Effect of network topology on accuracy of optical quality of transmission prediction algorithm

Rakesh Nair

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Future Networked System)

Supervisor: Dr. Marco Ruffini

August 2022

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Rakesh Nair

August 19, 2022

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Rakesh Nair

August 19, 2022

Effect of network topology on accuracy of optical quality of transmission prediction algorithm

Rakesh Nair, Master of Science in Computer Science

University of Dublin, Trinity College, 2022

Supervisor: Dr. Marco Ruffini

During the past decade, there has been a rapid growth of data traffic in optical transmission. The demand for network bandwidth keeps on growing with the emergence of internet applications such as streaming, cloud, virtual reality, 5G, internet of things. The increase in data traffic will affect the response time and quality of services provided over the network and increase demand in the backbone Dense Wavelength Division (DWDM) Multiplexing network traffic. The quality of transmission needs to be evaluated before a solution is introduced. Thus, the machine learning method is proposed to assess the quality of transmission. The performance of the optical transmission is measured with the help of signal-to-noise ratio, Q-factor, and dispersion. The network capacity throughout optimization is one of the most critical features in terms of a solution's commercial viability. This algorithm will improve the path performance estimation accuracy by interrogating optical performance monitoring (OPM) devices in the network. Before being implemented in a real system, the algorithm's scalability will be verified using a Mininet-Optical packet-network simulator. We will be able to examine the optical network impairments for varying topologies with the aid of this development. The Watts-Strogatz technique is utilized to configure different combinations of topologies, and metrics datasets gathered with OPM were used to train the model. The model used for this study has demonstrated a 99% accuracy in predicting QoT of established lightpaths in various topologies built using the emulator.

Acknowledgments

I would like to take this opportunity to express my sincere gratitude to my supervisor Dr. Marco Ruffini, for his valuable guidance and motivation throughout the course of this dissertation. I would like to thank my family for their support and encouragement during my course at Trinity. I would also want to express my gratitude to Bob Lantz and Atri Mukhopadhyay, staff - TCD CONNECT for their support and advice during the process; without them, I would have found it difficult to finish my dissertation.

Any omission of acknowledgement does not reflect my lack of regard or appreciation.

Rakesh Nair

University of Dublin, Trinity College
August 2022

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The possibility of predicting the Quality of Transmission (QoT) metrics using a machine learning algorithm is investigated in this dissertation. The algorithm was trained using datasets from various topologies developed using the Watts-Strogatz model and the Mininet-Optical emulator. It also focuses on selecting the machine learning algorithm among the four that can forecast the QoT metrics with the most significant degree of accuracy. The purpose of this dissertation is explained in this chapter. The following section of the chapter defines the research's main question and explains how it led to the set of objectives that were set forth. The dissertation's contents are outlined chapter by chapter until concluded.

1.1 Motivation

For the past few decades, the growth of data traffic in the network has increased tremendously which has resulted due to the relentless demand for the high capacity required for providing and accessing multimedia services, educational services, e-commerce and healthcare services. As a result, the field of optical communications has undergone a significant evolution to support increased traffic. Examples include the development of advanced optical modulation formats that offer high spectral efficiency and intricate network architectures utilizing reconfigurable optical add-drop multiplexers (ROADMs) to support dynamicity, flexibility, and better utilization of available transmission capacity. Due to this, the demand for the transport layer's dynamic reconfigurability has been presented with new challenges. Due to this, the dynamic addition, deletion, and routing of wavelength channels might create changes in optical power that could degrade the signal quality. One of the key elements for enabling dynamic switching in ROADM networks is the presence of optical performance monitoring (OPM) functions Kilper et al. (2004a)

which measure the performance of an optical signal at the intermediate network nodes or inside the receiver itself to estimate the performance of a transmission network. The information offered by the OPM not only assists in dynamic reconfiguration and network performance optimization but also provides better use of resources, such as OSNR estimations. Despite this, on-site signal monitoring remains challenging to implement, primarily because of its high CapEx and OpEx. In order to get beyond the OPM's restriction, estimating functions are employed to forecast the performance of optical networks. One such estimator is the Quality of Transmission estimator, which calculates the OSNR signal deterioration. The advanced development of SDN controllers and heterogenous components has increased the uncertainty of the system performance, such as reducing the predictability of QoT, which adversely affected the network efficiency and complicated the network fault diagnosis. Enhancing QoT estimate and monitoring has emerged as an important goal for increasing effectiveness. The uncertainty produced by active components, such as erbium-doped fiber amplifiers (EDFA), has made using analytical techniques more challenging. The application of machine learning techniques has drawn much interest in an effort to enhance the estimation of Quality of Transmission (QoT).

A large amount of optical link data and setup is required to carry out such studies, which can be complex and expensive to acquire. To address the scarcity of testbeds and test platforms, the Mininet-Optical emulator is used, which is an extension of the Mininet SDN emulator for modelling optical layer transmission and emulating optical devices that can be controlled via SDN interfaces. Mininet-Optical makes it possible to simulate the behaviour of optical components by allowing the modelling of components, including transceivers, ROADMs with wavelength selective switches (WSS), EDFAs for boost, inline, and pre-amplifications, and OPM devices. OpenvSwitch is used to implement the data plane capabilities of Line Terminals (transceivers) and ROADMs. Virtual Ethernet (veth) links are used to simulate optical fibre links between ROADM nodes in order to simulate multi-channel WDM communications. This study uses Mininet-Optical to develop a QoT-E system Díaz-Montiel et al. (2021) that enhances its prediction performance using monitoring data. It offers two contributions: The Watts-Strogatz model is used to construct the various topologies needed for this work. In addition, it provides a QoT-E technique based on active lightpath monitoring in an optical SDN environment that reduces estimation errors brought on by wavelength-dependent power dynamics. For this, machine learning regression models will be used to build an estimation model, and their accuracy will be calculated and compared with the help of evaluation metrics in the machine learning algorithms. This algorithm will improve the path performance estimation accuracy by interrogating optical performance monitoring (OPM) devices in the

network. The algorithm's scalability will be validated using the Mininet-Optical packet-network emulator before implementing in the existing system. This development will help us assess the optical network impairments for various topologies.

1.2 Research Objectives

To achieve the objectives of the dissertation, the following research areas are examined:

1. Generate multiple sets of optical network topologies using Watts-Strogatz model and Mininet-Optical emulator.
2. Implement a prediction algorithm for estimating the Quality of Transmission (QoT) using datasets generated from the topologies of different configurations with the help of Mininet-Optical for predicting the path performance accuracy
3. Evaluate the machine learning algorithm on multiple optical topologies and their lightpaths.
   (a) Their outright performance
   (b) Comparison between different machine learning algorithm to identify the model which provide better accuracy and low error.

1.3 Outline of the dissertation

The rest of the dissertation is organized as follows:

1. **Chapter 2** presents a brief background about the technologies used in this dissertation, gave an overview of Watts-Strogatz model and Mininet-Optical emulator. It also presents the State-of-the-art of QoT-E analysis performed in the optical transmission network.
2. **Chapter 3** discusses the methodology used in this dissertation. It covers machine learning techniques for evaluating the QoT and evaluation metrics used for understanding the accuracy of the machine learning models. It mentions the shortest path problem and the algorithm used to resolve this issue.
3. **Chapter 4** outlines the design and implementation of the components necessary for the research. This chapter sets up the required technical background for the experiment to follow. Deep dive into the implementation of the topologies generated

with the help of Watts-Strogatz model and the Mininet-Optical emulator and the machine learning models, and its parameters used for optimising the model.

4. **Chapter 5** the observations made during this investigation are thoroughly analysed, discussed, and evaluated, along with the effectiveness of the machine learning method used. Graphs showing the results are included with the discussion.

5. **Chapter 6** concludes the dissertation, discussing about the challenges encountered and contributions made in this research. It concludes with a discussion of potential directions for future investigation.

Chapter 2

State of the Art

This chapter discusses the conceptional background information required to understand the proposed work better. It will also present the current state-of-the-art in the research area and tools used for building the project. The first section gives an overview of optical communication and its components, and the second section will describe the Watts-Strogatz model and its impact on the project. The third section will focus on selecting the emulator used for the study and the previous and ongoing research that is relevant to this study. Lastly, the machine learning algorithm and techniques used for evaluating the predictions are explored.

2.1 Optical Network Communication

The tremendous growth in the demand for services such as video streaming and calls, video conferencing, streaming sports, or movie contents, accessing real-time services, 5G services, and voice over internet protocol (VoIP) and the number of users and bandwidth used by each user has increased a toll on network bandwidth in order to facilitate the required demands for bandwidth, and it is essential to understand and leverage the optical fibre limit. Due to the rising demands, the capacity of optical communication systems is constantly increasing, and optical network architectures are becoming increasingly complex, transparent, and dynamic. Because of their dynamic nature, these high-capacity fiber-optic networks are vulnerable to a variety of transmission impairments. Because each fibre carries a massive amount of data traffic, even a brief interruption in service can have disastrous consequences. An optical network provides enormous bandwidth and infrastructure to deliver varieties of services mentioned above when and where required. Economically transmitting enormous data at higher rates over a single fibre has led to implementing multiplexing techniques. There are two ways to do it, increase the bit rate by

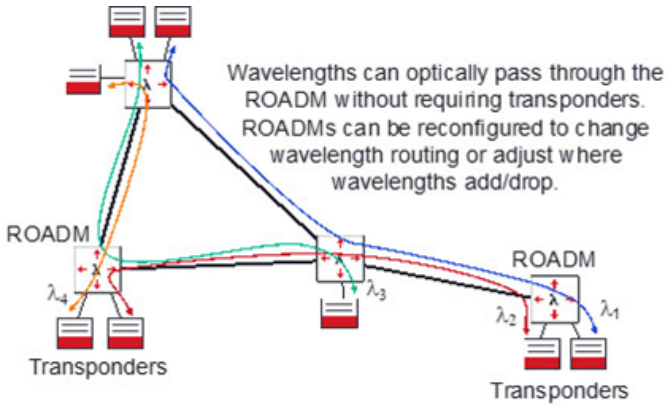

Figure 2.1: Traffics are reconfigured in the ROADM-based networks.

mean of time division multiplexing (TDM), where several optical signals are combined and transmitted together as a high bit-rate data stream, and it is separated (demultiplexed) again based on the arrival time into several lower bit-rate optical signals. However, engineering the impairments is difficult with increased bit rates. Another way to increase the link capacities is by using wavelength division multiplexing (WDM) by using multiple carrier wavelengths or channels to transmit data simultaneously. This technique helps make a single fibre look like multiple virtual fibres in which each virtual link will carry a single data stream. In optical communication, WDM transmission is widely used, which helps in providing more functions than just point-to-point transmission. The network provides lightpaths to each user which is an optical communication carried over a wavelength on each link. Different lightpaths in the network can use the same wavelength as long as they do not share any common path, which helps in reusing the same wavelength spatially in the different parts of the network. The optical line terminal (OLT) and reconfigurable optical add-drop multiplexer (ROADM) are the critical network components in enabling optical networking Tomlinson (2008). An OLT multiplexes multiple wavelengths into a single fibre and demultiplexes the same set of wavelengths on a single fibre into a separate one. It is generally deployed at the end of the point-to-point WDM links.

The ROADM Abedifar et al. (2013) is an optical add-drop multiplexer that allows the ability to add mechanism to route traffic by adding or dropping the wavelengths which are passing through the site, its working logic is similar to the generic routers that are used in the ethernet networks where it allows user to define the routing rules/tables, and the packets is dropped if it is meant to be in that path. The initial version of the optical add-drop multiplexer (OADM) did not have the ability to modify or reconfigure the switching rules after deployment. This drawback was later resolved when the ROADM

was introduced in the early 2000s; it enabled the option to reconfigure the lightpaths and supports more than multiple directions at a site. By solving this issue, switch rules and bandwidth assignment need not to be carried out during the phase of deployment of a system and can be reconfigured when required without affecting the traffic passing through the ROADM 2.1. These activities are enabled with the help of two components in the ROADM device; Wavelength Selection Switch (WSS) performs the actual wavelength switching and enables the user to route any wavelength to and from any port dynamically and Optical Channel Monitoring (OCM) for monitoring the optical power of each wavelength to ensure they are operating efficiently.

