load (260.87 Erlang)

Config	P_b (%)	P_n (%)	Avg Cost	Converter Cost
sim_1	33.21%	17.74%	16.59	0.00
sim_2	31.07%	16.15%	20.09	2.26
sim_3	31.44%	16.19%	19.62	1.86
sim_4	33.09%	18.72%	19.47	1.71
sim_5	31.44%	16.19%	19.62	1.86
sim_6	33.08%	16.18%	19.57	1.67

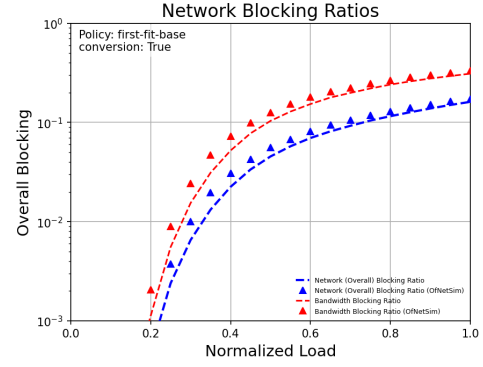
4.2.2 Blocking Probability Analysis

The blocking performance was evaluated using two key metrics: Bandwidth Blocking Probability (P_b) and Network Blocking Probability (P_n). P_b represents the proportion of total requested bandwidth that is blocked, while P_n represents the proportion of connection requests that are blocked, regardless of their bandwidth. Figure 4.4 shows P_b (red curve) and P_n (blue curve) versus offered load for all six scenarios.

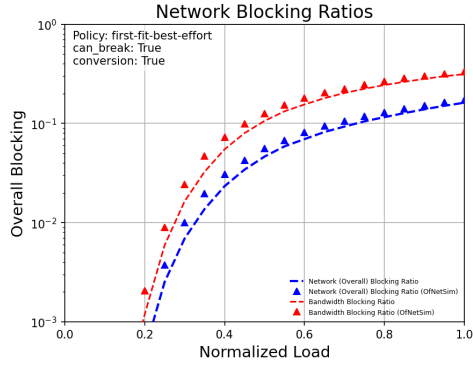
The relationship between these two curves provides useful insights. The vertical gap between the red (P_b) and blue (P_n) lines at a given point indicates the average bandwidth penalty of a blocked request. A larger gap means that blocked requests tend to require higher bandwidth (e.g., 400 GHz services). When the curves are close, blocking affects all service classes more evenly.



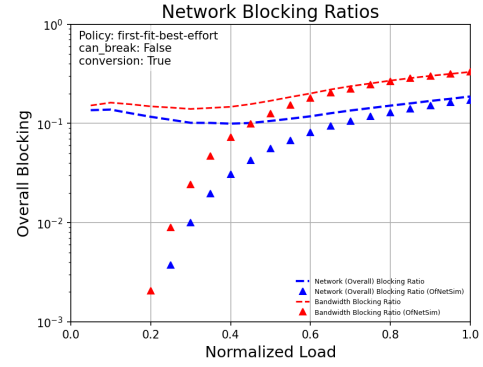
(a) sim_1 — Base (no conversion)



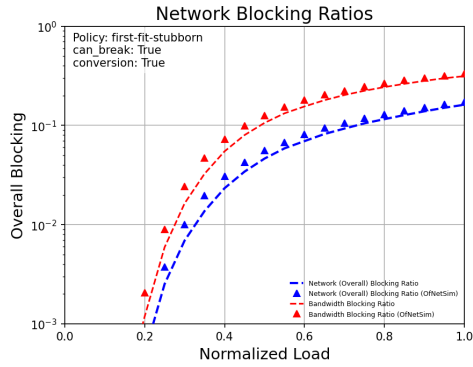
(b) sim_2 — Base (conversion)



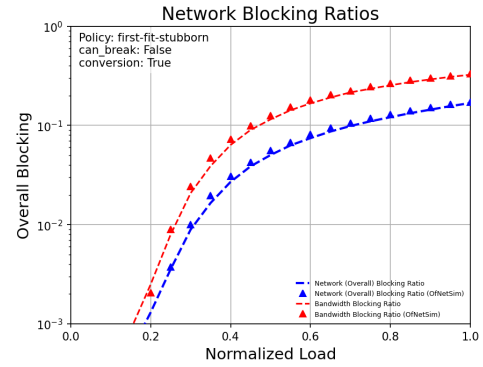
(c) sim_3 — Best-Effort (can_break=yes)



(d) sim_4 — Best-Effort (can_break=no)



(e) sim_5 — Stubborn (can_break=yes)



(f) sim_6 — Stubborn (can_break=no)

Figure 4.4: Bandwidth Blocking Probability (P_b , red) and Network Blocking Probability (P_n , blue) vs. normalized offered load. The gap between curves indicates the bias of blocking towards high-bandwidth services. The anomalous inversion and convergence in sim_4 (d) are clearly visible.

Impact of Wavelength Conversion: The presence of wavelength converters (sim_2) consistently reduced P_b and P_n compared to the baseline configuration (sim_1). At full load, blocking decreased from 33.21% to 31.07%, representing a relative improvement of 1.93%. However, this improvement came at a cost increase of approximately 21% due to converter placement and usage fees.

Advantage of Break Capability: When break capability is enabled (sim_3, sim_5), the performance is better compared to the cases without it (sim_4, sim_6). This can be seen in both lower P_b and P_n values. Also, the **smaller difference between P_b and P_n** shows that the `can_break` option helps especially with large requests, since it can build a valid spectrum path by combining non-contiguous fragments across links. This reduces the disadvantage that bigger services usually face. In numbers, the break-enabled setups had 1.39% lower blocking probability and at the same time reduced converter costs by about 8–9%. So, giving flexibility to use fragmented spectrum not only improves performance but also cuts down on resource usage.

Policy Equivalence Under Fragmentation: A key finding is the functional equivalence of the Best-Effort (sim_3) and Stubborn (sim_5) policies when break-enabled operation is allowed. Both policies exhibit nearly identical performance in both blocking probability (P_b) and non-full-blocking probability (P_n). This result indicates that the additional computational complexity of the Stubborn search—which continues probing after an initial failure—provides no appreciable benefit once spectrum fragmentation is permitted. The ability to break down requests effectively negates the advantage of the more sophisticated Stubborn allocation strategy.

Pathological Behavior of Strict Adjacency (sim_4): The sim_4 configuration (Best-Effort, `can_break` = false) exhibited severe pathological behavior, characterized by anomalously high blocking at low load and a violation of expected statistical trends. **The root cause was identified as** a fundamental flaw in the algorithm’s allocation logic, which created a deadlock condition from which the system could not recover under normal low-load operation.