In optical communication, the power and signal levels are measured using the decibel units (dB), a relative measurement, and absolute power levels are measured in "dBm". The signal's power will always be higher at the central office of the network connection than at the customer end. The loss must be calculated between two points to understand the signal attenuation. It is measured by calculating the difference between the power coupled into the cable at the transmitter and the power that comes out at the receiver end. The optical fibres are combined, connected or traversed through the passive optical network components. The signal travelling in an optical fibre loses power over distance. The loss also depends on the wavelength of the light; the shorter the wavelength is attenuated the most. The transmission quality is affected when noise is introduced to the network, an undesirable disturbance that masks the receiving signals in an optical system. Three main types of noises are present in the optical fiber communication system: Thermal noise, shot noise and Amplified spontaneous emission (ASE) noise. The thermal noise, also known as Johnson-Nyquist noise, is generated by the random motion of electrons always present at a finite temperature within an electrical conductor. Every component with some temperature will exhibit noise superimposed on the output. The thermal noise increases with the increase in the temperature. The noise is proportional to the square root of the product's temperature, resistance, and frequency bandwidth. This noise level is dependent upon the temperature and the value of resistance. The shot noise, also known as quantum noise, is raised due to the discrete nature of the electric charges. It describes the fluctuations of the number of photons detected due to their occurrence independent of each other. Unlike thermal noise, the shot noise is dependent upon the current flowing and has no dependency or relationship with the temperature at which the system operates. Shot noise is more apparent in devices such as a transmitter.

An erbium-doped fiber amplifier (EDFA) is a device that amplifies an optical fiber signal. It works on the principle of simulating the emission of photons. When a signal is transmitted over a long distance, there are high chances of signal loss due to fiber attenu-

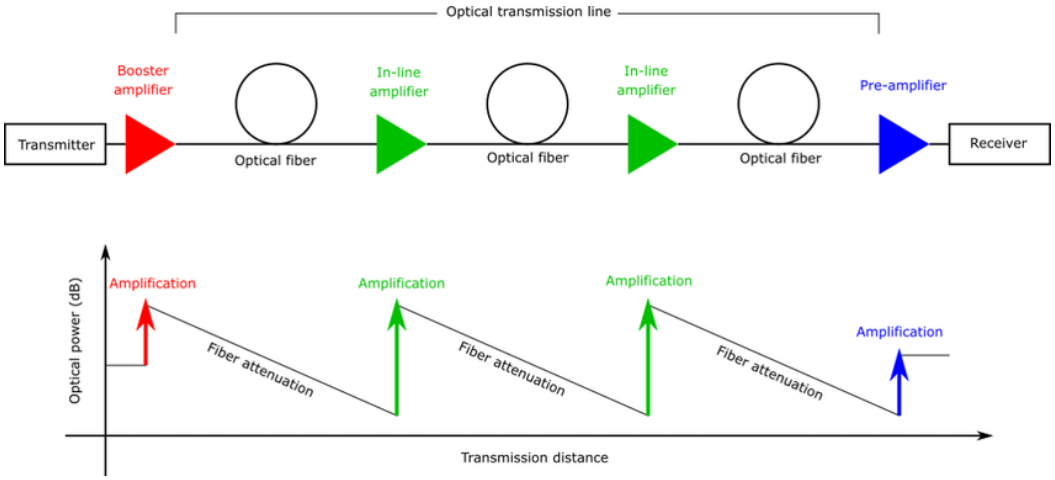

Figure 2.2: Booster, inline, and pre-amplifier EDFAs used in optical transmission line.

ation, connectivity losses, etc. these losses are compensated by amplifying many times in between. Earlier optical signal was converted first into an electrical signal, amplified and then converted back to an optical signal again. With EDFA, optical signals get amplified without the need to convert the signal into an electrical signal before amplifying. There are several types of fiber optic amplifiers: Semiconductor Optical Amplifier (SOA), fibre Raman and Brillouin amplifier, and EDFA. Among these, EDFA is most widely deployed in the WDM system. It can amplify multiple optical signals simultaneously combined with the WDM technology. The EDFA consists of an Erbium-doped fiber (EDF), pump laser and WDM combiner, which is used for combining the signal and pump wavelength for propagating simultaneously through the EDF. The EDFA are used as a booster, inline and pre-amplifier in an optical transmission line 2.2. The booster amplifier is placed after the transmitter to increase the optical launch power to the transmission line. It is not required in the single channel links but essential in the WDM link where the multiplexer attenuates the signal channels. It has high input and output power and medium output gain.

The inline amplifiers are placed in the transmission line to compensate the attenuation induced by the optical fiber. The in-line EDFA is designed for optical amplification between two network nodes on the main optical link. It features high output power, low to medium input power, low noise figure and high output gain. The pre-amplifier is placed before the receiver of a WDM link to have a sufficient optical power is launched to the receiver. It is used for compensating the losses in a demultiplexer near the receiver. It has relatively low input power, medium output power and medium gain power. The EDFA are widely adopted because of its high pump power utilization, support for simultaneously amplifying wide range of wavelength and relatively easy deployment and more affordable

compared to other signal amplification methods.

Unfortunately, the EDFA not only amplifies the input optical signal, but they also increases/amplifies the noise level and degrades the signal-to-noise ratio. When spontaneous emission happens in a gain medium such as EDFA, it gets amplified by simulated process. Amplified spontaneous emission (ASE) is produced when a gain medium is pumped to produce a population inversion. Consequently, spontaneous emission occurs initially, followed by its amplification by the stimulated emission process in the gain medium. For this study, we will mainly focus on the Amplified spontaneous emission (ASE) to understand and evaluate the quality of transmission (QoT) in the transmission line.

The optical signal-to-noise ratio (OSNR) is used to quantify the degree of optical noise interference on optical signals. It refers to the signal degradation brought on by the ASE noise that optical components like amplifiers bring to the transmission line. When the signal is amplified by the EDFA, its OSNR are reduced and impacts the receiver the most because the low OSNR value means that the receiver will not be able to recover the signal. The higher the OSNR value, better it is for the overall system. The performance of an optical transmission system is evaluated using the OSNR as a benchmark. To ensure error-free operation, the WDM network needs to be operated over its OSNR limit.

The standard calculation for OSNR is as follows:

$$ONSR = 10dB * \log_{10}\left(\frac{S}{N}\right) \tag{2.1}$$

Where S and N, both stated in watts/milliwatts, stand for the signal and noise powers, respectively.

2.2 Watts-Strogratz Model

A small world network is a network feature characterized by a large clustering coefficient and a small average shortest path length i.e., most nodes which are not neighbours of one another can be reached from every other by a small number of hops. Small world is usually identified by existence of short path length between two randomly picked nodes, cliques or near cliques with high clustering co-efficient i.e., sub-networks with connections between essentially any two nodes within them, are common in small-world networks, and degree of node in the network has a power distribution. it is unlikely that the deletion of a peripheral node will obstruct communication with other peripheral nodes. Some examples for small world network are – electric power grid, the network of brain neurons, airport network and telephone call graphs. To understand the distance geodesic distance

is calculated between two pairs of nodes which is a minimum number of edges that need to be traversed from the starting node to the destination node. The diameter of a network is the maximum of the geodesic distances between node pairs, and the world encompassed by a graph is "small" if the predicted number of hops between two randomly chosen persons is small. There are three most commonly heard network models used for generating small-world network - Erdos-Renyi Model, Watts-Strogatz Model and Barabasi-Albert Model. For this study, Watts-Strogatz model are used for generating the optical network topologies.

The Watts-Strogatz model Watts and Strogatz (1998) is a random graph generation mechanism that generates networks with small-world traits such as low average path lengths and strong clustering. It is widely used in the simulation of the small-world system. The model contains parameters set having three variables representing the group size, number of neighbours and rewiring probabilities.

The rewriting process or generation of a Watts-Strogatz model are as follow –

1. Build a regular ring lattice with **N** node with mean degree of **K**, with each node is connected to its **K/2** nearest neighbours on either side.

2. With probability β each edge (x, y) is rewired in the network to a random node selected using a randomizer and new connection (x, y') are established instead, where y' is picked at random from among all possible nodes while avoiding self-loops (y,x') and link duplication.

Below is an example for a ring lattice topology generated by Watts-Strogatz model with Node N = 20, mean degree K = 4, and different sets of probabilities.

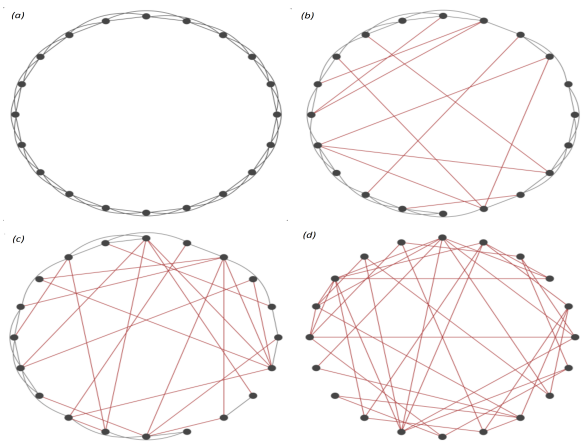

Figure 2.3: Ring topology with probability - (a). $\beta = 0$, (b). $\beta = 0.32$, (c). $\beta = 0.50$ and (d). $\beta = 1.0$

Figure 2.3 (a). A ring topology is created with probability $\beta = 0$ in which each node is connected to the same number of nearest neighbours on either side. A Watts-Strogatz model is created by removing each edge based on the probability value and rewired it to yield an edge between the new pair of nodes chosen uniformly at random. when the probability $\beta = 1.0$, all the edges are rewired, and the ring lattice network is transformed into random graph. The major limitation of this model is the unrealistic degree distribution which does not follow power-law. The Watts-Strogatz model also assumes a certain number of nodes, making it impossible to simulate network expansion.

2.3 Mininet-Optical

The continuous advance of the evolution of technologies and features in the optical network world has challenged vendors, researchers, and network planning community for foreseeing, investigating, and testing the new technologies to understand/evaluate deserving resources and investments to be made for the network planning, optimization of the optical layers and its deployments. Research solutions are undergoing significant changes in both technologies and methodology to provide cost reduction, more advanced security, reliability, scalability, and sustainability. It is not possible to test the features with large number of hosts, switches/devices and SDN controller on physical devices and servers. For this, simulation of several sets of configuration scenarios, network recovery tests, traffic load analysis and analysis of newly developed algorithm is required without relying on a specific vendor.

Mininet-Optical Mininet-Optical Project (2022) is an opensource network emulator for both simulating the mechanics of optical transmission and creating an optical transmission and switching plane that is controlled by the SDN. It helps in providing a virtual test bed for modelling optical transmission physics by creating hosts, links and switches and its behaviour by using the OpenFlow protocol with the help of processes, network namespaces and various features provided by linux kernel such as Open vSwitch and virtual ethernet (veth). It emulates the data plane of both packet and optical networks and simulates their physical behaviour and impairments of the optical network. Mininet-Optical supports discrete optical components such as amplifier (EDFA) for boost, inline and pre-amplification, ROADMs, transceivers, and optical fiber links. It allows you to connect an emulated packet-optical network to a widely used open source SDN controller (ONOS). This tool allows the user to customize the implementation and configuration of each network elements. To evaluate the impact of these configurations on the physical performance of the system, it makes use of gnpy library for calculating the propagation

performances by evaluating OSNR/g-OSNR of each channel. Mininet-Optical provides an external control API using which user can write their custom configurations or algorithms using python for interacting with SDN interface.

2.4 Machine learning Overview

Machine learning (ML) technique, as the name suggests, is a subclass of artificial intelligence that allows the system to automatically learn and improve from experiences from accessing data without explicit programming. Machine learning algorithm relies on input, such as training data to understand the entities or features and the connection between them. It builds a statistical model based on such input data for analysing complex structures and provide predictions. Models can be trained to identify the patterns and relationships between input data and automate the routine processes such as speech and image recognition, email filtering, computer vision, fraud detection and recommendation system. The learning system of a machine learning algorithm can be divided into three main parts –

1. **Decision Process:** In order to estimate a pattern in the data, the machine learning algorithm is utilized to make predictions or classifications based on the input data.

2. **Error Function:** The error function evaluates the prediction of the model by comparing the distance between the predicted values and the true values. It can make a comparison for assessing the accuracy of the model.

3. **Optimization Process:** The weights are autonomously adjusted until an accuracy threshold has been met to better fit the data points in the training dataset. The algorithm will repeat this evaluate and optimize process for reducing the variation between the true value and the model estimation.

The machine learning algorithm falls under three primary categories:

2.4.1 Supervised learning

Supervised machine learning algorithms are trained with labelled (desired output) datasets to learn and increase accuracy over time for classifying data or predicting outcomes more accurately. The model modifies its weights as input data is fed into it until the model is well fitted. The cross-validation process is used to ensure that the model avoids overfitting or underfitting. To train the underlying algorithm, a tagged training dataset is initially

used. The unlabelled test dataset is then fed this trained algorithm to make output value predictions.

The supervised learning is classified into two categories of algorithm:

1. **Classification:** It uses an algorithm to accurately categories test data into specific groups, such as identifying spam mails, classifying colour – red or blue, and housing prices. Models will label the data they analyse, which is learnt by the algorithm through training on labelled training data. The data input and output have been labelled so that the model can comprehend which characteristics will categorize an object or data point with distinct class labels. Common classification models are support vector machines (SVM), decision trees, and k-nearest neighbour.

2. **Regression:** Regression model are used to understand the relationship between the dependent and independent variables, most commonly used for predicting and projections/ forecasting. It is generally used to predict continuous outcomes. Some popular algorithms are linear regression, lasso regression, Ridge regression and Random forest.

2.4.2 Unsupervised learning

Unsupervised machine learning algorithm are used to analyse and cluster unlabelled datasets to discover hidden patterns or grouping the data without the need of human interventions. The count of the clusters is usually defined by setting it in the hyper-parameters. It works best when we do not have data on the intended results, such as when trying to figure out who the market is for a brand-new product that the company has never sold. Some use cases for unsupervised learning are looking through the online sale data and identify different types of clients making purchases, detecting anomalies and outliers, and clustering customer data based on similarities. Commonly used unsupervised algorithms are K-means algorithm, KNN (K nearest neighbours), and principal component analysis (PCA).

2.4.3 Reinforcement learning

Reinforcement machine learning algorithms are a learning method that interacts with its environment by training machines through trial and error to select the best actions with the help of a reward system. With the aid of this system, it is possible to automatically decide which actions to take in a given situation in order to maximize performance. Although both supervised learning and reinforcement learning use mapping between input

and output, reinforcement learning uses rewards and punishments as signals for positive and negative behaviour. This is in contrast to supervised learning, where the feedback provided to the agent is the correct set of actions to perform a task. The main elements of reinforcement learning systems are – The agents, environment, policy that the agents follow to take actions and reward system. Some examples are, robotics for industrial automations, strategy planning, and autopilot control for cars and aircraft. Some commonly used algorithms are Q-learning, policy iteration, value iteration and Markov decision process (MDP).

2.5 Previous Work

The previous section introduced the background for optical transmission and the overview about the machine learning algorithm and network generation model required for this study. This section will explain the previous work and studies carried out in this field.

2.5.1 Optical Performance Monitor

Zhenhua in his paper Zhenhua et al. (2016) discussed and analysed the significance of Optical performance monitoring (OPM) which does the estimation of different physical parameters of transmitted signals and various components of an optical network. The author examined potential difficulties that could arise in the scalable optical network and evaluated recent work in the field of optical monitoring. The author further discussed the OPM techniques for different systems such as direct detection systems, digital coherent systems, OPM functionalities in elastic network operations and OPM devices in optical networks.

Digital OPM methods, e.g., Asynchronous sampling-based techniques, asynchronous amplitude histograms (AAHs) Shake et al. (2001); Li et al. (2005); Kozicki et al. (2008), asynchronous delay-tap plots (ADTPs) Khan et al. (2010); Dods and Anderson (2006); Wu et al. (2010), asynchronous two-tap plots (ATTPs) Jong (2008); Khan et al. (2011), and asynchronous single channel sampling (ASCS) Yu et al. (2014), are considered attractive since they do not require clock information and they are also capable of monitoring multiple impairments simultaneously, thus being cost-effective. The spectral resolution, in this case, is determined by the linewidth of the LO laser and is several orders of magnitude higher than that of a tunable optical filter. Since the optical filter or the LO laser needs to be tuned for scanning the whole WDM spectrum, such techniques can introduce measurement latency. The clock tones-based monitoring techniques can measure CD and

PMD and are dependent on data rate and modulation format Wang et al. (2007). Besides monitoring the specific tones (i.e., clock and pilot tones) in the RF spectrum, changes in the spectral distribution of the overall RF spectrum due to various network impairments may also be analyzed for monitoring these impairments Zhao et al. (2009). Advances in coherent detection and DSP over the past decade together defined the current generation of optical transmission systems and opened up the phase and polarization of an optical carrier for information encoding. High order modulation formats such as PM-QPSK and PM-16QAM enable data transmission rates per channel to move beyond 100 Gb/s. This is more commonly known as channel estimation in wireless communications literature, but it shares the same objectives as OPM in a general sense. Impairment-aware routing has long been a goal for OPM, but it was not until the introduction of digital coherent transmissions that modulation formats, bandwidth and bit rate can be made adaptive according to real-time link impairments and traffic demands. The key enabling technologies supporting EON include de Miguel et al. (2013). Adaptive Elements These include flexible bandwidth transmitters and receivers (called bandwidth variable transceivers (BVTs)), bandwidth variable wavelength cross-connects (BV-WXCs), etc., which give the network the capacity to modify its configuration adaptively. Monitoring Mechanisms The monitoring elements allow the EONs to be fully aware of the current network conditions, which is a prerequisite to be adaptive. Rather, the useful network information can directly be retrieved inside the DSP-based coherent receiver itself. Since optical orthogonal frequency division multiplexing (OFDM) may play a role in future EON, there is considerable interest in developing OPM techniques for OFDM signals. Recently, a few experimental works have demonstrated OPM's functionalities in EONs Geisler et al. (2011); Jin et al. (2011). Geisler et al. (2011) employed OPM and a real-time adaptive control plane to optimize the network parameters depending on the PLIs. In their work, signal quality was monitored at various network nodes, equipped with the necessary monitoring mechanisms, and the resulting information was then communicated to the network control plane. OPM can also help in the realization of PLI-aware routing in EONs in order to improve the overall network efficiency. OPM devices typically employ a tuneable band-pass filter, or a diffraction grating combined with a single detector to monitor the mentioned parameters Kilper et al. (2004b). Recently a commercially available high-resolution OCM has been reported in Rosenfeldt et al. (2015). An ADTS-based technique is used to monitor the parameters mentioned above in a field trial simultaneously. For example, it is revealed that the root cause of the higher than expected pre-FEC BER and reduced system margin of the monitored link is the high amount of residual CD resulting from the use of an incorrect dispersion compensation module. Zhenhua et al. (2016) concludes that OPM remains an

essential component of optical network operation. Datacentres and cloud computing have increased the demand for OPM to manage network faults and EONs. OPM and related optical network functionalities are expected to play a growing role in the development of next-generation optical networks.

2.5.2 Network Topology Generation

While topology should have no bearing on the accuracy of network protocols, Hongsuda et al. (2002) stated in his article that topology occasionally significantly impacts network protocols' performance. For this reason, network researchers frequently create accurate topologies for their simulations using network topology generators. Waxman (1988) created the first network topology generator frequently used in protocol simulations. The link formation probabilities in this generator, an adaptation of the conventional Erdos-Renyi random graph Bollobás (1985), are skewed by the Euclidean distance between the connection ends. He considered three classes of network generators in this paper. The first category, random graph generators, is represented by the Waxman generator Waxman (1988). The second category, the structural generators, contains the Transit-Stub Calvert et al. (1997) and Tiers generators Doar (1996). Finally, degree-based generators such as power-law random graph (PLRG) Aiello et al. (2000) are discussed, which create several top-level networks, each attached to several intermediate-tier networks. Links are then assigned randomly, picking two node copies and assigning a link between them until no more copies remain. The author began by challenging the generally held notion that degree-based generators are preferable to structural generators simply because they fit the degree distribution of the Internet. They lack some nodes and links, making them insufficient. The graphs do not reflect link speeds or policy routing; they depict connectivity (although we have attempted to approximate policy routing). The attention in this study has been limited to relatively big graphs since they were trying to determine which family of generators best represents the large-scale structure of the Internet (the smallest generated graph had 1000 nodes). The structural generators in use today or those that have yet to be created might be preferable options for small-scale simulation investigations.

2.5.3 Quality of Transmission Estimation (QoT-E) using Machine Learning

Diaz-Montiel et al. (2019) studied the potential of deploying QoT estimation tools with a multi-class Support Vector Machine classifier to assist the routing and wavelength assign-

ment module of future optical control systems, in order to improve the management of network resources. They further discussed about the research on prediction model carried out in the field of optical control system.

Barletta et al. (2017) took into account the use case of assessing whether unestablished lightpaths meet a necessary BER threshold. The conclusions were expanded in Rottondi et al. (2018), which also provided a more in-depth investigation of the obstacles that network components in optical networks present to the creation of cognitive control systems. In order to simulate the situation of an online control system, Bouda et al. (2017) collected and trained synthetic data on-the-fly. Many physical layer settings were used (i.e., launch powers, fiber span losses of specific links). Using an emulated 88-channel system, QoT prediction with 0.6 dB Q-factor accuracy was accomplished. In impairment-aware wavelength-routed optical networks (WRONs), Mata et al. (2017) have explored the possibility of SVM in classifying lightpaths into good or low-quality categories. 11000 samples from a dataset were utilized for training the learning model. While maintaining a classification accuracy of 99.9%, the new models with RF and bagging trees beat their old SVM model in terms of computing efficiency. Meng et al. (2017) Markov Chain Monte Carlo-based learning model helped them achieve a Q-factor estimation inaccuracy of 0.5 dB. Deep neural networks (DNNs) were investigated by Mo et al. (2018) to forecast the power dynamics of a 90-channel ROADM system. They conducted studies in a small testbed and conducted online training with 6720 training samples to examine the effects of power excursions during the EDFA amplification process of EDFAs.

The author used an Optical-MAN emulator for generating network topologies composed of EDFA links, ROADM equipped with WSS and AGC-EDFA for post/pre-signal amplification. Each transmission was enabled with 90 channels, and for their study, they created a linear topology with multiple end-to-end transmission connections. From Ghobadi et al. (2016), four QoT classes are considered: OSNR $\geq$ 17 dB $\geq$ 14 dB $\geq$ 10 dB, corresponding to 16 Quadrature Amplitude Modulation (QAM), 8QAM, and Quadrature Phase Shift Keying (QPSK) modulation formats, respectively. They set the minimum OSNR threshold at 10 dB; below this level, the lightpath is not feasible, putting it in the "none" category. They used 15 wavelength load (WL) scenarios to train the SVM model: N=I, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, and 70. The data set contains 37,968 samples, with an 80-20% split between training and test data. They then used a 5-fold cross-validation strategy for the splitting strategy. For the four QoT classes, the multi-class SVM classifier is evaluated. This tool's overall accuracy for the multi-class classification use case was 96.2%. The confusion matrix divides the percentage of correctly classified OSNR levels into four classes: 16QAM, 8QAM, QPSK, and below the OSNR

threshold (none). Despite the exceptional classification accuracy, a significant drawback of this implementation of the multi-class SVM classifier is the computational time required to train the model, given its complexity $O(n^3)$. They concluded that they would look into alternative learning models, such as neural networks, to support the multi-class classification scenario and achieve faster training rates. They also intend to incorporate nonlinear noise effects into the Optical-MAN emulator to use their learning models on top of physical testbeds. Their ultimate goal was to integrate the QoT estimation tool into a real SDN optical control plane.

2.6 Summary

In this section, a background overview and discussed fundamental features of optical communication and its components, Watts-Strogatz model used for building the topologies, Mininet-Optical and the machine learning algorithms are given. We also discussed the key elements that enabled the optical link transmission.

Finally, this section was concluded, by discussing the research carried out in the field of Optical performance monitoring (OPM), network topology generation and existing suggestions in literature for implementing prediction system for analysing the quality of transmission estimation (QoT-E) over the optical link transmissions.

Chapter 3

Methodology

The chapter discuss the methodology used for building this dissertation work. The section 3.1 explains all the machine learning algorithms used in this research for predicting the Quality of transmission (QoT) for predicting and evaluating optical transmission performance. Section 3.2 mentions all the evaluation metrics used for evaluating the performance of the algorithm. Section 3.3 concludes the chapter with the formal introduction to the shortest path problem and algorithm used for solving this issue.

3.1 Machine learning algorithms

Machine learning algorithms are used for creating a prediction model. For this work, four supervised algorithms are used which best suits this study. These algorithms are Linear regression, Lasso/Ridge regression, Random forest, and neural network.