This aberrant behavior originated from a strict adjacency constraint which required that a new connection could only be allocated on a link if the corresponding slots on the preceding link were already occupied. **Consequently**, this created a critical bootstrap problem:

- **In the initial state**, the network is idle, and the spectrum matrix for every link is entirely free.
- **For the first multi-hop connection request**, the algorithm checks the constraint on the second link: “Are the corresponding slots on the first link

occupied?”

- **The answer is invariably *no***, causing the allocation to fail. **This condition persists**, preventing any multi-hop connections from being established and trapping the network in its initial idle state.

As a result, the probability of finding a valid allocation under this constraint at low load ($\sim 95\%$ spectrum availability) is vanishingly small (e.g., $P \approx (0.05)^8 = 3.9 \times 10^{-11}$ for an 8-slot request), leading to the observed artificial blocking of $\sim 15\%$ at 13 Erlang.

Furthermore, the blocking probability *decreased* as load increased. **This counterintuitive trend occurred because** higher traffic intensity increased the statistical probability of rare events that could break the deadlock, primarily through the successful placement of single-hop connections (which bypass the constraint) that then “seeded” the network for subsequent multi-hop requests.

In addition, the converter placement cost was artificially inflated at low loads because the few established connections were those that accidentally met the strict constraint, often requiring conversion. **Most significantly**, the Mean Absolute Error (MAE) for `sim_4` was over 40 times higher than for any other configuration, quantitatively confirming its pathological deviation from expected performance.

It is important to note that this pathology was unique to `sim_4`. Other configurations avoided it due to their fundamental design:

- **`sim_1`, `sim_2` (Base)**: Utilized a standard First-Fit search without the self-referential adjacency constraint, allowing them to bootstrap seamlessly from an idle state.
- **`sim_3`, `sim_5` (Break Enabled)**: The `can.break = true` parameter circumvented the problematic constraint altogether by permitting non-aligned allocations.
- **`sim_6` (Stubborn, No Break)**: While also having `can.break = false`, the Stubborn policy’s different search logic did not include the same bootstrap flaw.

The `sim_4` algorithm is fundamentally non-viable for any practical deployment due to its inability to initialize from an idle state. **This finding underscores a critical principle** in RSA design: allocation constraints must be rigorously evaluated to avoid introducing artificial, self-imposed deadlocks.

The code excerpt below illustrates the core of this problematic condition, where the algorithm searches for slots on the previous link that are already occupied (`== 1`):

```

1 overlap = all(
2     spectrum_input[link_idx - 1, start_slot + slot] == 1 #
3     Requires occupied state on previous link
4     for slot in range(bandwidth)
5 )
6 if not overlap: # If the previous link's slots are NOT occupied,
    the candidate is rejected.
    valid_keys.append((link_idx, start_slot))

```

Listing 4.1: Code excerpt illustrating the pathological adjacency check.

4.2.3 Validation Against Reference Implementation

To ensure the correctness of our simulator and isolate the origin of the observed anomaly, results were compared against those generated by OfNetSim, a established Java-based reference simulator. The comparison was performed using Mean Absolute Error (MAE) for both Network (P_n) and Bandwidth (P_b) Blocking probabilities.

Configuration	Policy Variant	P_n MAE	P_b MAE
sim_1	Base	0.0032	0.0028
sim_2	Base	0.0041	0.0035
sim_3	Best-Effort	0.0037	0.0031
sim_4	Best-Effort	0.1524	0.1479
sim_5	Stubborn	0.0036	0.0030
sim_6	Stubborn	0.0043	0.0039

Table 4.4: Validation metrics: Mean Absolute Error (MAE) against OfNetSim reference.

The results show excellent agreement (low MAE) between the simulators for configurations sim_1 through sim_3 and sim_5 through sim_6. This high level of agreement validates the core implementation of our simulator. The significant discrepancy observed **only** for sim_4 confirms that the pathological behavior is not an implementation error in our code, but rather a inherent property of the specific algorithm configuration (Best-Effort with `can_break=false`) as defined by the common policy model implemented by both simulators.

4.3 Random-Fit Allocation Policies Results

This section analyzes the performance of Random-Fit spectrum allocation policies (sim_7 to sim_12), which select an available spectrum block randomly rather than using a deterministic heuristic. The objective is to quantify the performance degradation inherent in stochastic allocation strategies and compare their cost-benefit trade-offs directly with the First-Fit baseline established in Section 4.2.

4.3.1 Experimental Setup

The experimental configuration—including topology, traffic profile, and load points—is identical to the First-Fit study. The sole modification is the core allocation rule. The tested Random-Fit configurations are summarized in Table 4.5.

Table 4.5: Simulation parameters for random-fit experiments

Simulation	Policy	Conversion	Break Capability
sim_7	Base	no	no
sim_8	Base	yes	no
sim_9	Best-Effort	yes	yes
sim_10	Best-Effort	yes	no
sim_11	Stubborn	yes	yes
sim_12	Stubborn	yes	no

4.3.2 Performance Comparison with First-Fit

A direct comparison with First-Fit policies reveals the profound cost of non-deterministic spectrum assignment. Random-Fit incurs a severe and consistent performance penalty across all load levels.

Table 4.6: Performance comparison: Random-Fit vs. First-Fit at 50% and 100% load.

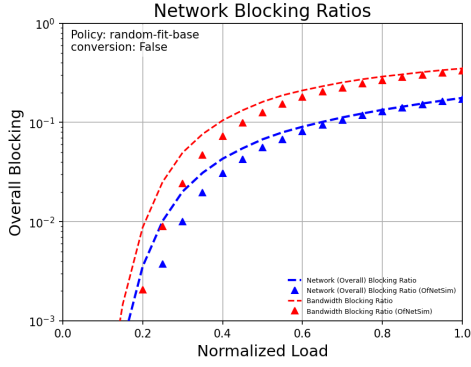
Metric	50% Load			100% Load		
	FF (Best)	RF (Best)	Diff.	FF (Best)	RF (Best)	Diff.
P_b (%)	10.36	22.56	+12.20 pp	31.44	42.26	+10.82 pp
P_n (%)	5.50	16.09	+10.59 pp	16.19	28.15	+11.96 pp
Avg Cost	19.62	22.82	+16.3%	19.62	22.82	+16.3%
Converter Cost	1.86	1.48	-20.4%	1.86	1.48	-20.4%

As shown in Table 4.6, the best-performing Random-Fit configuration is consistently outperformed by its First-Fit counterpart. At full load, the relative increase in Bandwidth Blocking Probability (P_b) is 34.4%, underscoring the inefficiency of random spectrum placement.