3.1.1 Linear regression

Linear regression is a supervised machine learning algorithm used for finding the linear relationship between an independent variable X and a dependent variable y for predicting the outcome of future events. For example, a model might want to find the relate the weights of an individual to their heights using linear regression model. The variable we want to predict is called the dependent variable or outcome variable; the variables used to predict the other variable's value is called the independent variable. The mathematical equation for the linear regression is as follow:

$$Y = m * X + b \tag{3.1}$$

3.1 Where X is a dependent variable (target), Y is an independent variable, m for estimated slope and b is the estimated intercept.

This regression algorithm's main objective is to determine two things:

1. does a set of independent variables does a great job in predicting an outcome/dependent variable

2. which independent variables are significant predictors of the dependent variable.

3.1.2 Ridge regression

Ridge regression is a technique for analysing multiple regression data that suffers from multicollinearity by performing L2 regularization. When multicollinearity occurs, least squares estimates are unbiased, and variances are significant, due to which the predicted value will be far away from the true values. Ridge regression lowers the standard errors by biasing the regression estimates to some extent. In ridge regression, the first step is to standardize the variables by subtracting their means and dividing by their standard deviations. Overfitting problem may lead to inaccurate and unstable model, to minimize the overfitting problem in the model a technique called regularization is used. It adds squared magnitude of coefficient as penalty to the loss function. A tunning parameter (λ) to control the strength of the penalty, when $\lambda = 0$, model will be least squares regression, while $\lambda = \infty$, all coefficients are shrunk to zero.

3.1.3 Random forest regressor

Random forest regressor is a supervised machine learning algorithm which is an ensemble of decision trees. Ensemble learning is the process of using multiple models which is trained over the same data to improves its accuracy and reduce overfitting, and the average result of each model is calculated to find more powerful predictive model. It randomly performs row sampling and feature sampling from the dataset to form sample datasets for every model. It uses mean squared error (MSE) to measure the quality of a split. It uses bagging method and random feature selection to resolves the problem of overfitting. It scales well when new features are added to the dataset. It provides better accuracy "out-of-the-box" without tuning hyperparameter when compared to other linear algorithms. It has three main hyperparameters which need to be set before training that are – node size, number of trees and the number of features sampled. But it does have few disadvantages – Finding trends that would allow it to extrapolate values outside of the training set is not possible, it takes much higher time when handling large datasets as they are computing

data for each individual decision tree, it requires significant amount of memory for storing and retaining information from several numbers of individual trees. Some common use-cases for random forest model are – recommendation system in e-commerce domain for cross-sell purposes, fraud detection and credit scoring in the banking sector, it is also used in the medical domain to estimates the drug responses to specific medications and price prediction for housing or any specific new products.

3.1.4 Neural network

Neural network learning algorithm is a computational learning system that uses a network of function to understand and translate a data input of one form into a desired output. It works in a similar way how neurons of the human brain function together to understand the inputs from human senses. It is comprised of node layers, consisting of an input layer, one or more hidden layers and output layer, which delivers the final output. Most neural network are fully connected to each other 3.1 i.e., each hidden unit and each output unit is connected to every unit in the layer either side.

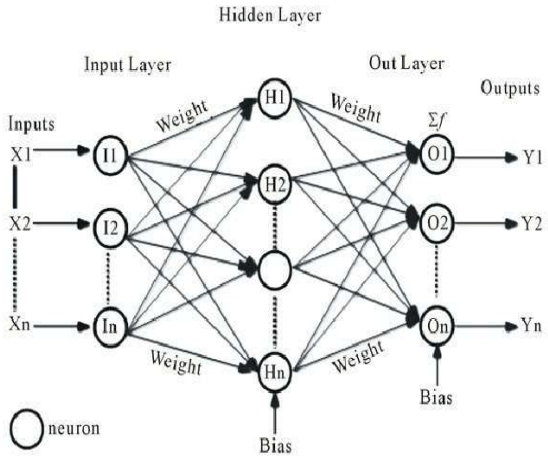

Figure 3.1: Neural network architectures

Each node connecting to one another has a weight and threshold associated with it. If the output of any individual node is above the threshold; data is sent to the next layer; no data is passed along within the network. This helps us learn which essential features are in the data to produce the output. Each node has its own linear regression model, consisting of data, threshold, weight, and output. Neural networks have several use cases across many industries such as – targeted marketing by social network filtering and behavioural data analysis, financial prediction based on processed historical data, chemical compound identification, visual recognition, automated chatbots, and recommendation engines.

3.2 Evaluation

Once the machine learning algorithm are trained properly with the training set, the performance of the model needs to be evaluated to understand the accuracy and the error for assessment. Various evaluation techniques and metrics are used for quantifying the performance of the machine learning model. In this section different techniques and metrics are discussed which will be used later for the evaluation processes.

3.2.1 Hyperparameter optimisation

Hyperparameters are parameters of the model that cannot be learned directly from the data. Hyperparameters define the degree of freedom the model has for exploring to find the proper fitting for the data. The hyper-parameters of each model must be optimized after the best training set size for each model has been determined. Each of the model's hyperparameters is selected at a distinct value during this phase and trained over. The validation set is then used to assess their performance. The linear regression does not have any hyperparameters, but random forest and ridge algorithm have the ability to tune the hyperparameters to avoid the overfitting by penalizing the model. As discussed earlier, ridge algorithm uses L2 regularization by shrinking coefficients for those input variables that does not contribute much to the prediction work. In random forest regressor, the hyperparameters are the number of decision trees. The more trees are defined, the time complexity of the model will also increase, and it is not necessary that having more number trees will provide more optimal results. The hyperparameters for both models mentioned above are found by iterating through the list of possible parameter values. The optimal hyperparameter values are identified by analysing the root mean squared error (RMSE) and mean absolute error (MAE).

3.2.2 Root Mean Squared Error (RMSE)

RMSE is the standard deviation of the residuals i.e., a measurement for amount of error in the model. It is calculated by taking the square root of average of square of the distance between actual/true values and estimated/predicted values. This distance is termed as an error. Lower the RMSE, better the forecast, i.e., closer the model is at finding the best fit. It indicates absolute fit of the model to the data. RMSE is commonly used in supervised machine learning algorithms.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (X_{obs,i} - X_{pred,i})^2}{n}} \tag{3.2}$$

3.2 n = number of observations, X_{obs} is an actual value and X_{pred} is a predicted value.

The RMSE lends comparatively significant weight to large errors since the errors are squared before they are averaged. In situations when significant errors are most unwelcome, the RMSE is thus most helpful. Because of this, RMSE is sensitive to outliers.

3.2.3 Mean Absolute Error (MAE)

Mean absolute error is a model evaluation metric used with regression models for calculating the absolute error difference between predicted value and true value. It is calculated by taking the sum of the absolute errors and then divide it with the sample size. Depending on the absolute value of the error, each error makes up a percentage of the MAE. Because RMSE involves the squaring of differences, a small number of noteworthy discrepancies might cause the RMSE to rise over the MAE.

$$MAE = \frac{\sum_{i=1}^{n} \mid X_{obs,i} - X_{pred,i} \mid}{n} \tag{3.3}$$

3.3 n = number of observations, X_{obs} is an actual value and X_{pred} is a predicted value.

Similar to RMSE, the closer the value of MAE is to zero, the better, which indicates a better model with lower error in its predictions. Unlike RMSE, MAE is not sensitive toward the outliers because it is generally used when we do not want outliers to impact the model's performance.

3.2.4 R^2 Score

R^2 Score also known as coefficient of determination, it indicates the percentage of variance in dependent variable that explained by an independent variable in a regression model. Higher the R2 score, smaller the differences observed between actual and predicted value. It is measured in scale 0-1. i.e., If score is 0.50 (50%), half of the actual variations can be explained by the model.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{3.4}$$

3.4 SS_{res} stands for sum of square of residuals i.e., unexplained variation, while SS_{tot} denotes total sum of squares which is total variation.

The score for all the machine learning model used in this study are performed using R^2 score.

3.3 Shortest path problem

Shortest path problem consists of finding the shortest paths between a given vertex/edges in a network graph while avoiding null path or self-looping in the network. It is usually calculated by considering the hop counts and/or least cost/weights as compared to all other existing paths. The input graph for shortest path algorithms is made up of vertex nodes and the connections that connect them. When edges are bidirectional, then graph is called undirected. Similarly, when edges are unidirectional, they are termed as a directed graph. It has several real-world use cases such as, it is used to automatically find directions between locations such as road networks using google maps, network and telecom communications, social network platforms for suggesting list of friends based on mutual connections and interests, robotics and drones, power grid contingency analysis and electronic designs.

For this study, an algorithm is implemented a single-source shortest path problem with unweighted graph, where we want to compute the distance $\delta(s, t)$ from a single source node s to every target node t. For an unweighted graph, i.e., all edges have the exact cost, it is implemented using a simple breadth-first search.

3.3.1 Breadth-first search algorithm

The breadth-first search algorithm is a traversing algorithm that begins at the starting/-source node and traverses the graph by exploring neighbouring nodes that are directly connected to the source or current node. It continues to move towards the next level of neighbouring nodes until it reaches the destined node. While traversing through each node/vertices, it keep track of which vertices have been visited to avoid self-looping. Thus, it keeps track of two categories of nodes – visited and not visited for avoiding cycles. A queue (FIFO – First In First Out) data structures are used by BFS for storing adjacent nodes to the selected or current node, so that node's neighbours will be viewed in the order in which it is inserted into the queue. BFS visits an adjacent unvisited node stored in the queue and mark it as done and remove the previously visited vertex from the queue in case no adjacent vertex is found.

Since the breadth-first search algorithm would, in the worst scenario, investigate all vertices and edges, its time complexity can be expressed as $O(|V| + |E|)$. The symbol for

Algorithm 1 BFS algorithm

```
procedure BFS(G, s)
    for each vertex v ∈ V[G] do
        explored[v] ← false
        d[v] ← ∞
    end for
    explored[s] ← true
    d[s] ← 0
    Q:= a queue data structure, initialized with s
    while Q ≠∅ do
        u ← remove vertex from the front of Q
        for each v adjacent to u do
            if not explored[v] then
                explored[v] ← true
                d[v] ← d[u] + 1
                insert v to the end of Q
            end if
        end for
    end while
end procedure
```

the number of vertices is $|V|$, while the symbol for the number of edges is $|E|$.

The topologies for this study are generated using Watts-Strogatz algorithm which introduces randomness based on the value of the probability parameter because of which, it was not possible to pre-define the routing path from the selected node to the destined node. Before configuring the network connections in the topologies shortest path between the nodes are calculated based on the hop count using the Breadth-First search algorithm and feed to the network configuration function.

Chapter 4

Implementation

This chapter discusses the design and the implementation of the above mentioned techniques required for this study. This chapter will start with the environment setup and proposed network design and present in-depth analysis of the systems and tools used for exploring the predictivity of the QoT-E of the optical transmission.

4.1 Environment Setup

In this paper, topologies are made using Mininet-Optical emulator for generating the required datasets for the prediction model and testing out the performance of the optical transmission links. This tool simulates the physical behaviour and impairments of the optical network and emulates the data plane of both packet and optical networks by exposing SDN API's to SDN controllers

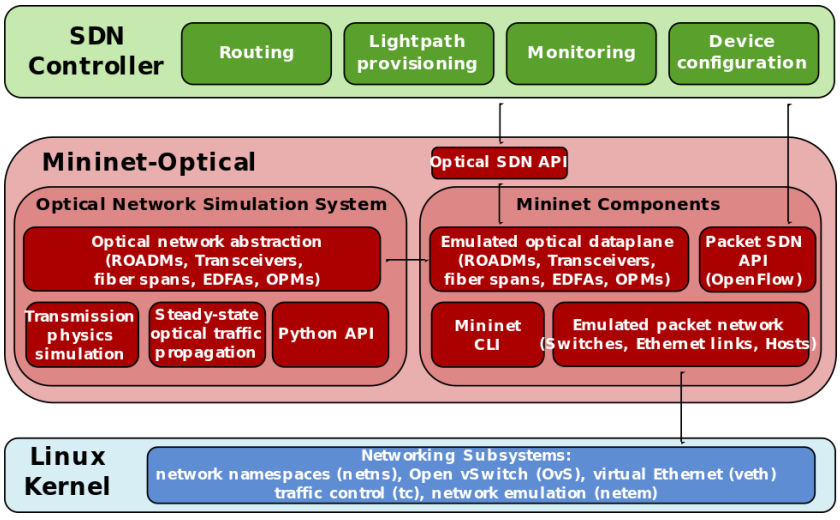

Figure 4.1: Architectural overview of the Mininet-Optical emulator

Mininet-Optical creates an abstraction layer over the Linux kernel. At the bottom, network subsystems are used for creating virtual network. The middle layer shows how Mininet are incorporated and extended for implementing various optical networking devices such as fiber optic cables, terminals, ROADMs, and EDFAs. The top layer is the control plane interface for SDN controllers for emulated network elements. These same interfaces are used by the controller to retrieve optical performance monitoring (OPM) information, allowing us to collect optical signal power, amplified spontaneous emission (ASE), and other noise data of individual channels, which also includes OSNR and gOSNR and study selective effects that can manifest over an optical link. With the help of this tool, the user will be able to customise the configuration of the network element individually by setting the wavelength dependency gain function in the EDFA and evaluating its performance in the optical transmission system with signal power behaviour modelling physical testbed performance. Due to the modular design of this system, it is possible to extend and modify the transmission physics models. The ROADM nodes offered by Mininet-Optical have Variable Optical Attenuators (VOA) at each output port to provide a mechanism of variable attenuation to the optical signal as well as WSSs for providing directions to the signals that are traversing. This simulation system is developed entirely in python language and can be used as a stand-alone system for offline simulations and for the purpose of prototyping, it can be imported as a library of APIs with access to the descriptive models of the network elements, allowing for active reconfiguration. The user can test their custom algorithms with the help of REST APIs exposed by the SDN controllers and write their custom scripts in python programming language. With the help of OPM, user can extract the required readings such as OSNR, gOSNR and ASE noise information of each channel in the transceivers which will be used as a dataset for training and validating the machine learning models.

Most of the network configurational work and machine learning algorithms are done with the help of python programming language. For building the topologies two systems are used –

1. Lab system with i7-7700k cpu, 16gb RAM and gtx 1080 on ubuntu 18.04 platform.
2. Virtual machines with i7-7700HQ with 4 cores, 12gb RAM and mx150 gpu on ubuntu 20.04 platform.

Machine learning models are built and tested using the Jupyter notebook and scikit-learn, matplotlib, pandas and numpy libraries are used in this experiment. Mininet-Optical emulator have provided official walkthrough for installing and running the emulator on the system which further instruct users to install dependent libraries such as

Mininet, Open vSwitch and SDN controller. As the tool uses veth and network namespaces for emulating the nodes, the users will require an administrative/root access to the system to execute and build the topology.

4.2 Topology generation using Mininet-Optical emulator

With the help of REST API, it become easier to write and test an algorithm for building a topology of desired configurations. Initially, a ring lattice topology was created but to make it easier for relating with the previous studies, the topology was switched to linear topology with the similar behaviour of ring lattice. Topology with 20 nodes is created with optical devices such as EDFA amplifiers, transceivers, ROADMs and each connected with optical links. For this study, each connection is loaded with 80 channels and the OSNR, gOSNR and ASE are collected for each channel of each connection. To connect each channel, the terminals are configured with 80 transceivers and monitoring mode are enabled for incoming traffic for capturing required statistics. Four amplifiers are added between two neighbouring ROADM nodes with each with random ripple function. Mininet-Optical supports three ripple functions which has predefined variations for each channel – linear, wdg1 and wdg2. Wavelength dependent gain (wdg) is a critical parameter in erbium-doped fiber amplifiers and the primary determinant of channel power divergence and excursions in optical transmission systems, both of which vary with channel loading in wavelength-division-multiplexed (WDM) systems. In an optical transmission system, measurements of the wavelength dependent gain between two locations can be used to estimate optical power excursions that occur during optical circuit switching. Because of the dynamic nature of the topologies, custom data structures have to be used to store the metadata information's of the link mapped between the ROADMs for creating a switch rule for all the connections.

Following are the sample snippets used for generating the topologies supported for this study, description of each snippet are provided as followed:

```
def build(power=0 * dBm, N, k, p, connection=[]):
        halfk = k // 2
        ch_link = 80
        links = neigh_list = neigh_metadata = roadm_links = {}
        neigh_graph = nx.Graph()
        seed = np.random.RandomState(42)
        nodes = list(range(1, N + 1))
```

```
rparams = {'monitor_mode': 'in'}
transceivers = tuple((f'tx{ch}', power) for ch in range(1,
                                        ch_link + 1))
tparams = {'transceivers': transceivers, 'monitor_mode': 'in'}
for node_num in range(1, N + 1):
    self.addSwitch(f'r{node_num}', cls=ROADM, **rparams)
    self.addSwitch(f't{node_num}', cls=Terminal, **tparams)
    self.addNode(f's{node_num}', cls=LinuxRouter)
    self.addHost(f'h{node_num}')
boost = ('boost', {'target_gain': 17 * dB})
spans = []
for c in range(1, 5):
    ripple_func = random.choice(list(ripple_functions.keys()))
    aparams = {'target_gain': distance * km * .22, 'wdg_id':
                                          ripple_func}
    spans.extend([distance * km, (f'amp{c}', aparams)])
```

To preserve the temporary topology-creation functions, their auxiliary methods, and the routing algorithm, a class called **LinearTopology** was developed. The snippet above is utilized to map the topology according to the inputs given to this function. The metadata of all connections that need to be initiated, which will have the information of starting node and destination node required for calculating the routing path between them, are also needed by this function. The topology's total number of nodes, **k**, representing the ROADM devices' closest neighbours, Watts-Strogatz probability, **p**, and metadata are all required. The initial stage of this approach generates a random seed variable and a list of all host nodes in the topology, which are then utilized in the Watts-Strogatz algorithm for comparison with the given probability and another for selecting the random node. The strategy for storing the data necessary for route discovery is crucial for the routing algorithm to operate flawlessly with minimal time complexity. On the algorithm's efficiency, it will have a noticeable effect. By feeding the information of the nodes and the vertices/edges connected to the neighboring nodes into the graph, a non-linear data structure, the structure of the topology is mapped (ROADM devices). The shortest-path technique was made simpler to implement with the aid of the graph data structure. In this topology, each connection will be loaded with 80 channels to simulate the wavelength load and to support this, 80 transceivers are required to be configured in the line terminal to support the transmission of all 80 channels. Both the ROADM and the line terminal elements are configured with monitoring mode enabled for incoming connections. For loop is used for automating the creation of **N** number of host nodes, **ROADMs**, **line terminals**

and **switches** required for building the topology. The switches in the Mininet-Optical tool are emulated by creating a virtual node with IP forwarding enabled.

The boost parameters are set for configuring the target_gain achieved from the boost amplifier residing between the neighbouring ROADM connections. In this case, the target_gain is set to 17, denoted in the **dB** unit. Four amplifiers are configured between the neighbouring ROADM nodes to amplify the signal over long distances. The following information is needed to generate a span variable that can be used to setup each ROADM with the same property. The connection length will be expressed in kilometres (km), the configuration parameter for each amplifier will have target gain calculated depending on the distance, and the ripple function will be chosen from the Mininet-Optical tool's linear and wavelength-dependent gain (wdg) default implementation. Using the random module's decision approach, the ripple function is randomly chosen for each amplifier. Each amplifier is designed for a distance of 17 kilometres in this study, and to preserve consistency, the topology will remain the same throughout. The distance between the two nearby ROADMs in this instance will therefore be $17 * 4$ km.

```
for i = 1 to N:
  for j = 1 to halfk # for ROADM <-> ROADM connection
  for port = 1 to 80 # for bidirectional connection between terminal <->
                                        ROADM
  for port = 1 to 80 # for bidirectional link eth connection between
                                       terminal <-> switch ports
  self.addLink(f'h{i}', f's{i}', port2=ch_link + 1) # ethernet link
                                       between host <-> switch
```

Within the initial for loop used for creating the N number of nodes, three separate for loops are used for building each network element and its connections. Later an ethernet link connection between host node and the switch is established. This complete flow is responsible for the topology building process and the breakdown of each for loop will be discussed below.

4.2.1 ROADM $\longleftrightarrow$ ROADM connection

While building the connections between the neighbouring ROADMs, it was crucial to store the port information of both the node links and the port number, which get used later for creating a switch rules. If this information is not updated properly, the algorithm will create a partial switch rule and the channels won't be transmitted to the destined node and get dropped. For storing this information, a simple object class is created which will have the following variables –

1. **node_id** denotes the identifier assigned to the current node.
2. **neigh_id** denotes the identifier of the neighbouring roadm node.
3. **lineout** is a list of outgoing ports connected to the neigh_id.
4. **linein** is a list of incoming ports in established from the node_id.
5. **reverse** indicating whether the connection is in the reverse order i.e., neigh_id is connected to the node_id, instead of vice-versa.

The same class provide a getter function to retrieve the information of single link connection which will have linein and lineout. It is usually called while building a routing connection information.

```
for j in range(1, halfk + 1):
  neig_node = i % N + j
  neig_node = neig_node if neig_node <= N else neig_node - N
  if neig_node == 1 and i == N:
    print(f"last node connection {i} {neig_node}")
    continue
  neigh_node = self.watts_strogatz_calc(i, neig_node, nodes, p, seed)
  roadm_links.setdefault(i, []).append(neigh_node)
  roadm_links.setdefault(neigh_node, []).append(i)
  links[f'r{i}'] = links.get(f'r{i}', {'linein': 1, 'lineout': 2})
  links[f'r{neigh_node}'] = links.get(f'r{neigh_node}', {'linein': 1, '
                                      lineout': 2})
  lineout = links[f'r{i}']['lineout']
  linein = links[f'r{neigh_node}']['linein']
  neigh_list.setdefault(f'r{i}', []).append(f'r{neigh_node}')
  neigh_graph.add_edge(f'r{i}', f'r{neigh_node}')
  linein_con = []
  lineout_con = []
  for _ in range(5):
    self.addLink(f'r{i}', f'r{neigh_node}', port1=lineout, port2=linein,
                                       boost=boost, spans=spans, cls=OLink
                                      )
    linein_con.append(linein)
    lineout_con.append(lineout)
    lineout += 2
    linein += 2
  connected_node = NodeInformation(f'r{neigh_node}', f'r{i}', linein_con
                                      , lineout_con)
  neigh_metadata.setdefault(f'r{i}', []).append({f'r{neigh_node}':
                                      connected_node})
```

```
links[f'r{i}']['lineout'] = lineout
links[f'r{neigh_node}']['linein'] = linein
```

For this study, the topology with four neighbouring nodes is configured. Above snippets are used for enabling this configuration. The neighbour node is calculated based on the expression **i%N+j** to avoid the self-looping or connecting to a wrong node and if check statement is used. To randomize the topology, the Watts-Strogatz function is called, which will be discussed later in this section and based on the pre-defined probability value, a new neighbouring node is picked. A **roadm_link** variable is used for storing the nodes and their neighbouring node's information. This information is later used in the calculation of the Watts-Strogatz algorithm. Each node's used linein and lineout link information is also maintained in a **links** variable to prevent the link used to connect neighbouring nodes from being used twice. At the same time, the updated linein and lineout port information is retrieved from the same variable, which is later used to establish the connections. The **neigh_graph** variable is the graph data structure mentioned before used for maintaining the topology metadata used for building the shortest routing path. As the connections are getting loaded with 80 channels, to avoid the overlapping of the transmission of the channels between the nodes, five wdm links are created between each neighbouring ROADMs.

When switch rules are configured, the linein and lineout port are mentioned for both the current node and the adjacent node where the connection are getting terminated, the rule for each channel are set for the path because of which if there are one more connection going through the same wdm link with the same channel configuration, then there are high chances that the signal will get dropped by the roadm node. To avoid this, five links are created. The linein and lineout port information of each this link are stored in the NodeInformation class object to make it easier to retrieve the same information while configuring the routing path. Both the linein and lineout ports of these five connections should be unique and should not overlap with each other. The **NodeInformation** object created earlier are stored in the **neigh_metadata** variable which is later passed on to the **get_connection_detail** function for building a routing path between the starting and destination nodes.

4.2.2 Terminal ⟷ ROADM and Switch ⟷ Terminal connection

```
(a)
for port in range(1, ch_link - 1):
  # Bidirectional terminal <-> roadm optical links
  self.addLink(f't{i}', f'r{i}', port1=port+2, port2=roadm_line + port +
                                    2, spans=[1*m], cls=OLink)
```

```
(b)
for port in range(1, ch_link + 1):
  # Terminal<->switch ethernet links
  self.addLink(f's{i}', f't{i}', port1=port, port2=ch_link + port)
```

(a) This snippet is used to establish 80 bidirectional connections between the roadm and the terminal element, which are placed within a span of one meter.