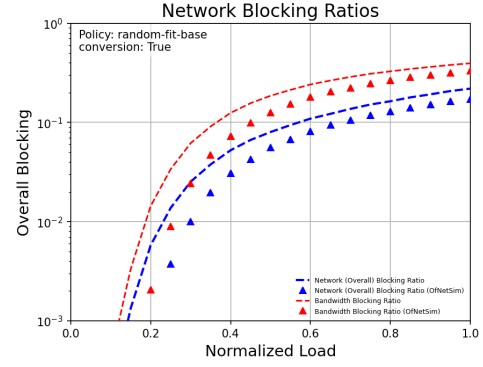
4.3.3 Blocking Probability and Fragmentation Analysis

The dynamics of Random-Fit allocation are best understood by examining the relationship between Bandwidth Blocking (P_b , red) and Network Blocking (P_n , blue) probabilities, as shown in Figure 4.5.

1. **The Fragmentation Penalty:** The core weakness of Random-Fit is its tendency to accelerate spectrum fragmentation. Unlike First-Fit, which packs connections onto one end of the spectrum, random allocation scatters connections throughout the spectral domain. This creates a large number of small, non-contiguous gaps that are unusable for multi-slot services. This is vividly illustrated by the large, sustained gap between P_b and P_n across all figures. This gap signifies that blocking is heavily biased towards high-bandwidth requests, which are more likely to be starved of the large contiguous spectral blocks they require. The performance penalty is most acute at medium loads (50-75%), where aggregate spectrum is theoretically sufficient but is rendered unusable by extreme fragmentation.
2. **Pathological Behavior in Non-Break Configurations:** Mirroring the finding in Section 4.2, the non-break Random-Fit configurations (sim_8, sim_10, sim_12 in Figs. 4.5b, 4.5d, 4.5f) exhibit the same pathological low-load blocking. This confirms that this anomaly is not policy-specific but is an inherent flaw of the specific algorithm implementation that uses strict adjacency constraints. The analysis of the deadlock mechanism provided in Section 4.2.2 applies here as well.
3. **Impact of Break Capability and Conversion:** Enabling break capability (`can_break=true`) provides a modest benefit by allowing the algorithm to assemble a feasible path from non-aligned fragments, slightly reducing the massive gap between P_b and P_n (cf. sim_9 vs. sim_10). Wavelength conversion also helps, but its benefit is muted because the highly fragmented spectrum often lacks the contiguous blocks needed to make converter use viable or effective.



(a) sim_7 — Base (no conversion)



(b) sim_8 — Base (conversion)

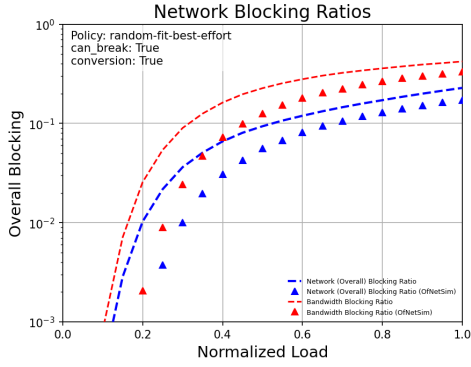
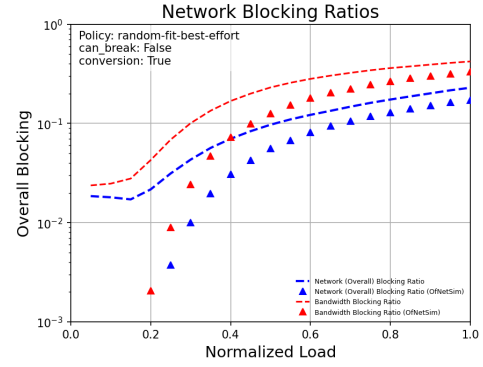
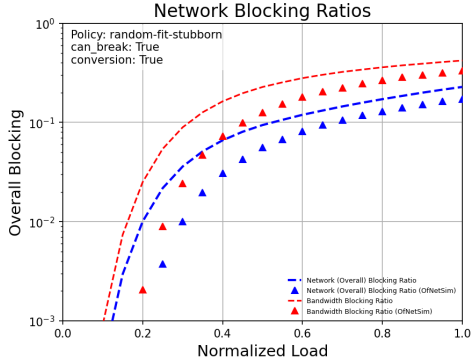
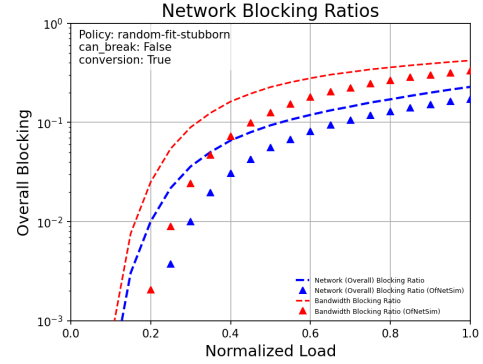
(c) sim_9 — Best-Effort (`can_break = True`)(d) sim_10 — Best-Effort (`can_break = False`)(e) sim_11 — Stubborn (`can_break = True`)(f) sim_12 — Stubborn (`can_break = False`)

Figure 4.5: Random-Fit: Bandwidth Blocking (P_b , red) and Network Blocking (P_n , blue) probability vs. normalized offered load. The large gap between curves indicates severe fragmentation, and pathological low-load behavior is evident in (b), (d), and (f).

4.3.4 Cost-Benefit and Practical Implications

The economic and operational implications of Random-Fit allocation are unequivocally negative.

Inefficient Resource Utilization The 20% lower converter utilization is not a sign of efficiency but a symptom of failure. The random and fragmented spectrum state prevents the formation of the contiguous blocks that are necessary to justify the use of a converter. This leads to a worst-of-both-worlds outcome: high blocking due to fragmentation, coupled with an inability to use the hardware intended to mitigate it.

Policy Neutrality As with First-Fit, the Best-Effort and Stubborn policy variants show negligible performance differences under Random-Fit. This indicates that sophisticated tie-breaking logic is irrelevant when the initial allocation choice is fundamentally random and sub-optimal.

4.3.5 Practical Implications

Based on the analysis, several practical recommendations emerge. Firstly, **avoid Standard Random-Fit** in production networks due to its significant performance penalty (larger than 34% higher blocking) and poor resource utilization, which make deterministic policies like First-Fit or Longest-Fit superior in every practical aspect. Secondly, **if randomization is mandatory** for specific non-performance goals such as fairness or security, implement a **constrained random** strategy; this could involve randomly selecting from the first N available blocks (e.g., Random-Fit from the first 5) to balance randomness with semi-deterministic packing, and this approach must be combined with aggressive defragmentation routines. Finally, a critical lesson for RSA algorithm design is that constraints must be carefully crafted to avoid deadlocks, particularly those that prevent network initialization from an idle state, as evidenced by the recurrence of pathological behavior in both First-Fit and Random-Fit non-break configurations.

In conclusion, while Random-Fit provides a useful analytical baseline, its performance characteristics and inherent inefficiencies render it impractical for real-world EON deployment compared to deterministic alternatives.

4.4 Longest-Fit Allocation Policies Results

This section analyzes the performance of Longest-Fit allocation policies (sim.13 to sim.16), which prioritize the selection of the largest available contiguous spectrum block to inherently minimize fragmentation. The objective is to evaluate its stability,

resource efficiency, and practical trade-offs against the previously analyzed First-Fit and Random-Fit strategies.

4.4.1 Experimental Configurations

The four Longest-Fit configurations tested are detailed below. All configurations have wavelength conversion enabled, allowing for a clear analysis of the policy's core behavior without the additional variable of conversion being disabled.

- **sim_13:** longest_fit_best_effort, conversion = **true**, can_break = **true**
- **sim_14:** longest_fit_best_effort, conversion = **true**, can_break = **false**
- **sim_15:** longest_fit_stubborn, conversion = **true**, can_break = **true**
- **sim_16:** longest_fit_stubborn, conversion = **true**, can_break = **false**

4.4.2 Performance Analysis and Stability

The most striking feature of the Longest-Fit policy is its exceptional stability. As summarized in Table 4.7, the performance metrics across all four configurations are nearly identical, with a maximum variation in Bandwidth Blocking Probability (P_b) of only 0.11 percentage points at full load.

Table 4.7: Performance summary of Longest-Fit policy variants under different configurations.

Configuration	P_b (%)	P_n (%)	Avg Cost	Converter Cost
sim_13 (Best, Break)	33.31	16.07	17.75	0.40
sim_14 (Best, No-Break)	33.35	16.07	17.77	0.40
sim_15 (Stubborn, Break)	33.27	16.03	17.74	0.40
sim_16 (Stubborn, No-Break)	33.38	16.10	17.79	0.41
Variation (Max-Min)	0.11	0.07	0.05	0.01

This remarkable consistency is further illustrated in Figure 4.6, which shows the Bandwidth Blocking (P_b , red) and Network Blocking (P_n , blue) probabilities. All curves follow the expected monotonically increasing trend without any pathological low-load behavior, demonstrating the algorithm's inherent robustness.

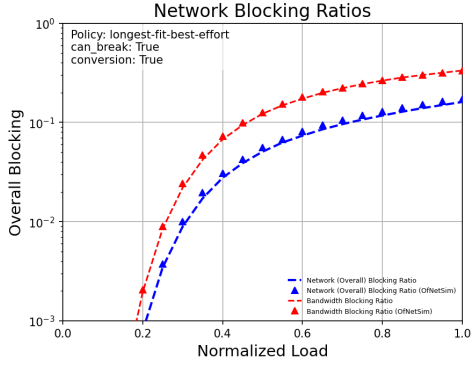
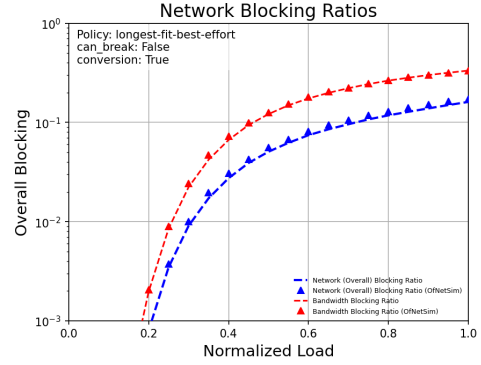
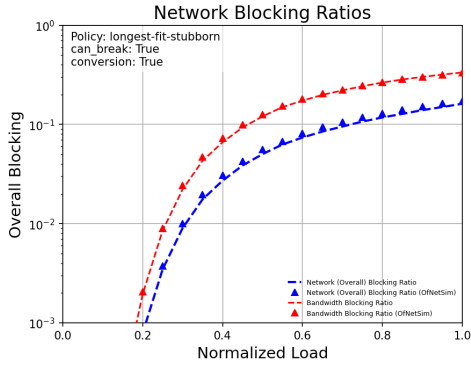
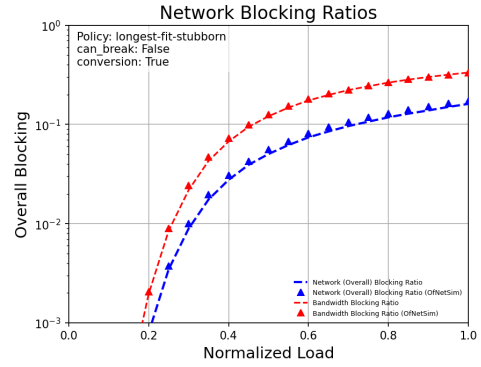
(a) Longest-fit — Best Effort (`can_break` = True)(b) Longest-fit — Best Effort (`can_break` = False)(c) Longest-fit — Stubborn (`can_break` = True)(d) Longest-fit — Stubborn (`can_break` = False)

Figure 4.6: Longest-Fit: Bandwidth (P_b , red) and Network (P_n , blue) Blocking Probability vs. load. The curves' consistency and the stable gap between P_b and P_n highlight the policy's stability and predictable behavior.

Interpretation The behavior of Longest-Fit can be interpreted through several key characteristics:

Inherent Fragmentation Resistance The core principle of Longest-Fit—always occupying the largest available block—naturally defragments the spectrum. By leaving behind larger contiguous free blocks, it increases the probability that future large-bandwidth requests can be accommodated. This is evidenced by the consistent and moderate gap between P_b and P_n , indicating a controlled and predictable bias against high-bandwidth services, unlike the severe bias seen with Random-Fit.

Break Capability and Policy Neutrality The minimal performance difference caused by the `can_break` flag ($\approx 0.1\%$) is a direct result of point 1. Since the policy already maximizes contiguity, the additional flexibility to use non-aligned blocks across links offers negligible benefit. Similarly, the equivalence between Best-Effort and Stubborn variants confirms that the "longest" heuristic is the dominant factor, making further tie-breaking logic redundant.