(b) Between the switch and the host, the same number of ethernet connections as earlier are established.

```
for conn_node in connection:
            get_path = self.bfs(neigh_graph, f'r{conn_node["start"]}', f
                                                'r{conn_node["end"]}')
            connection_detail.append(self.get_connection_detail(get_path
                                                , neigh_list,
                                                neigh_metadata, f'r{
                                                conn_node["start"]}'))
```

Two functions are called to populate the shortest path information between the communicating nodes. Eight connections are established for this study and for each connection, the **bfs()** and **get_connection_detail()** functions are called. These functions will be discussed later in the below section.

Figure 4.2: Linear topology with $p = 0$, $k = 4$, and $N = 20$

Using the functionality previously discussed, the above linear topology 4.2 with 20 nodes is created. Each roadm is connected to the neighbouring roadm nodes with 5 WDM links, and the roadm is connected to the line terminal with 80 links, as the transceiver needs for each channel loaded in the connection. The connections between the terminal and the switch are the same. The connection between the switch and the host can be established with just one connection.

4.3 Watts-Strogatz algorithm

For this study multiple sets of topologies are required, for generating the network topologies, Watts-Strogatz algorithm is implemented, in which the existing topologies are modified based on the probability value set during the execution. Below is the implementation of the algorithm used for this study –

```
def watts_strogatz_calc(curr_node, neigh_node, nodes, p, seed):
  if seed.random() < p:
    # to avoid loop connection
    choices = [e for e in nodes if e not in (curr_node, neigh_node)]
    new_neigh_node = seed.choice(choices)
    if curr_node in roadm_links:
      if new_neigh_node in roadm_links[curr_node]:
```

```
        return neigh_node
      if new_neigh_node in roadm_links:
        if curr_node in roadm_links[new_neigh_node]:
          return neigh_node
    return new_neigh_node
  else:
    return neigh_node
```

The algorithm requires a set of parameters to calculate whether the connection needs to be re-established with new node. Current node and current adjacent node identifier, list of existing nodes, probability value set in the initial phase of the execution, and the random seed variable. If the value generated by the random seed variable is greater than the probability value, the adjacent node is returned as it is, and the connection is not modified. If it is less than the probability value, the new node is selected from the list of nodes from which current node and the old adjacent node are removed from the same to avoid self-looping. Before returning the neighbouring node identifier, certain checks are made to prevent the error of linking two existing neighbouring nodes. The present roadm node and the recently formed neighbouring roadm node are subject to the same conditional check by the algorithm. Only the new neighbour node identifier will be returned to the calling method if all requirements are met; else the old adjacent node identifier will be returned.

4.4 Configuring Dynamic routing algorithm

4.4.1 Implementing BFS algorithm

For calculating the shortest path between the starting and destination node, Breadth-First search algorithm is implemented by using a queue data structure for storing the traversed node information for avoiding self-looping scenario.

```
    def bfs(graph, node, target):
        visited = actions = []
        queue = Queue()
        start_node = node
        if start_node == target:
            return []
        queue.push((start_node, actions))
        while not queue.isEmpty():
```

```
            node, action = queue.pop()
            if node not in visited:
                visited.append(node)
                if node == target:
                    return action
                for neighbour in graph[node]:
                    new_action = action + [neighbour]
                    queue.push((neighbour, new_action))
        return []
```

Two variables - **visited** and **actions** are maintained within the functions for tracking the visited node identifier and later for returning the hops/nodes it traversed to reach the destination. In each iteration of the list of neighbouring nodes, if the node information is popped from the queue and added to the visited list and the action variable is updated with the new path/node information and returned to the calling function once the destination node is reached, else null route is returned. This function plays a base for building the routing path information between the connections. Later this information is passed to **get_connection_details()** function to generate the switch rule required for configuring the path in the optical transmission devices such as line terminal and roadm's.

4.4.2 Generating switch rules information required by the route configurating functions

ROADM acts as a routing mechanism for routing the transmission signal between the roadm devices. linein and lineout port information is required along with the channel that needs to be enabled while configuring the switch rules. Below is the implementation used for building the routing information required for building the switch rules with the help of hops information calculated using the BFS algorithm.

```
    def get_connection_detail(shortest_path, neigh_list, neigh_metadata,
                                            initial_node):
connection = []
for path in shortest_path:
  if path in neigh_list[initial_node]:
    neigh_node = neigh_metadata[initial_node]
    reverse = False
    find_node, node = path, initial_node
  elif initial_node in neigh_list[path]:
    neigh_node = neigh_metadata[path]
```

```
      reverse = True
      find_node, node = initial_node, path
    else:
      break
    neigh_obj = [list(k.values())[0] for k in neigh_node if list(k.keys
                                        ())[0] == find_node][0]
    lineout, linein = neigh_obj.get_link()
    if reverse:
      find_node, node = node, find_node
    node_info = NodeInformation(node, find_node, linein, lineout,
                                        reverse)
    connection.append(node_info)
    initial_node = path
  return connection
```

For the purpose of determining the connection path between the originating node and the destination node, the function **get_connection_detail()** needs a set of input parameters. The BFS algorithm determines the shortest path, the **neigh_list** kept by the topology building function previously mentioned 4.2, and the **neigh_metadata** variable containing the link connection information, including the linein and lineout port numbers between the roadm connection stored during topology building activity. Finally, the **initial_node** designating the starting node of the connection is all needed The **linein** and **lineout** port numbers are retrieved by iterating over each node/hop present in the **shortest_path** variable and checking a set of conditions. The **reverse** flag indicates whether the connection between the nodes is in the reverse direction. For example, if r2 is connected to r1 instead of r1 to r2, the port numbers linking the nodes will be completely different, as will the initial node and destination node identifiers. After these calculations have been made, the **linein** and **lineout** are retrieved using the **getter** function written in the **NodeInformation** class. This function will select the **linein** and **lineout** port number from the array of ports recorded during the construction of the topology connections. This retrieved information along with the roadm's identifier are stored in a new **NodeInformation** object which will be kept in a **connection** array referred while configuring switch rules and returned to the calling function.

Following the generation of the topology object **(LinearTopo)** and the **connection** metadata, the Mininet-Optical emulator's REST server is launched, which will process any API requests sent to the SDN controller for handling operations such as installing, deleting, or resetting rules, retrieving a list of nodes, links from terminals, roads, routers, and switches, and updating the amplifiers' properties. The **RESTProxy** object is also

created for monitoring the reading generated on the line terminals for each channel. This object is passed as a parameter to the **monitorOSNR()** function which will be discussed later in this section. Once all the API related functionality is enabled, graphical plot of the topology is plotted with the help of **plotNet()** function which uses **matplotlib** and **pygraphviz** library. After visualization, the next task is to configure the switch rules required by the roadm element for transmitting the optical signal. This process is elaborated in the next section.

4.4.3 Configuring optical switching in ROADM devices

Before starting the transmission of the optical signals between the connections, switch rules need to be configured in the roadm nodes for forwarding the signals coming from the specific ports to be forwarded to other destined ports. Below is how configuration function are called for configuring the network connections for the topology built for this study –

```
        for ch in range(channels_length):
        count = 0
        for end_conn in conn:
            configNet(net, connection_detail[count], end_conn['start'],
                                                    end_conn['end'], counter
                                                    , [channels[count][ch]])
            end_conn['ch'].append(channels[count][ch])
            count += 1
        start_transceiver(net, conn)
        monitorOSNR(requestHandler, conn, connection_detail)
        counter += 1
```

The 80 channels in each connection are randomly ordered, and the same channel lists from each connection are utilized to construct the metrics for all the topologies mentioned in this study. This keeps the metrics consistent and makes it simpler to compare the effects of topology on the metrics. The channels list of all eight connections are kept in a CSV file so they can be referenced to at a later time for all other topologies. In order to simulate the wavelength loads on each connection and to monitor the influence on OSNR metrics, each channel from this list of 80 channels is loaded into the connections one at a time, until all the 80 channels are loaded in the connections. In the above implementation, a nested hashmap data structure is used for storing the detail of the connections, such as starting node, destination node, a file pointer for the CSV file and an array of channels

currently loaded in the connection. An iterator object with a range of 80 channels is created and iterated for every eight connections. Within the iteration of connections, the switch rule configuration function and a function for monitoring OSNR and other required metrics are called, and the readings are noted. At the same time, the loaded channels in each iteration are updated in the array of channels which are later used for populating the CSV file.

The implementation for configuring the switch rules for configuring the route for the transmission of the optical signals is stated below –

```
def configNet(net, connection, start, end, ctr, ch):
N = net.topo.N
channels = ch
defaultEthPort = 90
defaultWDMPort = 2
counter = ctr
# Terminal hostport<->(uplink,downlink)
for ch in channels:
    ethPort = defaultEthPort + counter
    wdmPort = defaultWDMPort + counter
    net[f't{start}'].connect(ethPort=ethPort, wdmPort=wdmPort,
                                        channel=ch)
    net[f't{end}'].connect(ethPort=ethPort, wdmPort=wdmPort, channel
                                        =ch)
    counter += 1
# Configuring ROADM to forward ch1 from one terminal to other e.g.
                                    t1 to t2"
for index, conn in enumerate(connection):
    counter = ctr
    for ch in channels:
        terminal_port = neigh_forward_port = counter + roadm_line +
                                            2
        counter += 1
        node = conn.node_id
        neigh_node = conn.neigh_id
        default_linein = conn.linein
        default_lineout = conn.lineout
        if conn.reverse:
            default_lineout, default_linein = default_linein,
                                                default_lineout
        if index != 0:
            if (conn.reverse and connection[index - 1].reverse) or (
```

```
                                                    not conn.reverse and
                                                     connection[index -
                                                    1].reverse):
                terminal_port = connection[index - 1].lineout
            else:
                terminal_port = connection[index - 1].linein
        if index != (len(connection) - 1):
            if conn.reverse and connection[index + 1].reverse:
                neigh_forward_port = connection[index + 1].linein
            elif connection[index + 1].reverse:
                if conn.reverse:
                    neigh_forward_port = connection[index + 1].
                                                            lineout
                else:
                    neigh_forward_port = connection[index + 1].
                                                            linein
            else:
                neigh_forward_port = connection[index + 1].lineout
        net[node].connect(terminal_port, default_lineout, channels=[
                                                ch])
        net[neigh_node].connect(default_linein, neigh_forward_port,
                                                channels=[ch])
```

This function makes use of the previously generated shortest path with a list of hops, switch rule information needed for configuring the rules that were generated by the previously mentioned function (add a reference to it), the starting and destination nodes of each connection, Mininet object as a reference for calling the necessary class functions, counter variable which is used to pick a unique port for each channel, and channel number that needs to be added to the transmission signal. Since 80 channels will be loaded and a different port must be used for each channel, the default **EthPort** and default **WdmPort** numbering start with 90 and 2 correspondingly to prevent port number overlap. The default port is combined with the counter variable to create a unique port for both connecting nodes. The switching rule is applied in the terminal device's **connect()** function, where the ethPort, wdmPort, and channel number parameters must be given. Once the line terminal has been established with the rules, the roadm must be configured with the switch rules in order to route the signals to the intended node. In order to update the proper flow of the port numbers, the previously created hop information's is iterated in order, containing the linein and lineout port numbers. The rules are also configured using the connect() method after a few conditions are met. When configuring the rules,

the reverse flag is used to determine the flow of signals between the two connecting nodes, based on which the linein and lineout port numbers of both nodes are recalculated and updated.