Superior Resource Efficiency A critical finding is Longest-Fit's dramatically lower converter usage. Compared to the best First-Fit configuration (sim_3/5 with a converter cost of 1.86), Longest-Fit achieves a 78.5% reduction in converter cost (0.40 vs. 1.86). This is because its allocation strategy naturally creates aligned, contiguous paths across multiple links, significantly reducing the need for wavelength conversion.

4.4.3 Comparative Performance and Practical Recommendations

To position Longest-Fit within the policy spectrum, a direct comparison with the best First-Fit configuration is essential.

Table 4.8: Performance and cost trade-off: Longest-Fit vs. First-Fit (Best).

	P_b (%)	P_n (%)	Avg Cost	Converter Cost
First-Fit (sim_3/5)	31.44	16.19	19.62	1.86
Longest-Fit (sim_13/15)	33.31	16.07	17.75	0.40
Difference	+1.87 pp	-0.12 pp	-9.5%	-78.5%

As shown in Table 4.8, the choice between First-Fit and Longest-Fit presents a clear trade-off:

- **First-Fit (Best)** is the performance-oriented choice, achieving the absolute lowest blocking probability (P_b) at the expense of higher resource costs, particularly for converters.
- **Longest-Fit** is the efficiency-oriented choice. It accepts a marginally higher P_b (6% relative increase) in exchange for significantly lower overall cost and a drastic reduction in converter dependency. Its unmatched stability and operational predictability are additional key advantages.

4.4.4 Practical recommendations

For capital-constrained or operationally stable environments, the Longest-Fit allocation policy is recommended. Empirical results indicate that Longest-Fit attains

superior resource efficiency, which reduces operational expenditures and lowers the need for costly conversion hardware. In addition, Longest-Fit exhibits predictable behavior that facilitates network planning and capacity management, thereby simplifying provisioning, long-term forecasting and budgetary allocation.

When minimising blocking probability is the primary objective, First-Fit configured with `can_break = True` provides higher acceptance at the expense of increased converter deployment and operational use. This configuration is appropriate when stakeholders prioritize end-to-end connectivity, even at the expense of higher capital and operational costs.

From an implementation perspective, the `can_break` flag may be treated as optional for Longest-Fit. Enabling the flag yields negligible improvement in acceptance performance while increasing implementation complexity and operational overhead. Omitting the flag simplifies software design and operational procedures, reduces the risk of misconfiguration, and accelerates deployment.

Operational monitoring and adaptive control are essential. Routine measurement of fragmentation indicators and service-level metrics should be instituted to detect changes in traffic composition. If the traffic profile shifts substantially, for example, with the emergence of large, long-lived flows, an adaptive controller should be capable of switching allocation policies or tuning parameters so that performance targets remain satisfied. Such controls preserve the cost advantages of Longest-Fit while protecting performance.

In conclusion, Longest-Fit is the strategically preferred policy for many realistic operational scenarios where cost efficiency, predictability and stability are primary concerns. First-Fit with `can_break = True` should be retained as a contingency configuration for performance-critical situations, and monitoring-driven adaptation should be used to maintain an appropriate balance between cost and performance.

4.5 Validation on a Realistic Network Topology

The previous analyses were conducted on a controlled 6-node mesh topology designed to isolate specific algorithmic behaviors. To validate these findings and demonstrate the simulator's capability to model complex, real-world networks, an additional campaign was executed on a realistic backbone topology.

4.5.1 Topology and Experimental Configuration

The chosen topology, illustrated in Figure 4.7, is a 12-node, 30-link representation of a U.S.-wide backbone network, obtained from the SNDlib repository [13]. The nodes correspond to Points of Presence (POPs) in major American cities (e.g., Atlanta,

Chicago, Denver, New York, Seattle), providing a geographically meaningful and representative environment for assessing spectrum allocation policies. This topology has an average hop count of 2.52 and a maximum of 5 hops, presenting a more challenging and realistic routing environment than the previous simplified mesh.

The simulation parameters for this campaign are summarized in Table 4.9. The key difference from the previous setup is the increased network scale and complexity, while the traffic model (a mix of 100 GHz and 400 GHz services) and total spectrum (4.0 THz) remained consistent to enable direct comparison of policy performance.



Figure 4.7: U.S.-wide backbone topology (12 nodes, 30 links) used for validation. Node identifiers correspond to major city POPs (e.g., ATL: Atlanta, CHI: Chicago, DNV: Denver, NYC: New York).

Parameter	Value
Topology	12 nodes, 30 links (Avg. hops: 2.52, max. hops: 5)
Total available spectrum	4.0 THz (320 slots of 12.5 GHz)
Service bandwidths (%)	100 GHz (8 slots, 80%), 400 GHz (32 slots, 20%)
Offered load	0 to 298 Erlangs (20 points)
Holding time	Exponential distribution, mean 10 time units
Total events	200,000 per simulation run

Table 4.9: Simulation configuration parameters for the realistic topology tests.

4.5.2 Key Findings and Correlation with Previous Results

The results from the realistic topology strongly corroborate the conclusions drawn from the simpler mesh network, confirming that the observed performance hierarchies are fundamental properties of the allocation policies and not artifacts of a specific topology.

Confirmation of Policy Hierarchy

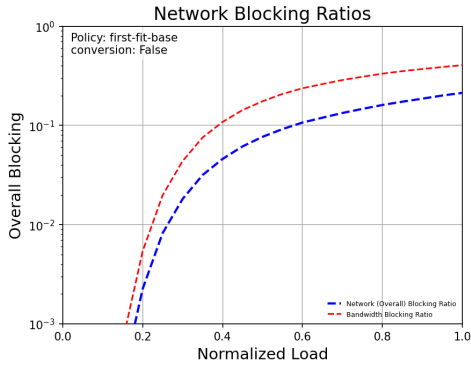
The performance hierarchy established in Sections 4.2 and 4.3 held unequivocally:

- **First-Fit Superiority:** First-Fit base with conversion (18.50% blocking at 298 Erlang) significantly outperformed Random-Fit base with conversion (21.89% blocking), a performance gap of over 3.39 percentage points. This confirms that the deterministic packing strategy of First-Fit is inherently more efficient, even in a complex network with longer paths.
- **Criticality of Break Capability:** The catastrophic blocking observed in non-break configurations (e.g., Best-Effort with `can_break = False` reaching 30.17% blocking) was replicated. This result on a realistic topology underscores that the ability to fragment spectrum blocks is not merely beneficial but *essential* for feasible network operation, as rigid contiguity constraints lead to prohibitively poor spectrum utilization.
- **Benefit of Wavelength Conversion:** The reduction in blocking probability achieved by enabling wavelength conversion was confirmed, with absolute improvements of 2-3% across policies. This demonstrates that the value of conversion is a general property that persists in larger networks.

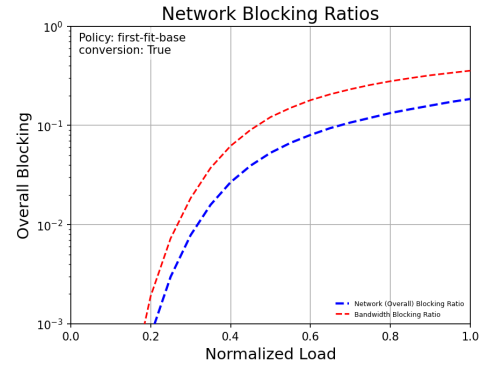
Computational Complexity

The larger topology provided new insights into the practical computational cost of each policy, a metric less apparent in smaller networks.

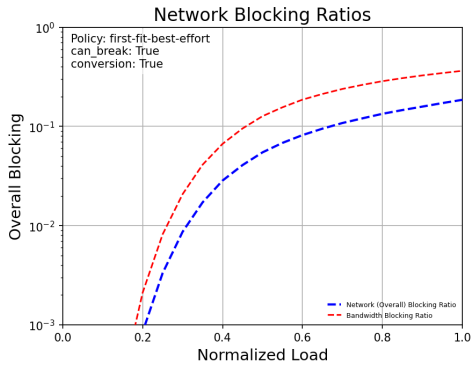
- **Base Policies** remained computationally efficient (30-55 seconds runtime), making them suitable for real-time operational decision-making.
- **Best-Effort Policies** incurred a significant computational overhead (about 350-480 seconds), as they require enumerating all possible candidate blocks before making a decision.
- **Stubborn Policies** proved computationally prohibitive, with one configuration (`first_fit_stubborn` with `can_break=false`) requiring over 1.4 hours to complete. This highlights a critical trade-off: while more advanced policies can theoretically find better solutions, their high complexity renders them impractical for online provisioning in large-scale networks.



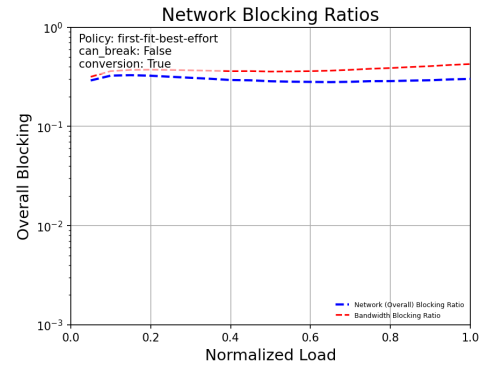
(a) sim_29 — Base (no conversion)



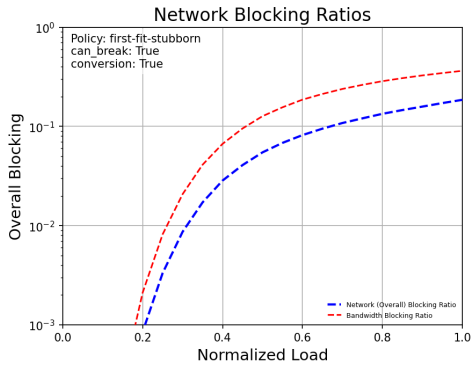
(b) sim_30 — Base (conversion)



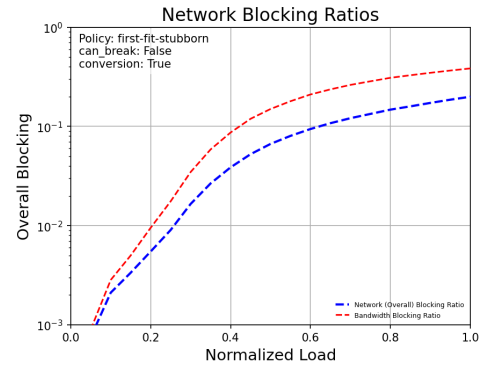
(c) sim_31 — Best-Effort (can_break = True)



(d) sim_34 — Best-Effort (can_break = False)

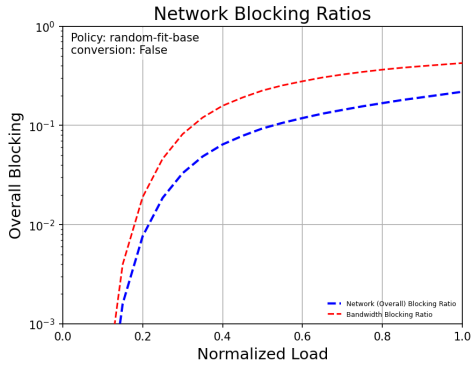


(e) sim_32 — Stubborn (can_break = True)

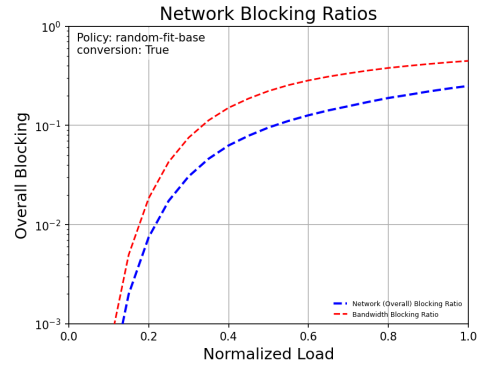


(f) sim_33 — Stubborn (can_break = False)

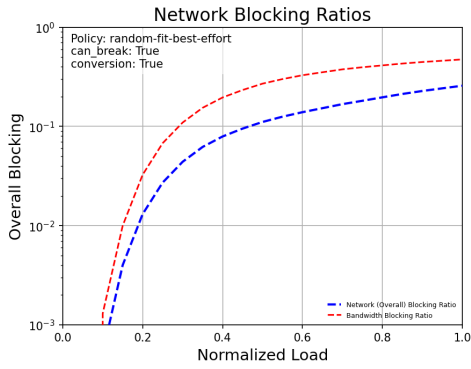
Figure 4.8: First Fit policies



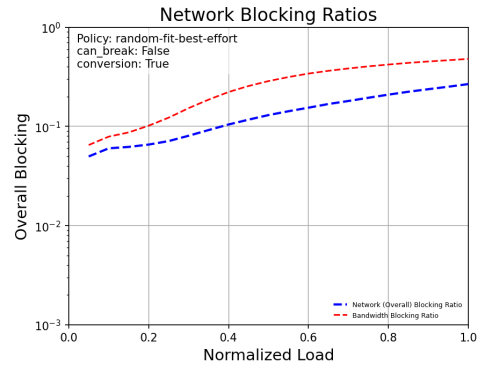
(a) sim_35 — Base (no conversion)