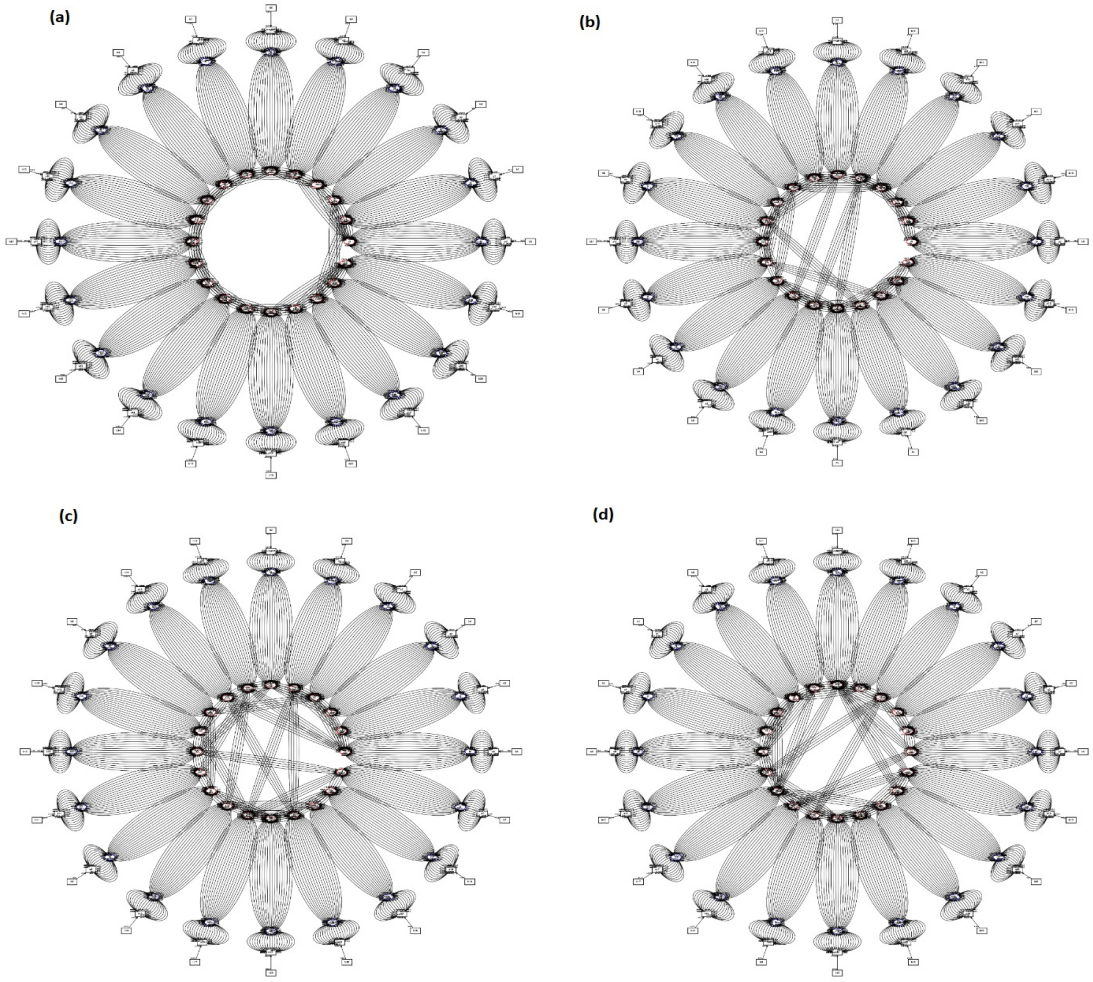

Figure 4.3: Topologies generated by the Mininet-Optical emulator

Figure 4.3 (a). A linear topology with probability $\beta = 0$ in which each node is connected to the same number of nearest neighbours on either side. (b). A linear topology with probability $\beta = 0.32$ (c) A linear topology with probability $\beta = 0.50$ (d). A linear topology with probability $\beta = 0.75$

4.5 Dataset collection using OPM function

Once the routing rules are configured on both the roadm's and the line terminal present in the connection line, line terminal are switched on and the signal transmission are

initiated. The QoT is monitored using the OPM with the help of REST APIs exposed by the controller. Below is the implementation done for the collecting the metrics for each eight connection –

```
def monitorOSNR(request, conn, conn_detail):
for conn_index in range(0, len(conn)):
    start = conn[conn_index]['start']
    end = conn[conn_index]['end']
    writer = conn[conn_index]['file']
    ch_enabled = conn[conn_index]['ch']
    path = get_path_distance(conn_detail[conn_index])
    start_response = request.get('monitor', params=dict(monitor=f't{
                                        start}-monitor', port=None,
                                        mode='in'))
    end_response = request.get('monitor', params=dict(monitor=f't{
                                        end}-monitor', port=None,
                                        mode='in'))
    start_osnr = start_response.json()['osnr']
    end_osnr = end_response.json()['osnr']
    # populate csv row with initial value
    csv_row = [0] * channels_length
    csv_row = [1 if x + 1 in ch_enabled else y for x, y in enumerate
                                        (csv_row)]
    for (channel1, data1), (channel2, data2) in zip(start_osnr.items
                                        (), end_osnr.items()):
        s_osnr, s_gosnr = data1['osnr'], data1['gosnr']
        e_osnr, e_gosnr = data2['osnr'], data2['gosnr']
        s_ase, e_ase = data1['ase'], data2['ase']
        e_power_dbm = abs_to_dbm(data2['power'])
        row = copy.deepcopy(csv_row)
        row.extend([channel1, e_power_dbm, path, s_ase, e_ase,
                                            s_osnr, , e_osnr,
                                            s_gosnr, e_gosnr])
        writer.writerow(row)
```

Along with the **RESTProxy** object, **connection_detail** variable, and an array of all eight connections, they are supplied as parameters. A REST API request is sent to the controller using the **RESTProxy** object, and the **connection_detail** variable contains all the hop-specific data needed to calculate the distance between the beginning and the target nodes. The dataset needed for the machine learning model's training and

validation is filled up using this function. The dataset contains the following data that was obtained from the emulator: 0 to 80 channels with values of one or zero indicating whether a particular channel has been enabled, the current channel number, signal power in **dBm**, the distance between the start and destination nodes in **km**, amplified spontaneous emission (ASE) noise, ONSR, and gOSNR values in **dB**. The readings are gathered and saved in a CSV file for each channel that has been enabled when it has been loaded onto the connection. Each of the eight connections has a separate CSV file. Each dataset for the connection is of **3241**, and the same process is performed with topology of different probability value.

Below table describe the features and its datatype/unit present in the datasets –

feature	**Description**
Loaded_channels	Channels currently loaded in the connection (Boolean)
channel	Id of the new channel loaded to the connection. (integer)
e_power_dbm	The power of the optical signal observed in the connection (float/dBm)
path	Total distance between the start and the destination node. (float/km)
s_ase	ASE noise value registered at the starting node (float)
e_ase	ASE noise value registered at the destination node (float)
s_osnr	The reading ONSR value registered at the starting node (float/dB)
e_osnr	The reading ONSR value registered at the destination node (float/dB)
s_gosnr	The reading gONSR value registered at the starting node (float/dB)
e_gosnr	The reading gONSR value registered at the destination node (float/dB)

Table 4.1: Metrics collected using the OPM APIs

4.6 Implementation of ML model for predicting QoT

The following section describes the data transformation steps applied for selecting and engineering the features for achieving the best prediction model. Once dataset is cleaned, the dataset is split into training and testing data.

4.6.1 Dataset Pre-processing

The pandas and NumPy modules are used for loading, understanding, and preprocessing the dataset required for the model. The dataset of each topology segregated based on probability value are kept in a separate directory having eight dataset - CSV files for each connection. The model for each topology is trained and evaluated separately. The CSV

file of each topology is loaded and aggregated into a single dataset. To comprehend the influence of the likelihood and its impact on the connections of the topology, the hop count of each link is determined. The below figure indicates that with the increase in the probability value of the Watts-Strogatz algorithm, the hop count for the connection's decreases, which will also decrease the distance between the connections. The probability with a value of 0.0 has the most hop count and decreases compared with the topology of p=0.32 4.4. This dataset is split into feature and target category in which 0-80 channels, current channel (ch), power, path and source ase noise (s_ase) are considered as the feature set while reading of OSNR and gOSNR in the destination node are considered as target dataset. This process is used for processing the datasets for linear regression, ridge regression and random forest regressor model. This process is not viable while training the neural network model.

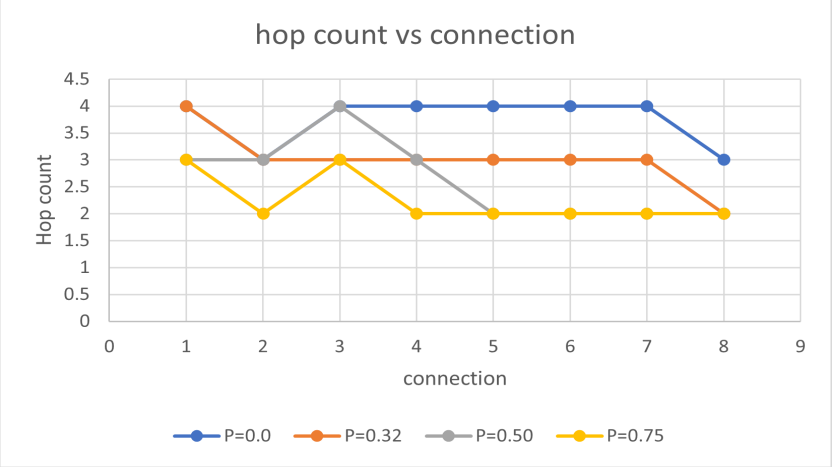

Figure 4.4: Plot for hop count of each connection in each topology

For preprocessing the dataset for the neural network model, the needs to be normalized and reshaped before start training the model.

```
from sklearn.preprocessing import StandardScaler
PredictorScaler=StandardScaler()
TargetVarScaler=StandardScaler()

# Storing the fit object for later reference
y = y.reshape(-1,1)
PredictorScalerFit=PredictorScaler.fit(X)
TargetVarScalerFit=TargetVarScaler.fit(y)
```

```
# Generating the standardized values of X and y
X=PredictorScalerFit.transform(X)
y=TargetVarScalerFit.transform(y)
```

The **sklearn** library provides a function StandardScaler to standardize the dataset by rescaling the distribution of values to make standard deviation to one and mean of observed values to 0. This process is performed to both feature and target dataset. To perform the standardization process, the fit() and transform() function is used for computing the mean and standard deviation and later for performing standardization by centering and scaling.

Once the feature and target dataset are preprocessed using respective methods, the model building process is initiated. The model is created in two different ways: first, by splitting the entire dataset into training and validating sets with an 80/20 ratio, and second, by training and validating the model by splitting the set by increasing the dataset's ratio to understand the trade-off. In this case, the model is trained over a number of iterations by first taking into account 20% of the entire dataset and then splitting that dataset into an 80/20 ratio; after each iteration, the amount of data to be used keeps on increasing until it uses the entire dataset.

4.6.2 Architecture of the Neural Network Model

There are multiple supported frameworks/libraries for implementing neural networks. This study will use **TensorFlow** and **Keras** to build the neural network prediction model. Both are open-source machine learning libraries offering high flexibility, portability and performance among multiple platforms. The neural network architecture is composed of four types of layers, namely – fully connected, convolution, recurrent and deconvolution layer. A neural network model with fully-connected layers is used for this study which will connect every neutron in one layer to every neuron in the next layer. The neural network requires an activation function for transforming the weighted input from the node into the output or activation of the node. The rectified linear activation function (ReLU) is used here, this function returns zero if it receives negative input and for any positive value it returns that value back.

```
normalizer = layers.Normalization(axis=-1)
normalizer.adapt(X_train)

model = keras.Sequential([
```

```
  normalizer,
  layers.Dense(84, activation='relu', name='fc1'),
  layers.Dense(256, activation='relu', name='fc2'),
  layers.Dense(128, activation='relu', name='fc3'),
  layers.Dense(128, activation='relu', name='fc4'),
  layers.Dense(128, activation='relu', name='fc5'),
  layers.Dense(1)
])
model.compile(loss=customLoss,
              optimizer=tf.keras.optimizers.Adam(learning_rate=0.01))
model.summary()
history = model.fit(X_train, y_train, batch_size=32,epochs=50,verbose=2,
                    validation_data=(X_test, y_test))
```

The model is configured with normalized layer then dense layer with custom shape of input data, ReLU as an activation function and name for identifying the layer. The learning rate must be carefully chosen because it affects the size of weight updates in the neural network. If the rate is set too low, training will proceed slowly because it makes very small updates to the weights, but if the rate is set too high, it can result in unfavorable divergence in the loss function. The model is constructed with the custom loss functions that are discussed below and an Adam optimizer with a learning rate of 0.01. The model is trained by calling the **fit()** function on the model with batch_size of 32 and 50 epochs.

```
    def customLoss(y_actual, y_pred):
  # Calculate the number of loaded channel
  no_loaded_channels = tf.dtypes.cast(TFmath.count_nonzero(y_actual),
                                      tf.float32)
  # Set the values of the unloaded channels to zero
  modified_y_pred = TFmath.divide_no_nan(TFmath.multiply(y_pred,
                                      y_actual),
                                         y_actual)
  # Calculate the loss
  error = TFmath.abs(TFmath.subtract(modified_y_pred, y_actual))
  loss = TFmath.divide(TFmath.reduce_sum(error), no_loaded_channels)
  return loss
```

Instead of using the existing mean absolute error provided by the tensorflow, custom loss function is used to calculate the actual mean absolute error based on this datasets and model.