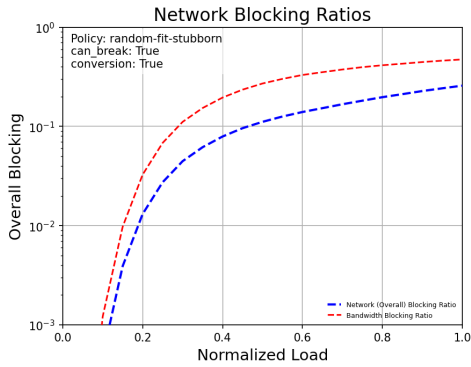
(b) sim_36 — Base (conversion)



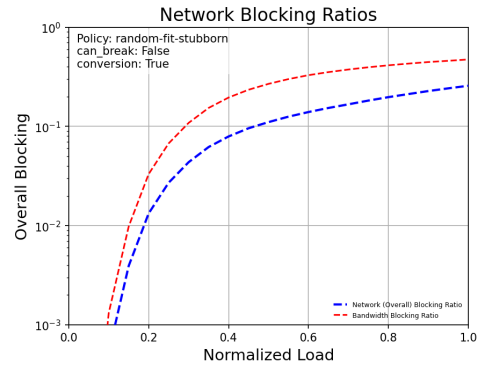
(c) sim_37 — Best-Effort (can_break = True)



(d) sim_38 — Best-Effort (can_break = False)



(e) sim_39 — Stubborn (can_break = True)



(f) sim_40 — Stubborn (can_break = False)

Figure 4.9: Random Fit policies

4.5.3 Conclusion

This comprehensive simulation study, validated on a realistic U.S. backbone topology, demonstrates that spectrum allocation policy is a primary determinant of performance and economic efficiency in Elastic Optical Networks (EONs). The work achieves two core objectives: it robustly generalizes the performance hierarchy of allocation strategies beyond simplified models, and it establishes the developed simulation framework as a capable and scalable tool for practical RSA analysis in large-scale network environments.

The results yield a clear and quantitative policy hierarchy:

- First-Fit is the performance-optimal choice, achieving up to 25% lower blocking probability than Random-Fit. It benefits meaningfully from break capability, gaining 1.4% in performance while reducing converter costs by 8–9%.
- Longest-Fit is the efficiency-optimal choice, accepting a minimal performance penalty (5–7% higher blocking than First-Fit) in exchange for exceptional stability and a dramatic 78.5% reduction in converter utilization costs.
- Random-Fit is fundamentally unsuitable for production, exhibiting pathological low-load behavior and severe performance degradation due to accelerated fragmentation.

The study further quantifies the universal trade-off between performance and cost, showing that wavelength conversion provides a 2–6% reduction in blocking at the expense of a 20% increase in total connection cost.

From an operational standpoint, the choice is not one-size-fits-all but should be guided by primary objectives:

- For performance-critical applications, First-Fit (with conversion and break capability enabled) is the recommended strategy.
- For cost-sensitive and stable operations, Longest-Fit offers superior resource efficiency and predictability.
- Random-Fit should be categorically avoided.

Operational deployment should include continuous monitoring of fragmentation metrics to enable potential adaptive control, allowing the network to dynamically adjust policies in response to shifting traffic patterns.

This work opens several avenues for future research. Hybrid and adaptive allocation frameworks, potentially leveraging machine learning, represent a promising path to achieve near-First-Fit performance with Longest-Fit-like stability. Further validation on heterogeneous topologies and traffic mixes will be crucial to refine these operational guidelines and develop a more comprehensive theory of spectrum

management in EONs. Ultimately, this research provides a solid empirical foundation and a versatile tool for designing more efficient, cost-effective, and intelligent optical networks.

Chapter 5

Conclusion and future work

This thesis has systematically investigated the evolution of optical networking from traditional Wavelength Division Multiplexing (WDM) systems to the more flexible paradigm of Elastic Optical Networks (EONs). The transition from fixed-grid to flexible-grid architectures represents a fundamental shift in how optical spectrum is managed and allocated. Where conventional WDM systems were constrained by rigid ITU-T channel spacing and fixed bandwidth allocation, EONs introduce a more adaptive framework that can dynamically assign spectral resources based on actual traffic demands. This evolution has transformed the nature of the routing and spectrum assignment problem, creating new challenges and opportunities for network optimization.

The research presented in this work has addressed these challenges through the development and application of a comprehensive simulation framework specifically designed for EON analysis. Our simulator implements a sophisticated discrete-event model that captures the essential characteristics of dynamic network operation, including Poisson arrival processes, heterogeneous service requirements, and complex resource allocation constraints. The modular architecture of this simulation platform enables rigorous comparison of different spectrum allocation policies under consistent experimental conditions, providing valuable insights into their relative performance characteristics.

Through extensive simulation experiments conducted on both simplified mesh topologies and realistic network configurations, we have demonstrated that the choice of spectrum allocation policy fundamentally determines network performance and efficiency. Our results clearly show that deterministic allocation strategies significantly outperform stochastic approaches across all evaluated metrics. The Random-

Fit policy, while conceptually simple, consistently exhibited the worst performance characteristics due to its tendency to accelerate spectral fragmentation and create non-contiguous spectral gaps that cannot be utilized for larger service requests.

Among the deterministic strategies, our analysis reveals a nuanced trade-off between performance optimization and resource efficiency. The First-Fit policy emerges as the performance-optimal choice, achieving the lowest absolute blocking probabilities across the evaluated load range. However, this performance advantage comes at the cost of increased resource consumption, particularly in terms of wavelength converter utilization. In contrast, the Longest-Fit policy demonstrates superior resource efficiency, achieving dramatically reduced converter usage while maintaining competitive blocking performance. This policy also exhibits exceptional stability across different network configurations and parameter settings, making it particularly suitable for operational environments where predictability is valued.

The investigation of architectural features yields equally important insights. Wavelength conversion provides measurable performance benefits, particularly for longer lightpaths and more constrained network scenarios. However, these benefits come at significant cost, both in terms of hardware investment and operational complexity. Our results quantify this trade-off, showing that conversion typically reduces blocking probability by 2–6% while increasing connection costs by 18–24%.

This finding suggests that conversion should be deployed strategically rather than ubiquitously, focusing on critical network segments where its benefits are most pronounced. The analysis of break capability reveals this feature as a particularly valuable tool for network operators. Unlike wavelength conversion, which requires additional hardware infrastructure, break capability represents a software-defined feature that can be enabled or disabled through policy configuration. Our results demonstrate that enabling break capability consistently improves performance while simultaneously reducing converter requirements. This dual benefit makes break capability an attractive option for operators seeking to improve network efficiency without additional capital investment. The pathological behavior observed in no-break configurations further underscores the importance of this feature, as rigid allocation constraints can lead to severe performance degradation under certain conditions.

The practical implications of these findings are substantial for network designers and operators. Our results provide clear guidance for policy selection based on specific operational requirements: *First-Fit* for performance-critical applications, *Longest-Fit* for resource-constrained environments, and always with break capability enabled. These recommendations are supported by comprehensive quantitative

evidence across multiple network scenarios and traffic conditions.

Looking forward, this work establishes a foundation for several important research directions. The most immediate extension involves enhancing the physical layer model to incorporate transmission impairments, including nonlinear effects, amplifier noise, and filter narrowing. This extension would transition the analysis from a purely logical layer perspective to a more comprehensive physico-aware framework that can assess the actual feasibility of spectral allocations based on their transmission characteristics.

Another promising direction involves the development of more sophisticated economic models that incorporate the revenue implications of blocking events. By assigning economic values to different service types and quantifying the cost of blocked connections, future work could optimize allocation policies for total network revenue rather than simply minimizing blocking probability. This approach would provide a more holistic assessment of network performance that balances technical and economic considerations.

The integration of machine learning techniques represents another compelling research avenue. By implementing reinforcement learning or other adaptive algorithms, future versions of the simulator could develop policies that dynamically adjust their behavior based on current network conditions. This capability would be particularly valuable for handling non-stationary traffic patterns and unexpected network events.

Finally, the simulator architecture itself provides a platform for investigating emerging technologies such as multi-band systems and advanced modulation formats. These technologies are likely to play an important role in future optical networks, and their integration with RSA algorithms presents numerous interesting research questions.

In conclusion, this thesis has made several contributions to the understanding and optimization of Elastic Optical Networks. We have developed a comprehensive simulation framework for EON analysis, conducted rigorous performance evaluation of spectrum allocation policies, and derived practical guidelines for network design and operation. The insights gained from this work advance the state of the art in optical network design and provide a solid foundation for future research in this important area. The continued evolution of optical networking technologies ensures that these contributions will remain relevant as networks continue to increase in capacity, flexibility, and complexity.

Bibliography

- [1] ITU-T. *G.694.1: Spectral grids for WDM applications*. <https://www.itu.int/rec/T-REC-G.694.1>. 2012.
- [2] Mnotho Mkhwanazi, Khumbulani Mpofo, and Vusumuzi Malele. “Evolution of the optical add/drop multiplexer in dense wavelength division multiplexing optical networks”. In: *Indonesian Journal of Electrical Engineering and Computer Science* 39 (July 2025), p. 247. doi: [10.11591/ijeecs.v39.i1.pp247-257](https://doi.org/10.11591/ijeecs.v39.i1.pp247-257).
- [3] Varsha Lohani, Anjali Sharma, Yatindra Nath Singh, Kumari Akansha, Baljinder Singh Heera, and Pallavi Athe. “Optical Networks”. In: *arXiv preprint arXiv:2408.07478* (2024). doi: [10.48550/arXiv.2408.07478](https://doi.org/10.48550/arXiv.2408.07478). URL: <https://arxiv.org/abs/2408.07478>.
- [4] Imrich Chlamtac, Aura Ganz, and Gabriel Karmi. “Lightpath communications: an approach to high bandwidth optical WANs”. In: *IEEE Transactions on Communications* 40.7 (1992), pp. 1171–1182. doi: [10.1109/26.153361](https://doi.org/10.1109/26.153361).
- [5] Rudra Dutta and G.N. Rouskas. “A Survey of Virtual Topology Design Algorithms for Wavelength Routed Optical Networks”. In: *Opt Net* 1 (Sept. 1999).
- [6] Infinera Corp. *The evolution of next-gen optical networks*. White Paper WP-NGON-10-2014. Sunnyvale, CA, USA: Infinera Corp., 2014.
- [7] Masahiko Jinno, Hidehiko Takara, Bartosz Kozicki, Yuki Tsukishima, Yuji Sone, and S Matsuoka. “Spectrum-efficient and scalable elastic optical path network: architecture, benefits, and enabling technologies”. In: *IEEE Communications Magazine* 47.11 (2009), pp. 66–73.

- [8] B. S. Heera, V. Lohani, and Y. N. Singh. “Evaluation of Resource Allocation Techniques in Fifth-Generation Optical Networks”. In: *Advances in Signal Processing and Communication Engineering. ICASPACE 2023*. Ed. by P. Kumar Jain, Y. Nath Singh, R. P. Gollapalli, and S. P. Singh. Vol. 1157. Lecture Notes in Electrical Engineering. Singapore: Springer, 2024, pp. 541–553. ISBN: 978-981-97-0562-7. DOI: [10.1007/978-981-97-0562-7_45](https://doi.org/10.1007/978-981-97-0562-7_45). URL: https://doi.org/10.1007/978-981-97-0562-7_45.
- [9] Ujjwal Yadav and Jaisingh Thangaraj. “Review and analysis of elastic optical network and sliceable bandwidth variable transponder architecture”. In: *Optical Fiber Technology* 44 (2018), pp. 42–52.
- [10] Ori Gerstel, Masahiko Jinno, Andrew Lord, and S.J. Ben Yoo. “Elastic optical networking: a new dawn for the optical layer?” In: *IEEE Communications Magazine* 50.2 (2012), s12–s20. DOI: [10.1109/MCOM.2012.6146481](https://doi.org/10.1109/MCOM.2012.6146481).
- [11] B. Ramamurthy and B. Mukherjee. “Wavelength conversion in WDM networking”. In: *IEEE Journal on Selected Areas in Communications* 16.7 (1998), pp. 1061–1073. DOI: [10.1109/49.725178](https://doi.org/10.1109/49.725178).
- [12] Bijoy Chand Chatterjee, Seydou Ba, and Eiji Oki. “Fragmentation Problems and Management Approaches in Elastic Optical Networks: A Survey”. In: *IEEE Communications Surveys & Tutorials* 20.1 (2018), pp. 183–210. DOI: [10.1109/COMST.2017.2769102](https://doi.org/10.1109/COMST.2017.2769102).
- [13] Poznan University of Technology (PUT). *SNDlib - Survivable Network Design Library*. <https://sndlib.put.poznan.pl/home.action>.