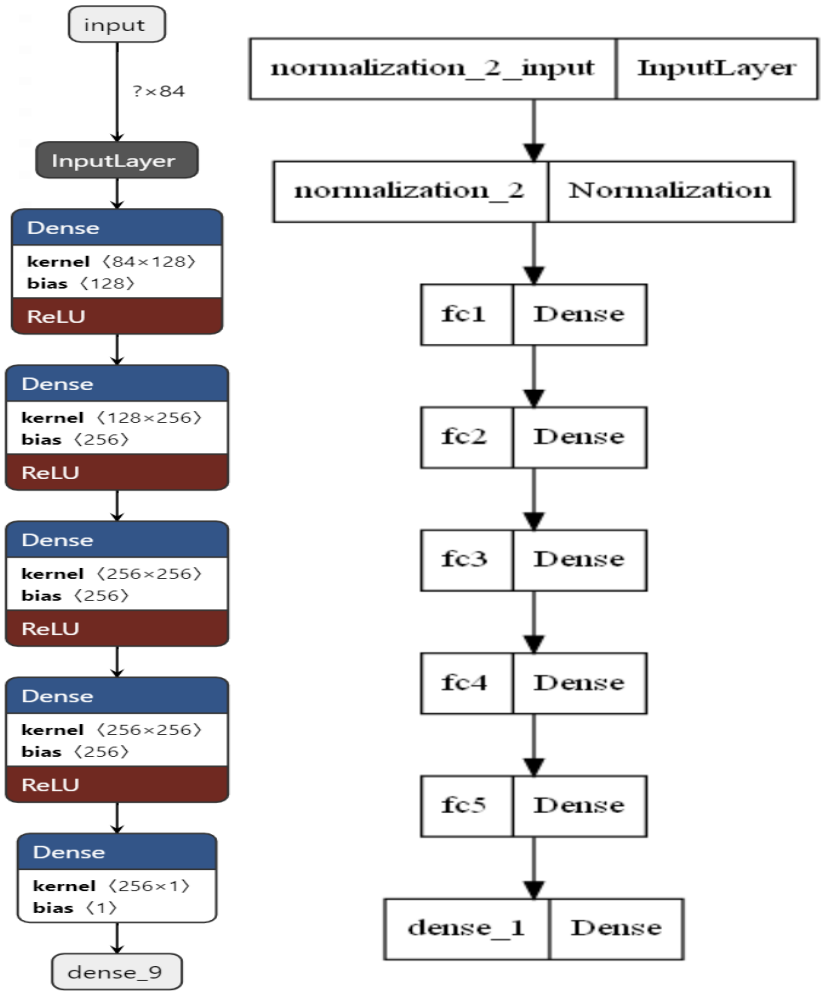

Figure 4.5: Architecture of the neural network layers

The above figure 4.5 provides the summary of the network model built using the above snippet which explains the name and type of the layer used and output shape of each layer and input each layer receives.

4.7 Summary

In this section, implementation of the Mininet-Optical emulator, Watts-Strogatz and BFS algorithms are elaborated. The process of selecting the dataset and preprocessing the dataset is discussed here. The next section will elaborate about the hyperparameters used for each machine learning models, results and evaluation of the ML model.

Chapter 5

Evaluation

The outcomes of the evaluation procedures outlined in the methodology 3.2 are reported in the subsequent chapter. There will also be a discussion of the findings and information on the key methods through which they were attained. It will be explained how the experiment relates to other areas where the methodology excels as well as potential future applications that might further enhance the findings. After the results, there is a discussion segment where we will elaborate on how we compared all the output.

5.1 Selection of hyper parameters

Before training the model, it is necessary to identify and fine-tune the hyper-parameters because their values vary for each model. Selected sets of potential hyper-parameters are iterated over with the same datasets to determine their accuracy and error in order to obtain hyper-parameters that fit the data more effectively. For these datasets, the findings with the best hyper-parameters are chosen to be used in the models. Both the Random Forest and Ridge regression regressor models are used in this process.

The values selected are shown in table 5.1.

Model	**Hyperparameter values**
Linear regression	N/A
Ridge regression	0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05
Random forest regressor	100, 200, 300, 1000

Table 5.1: The hyperparameters tested for each model

The same dataset is used for both training and validation of the Ridge and Random Forest regressor models for each selected potential hyperparameter value. In order to comprehend and choose the hyperparameter with the lowest error value, the error is determined using the RMSE method. According to the figure 5.1, the value **0.05** is optimal when considering alpha penalty for the Ridge model, and the value **400** is the best estimator for the random forest regressor model.

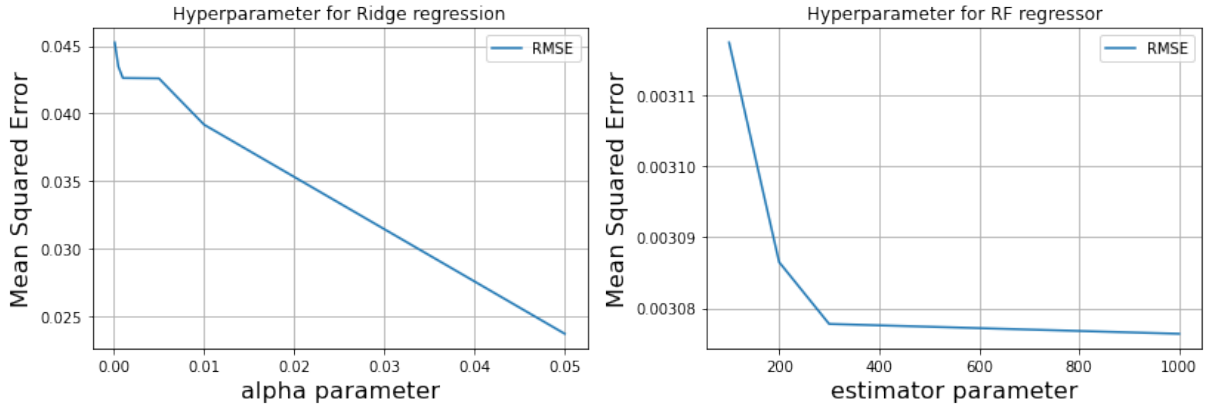

Figure 5.1: Error outcome for each hyperparameter value

5.2 Evaluation and comparison of Machine learning approach

Data correlation techniques are used to understand how one set of data corresponds to another set of data in order to comprehend the impact of features on the target dataset. This mechanism aids in determining which features should be chosen and which may cause significant damage during the fitting model. The **corr** functions provided by pandas aid in calculating the pairwise correlation of columns. The calculated values are fed into the **heatmap** function from the **searborn** library.

From figure 5.2 it is evident that the path distance, current loaded channel and ase values have one of most correlation with the OSNR and gOSNR value recorded at the destination node. The Pearson Correlation Coefficient is used by the **corr** function to determine the degree to which two sets of data are linearly correlated. It is calculated as the covariance of the two variables divided by the product of the standard deviation of each data sample, i.e., it is a normalized calculation of covariance between the two variables that falls between the range of -1 to 1.

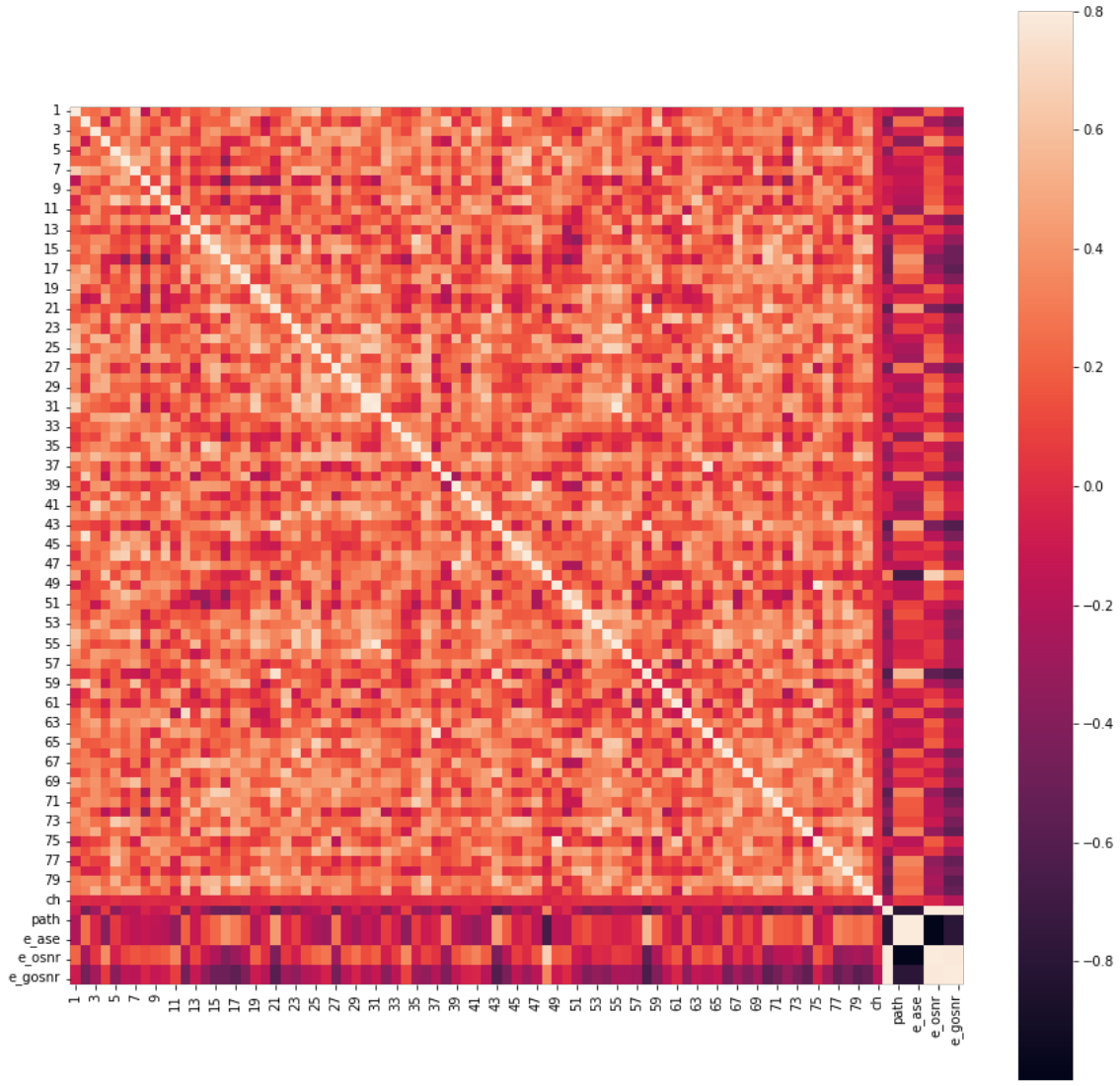

Figure 5.2: Pearson's correlation coefficient heatmap of the metrics dataset

The result achieved from each model is discussed in this section.

5.2.1 Evaluation of regression models

To evaluate the performance of the each model, set of earlier discussed 3.2 evaluation metrics are analysed to understand the error between the predicted and true value. Four models were trained separately for each topologies to understand the impact of having different topologies on the predictivity of the model. For each topology model, the dataset of eight connections are combined to one dataset with 25920 entries. This combined dataset are split in ratio of 80/20, 80% data is used for training the model and rest 20%

dataset for testing and validating the model.

Probability	Linear Reg			Ridge Reg			Random Forest		
	score	RMSE	MAE	score	RMSE	MAE	score	RMSE	MAE
p=0	0.998	0.036	0.019	0.993	0.197	0.142	0.999	0.003	0.001
p=0.32	0.997	0.116	0.023	0.982	0.218	0.148	0.999	0.003	0.002
p=0.50	0.995	0.161	0.035	0.978	0.188	0.115	0.999	0.003	0.002
p=0.75	0.995	0.172	0.045	0.979	0.123	0.206	0.999	0.003	0.002

Table 5.2: The observed outcome of the regression model for each topology.

This table 5.2 contains the findings from all the models that were utilized in the study. With a root mean square error (RMSE) error of 0.036 and a mean absolute error (MAE) error of 0.019, the topology for the linear regression model with probability β value set at 0 has the maximum accuracy of 0.998. In both the Ridge model and the random forest regressor, the topology with probability $\beta = 0$ has the highest accuracy and the lowest RMSE/MAE error when compared to the scores of other topologies. Despite the fact that the score falls as the Watts-Strogatz probability rises, it is clear that the error also goes down. In this case, the error represents the discrepancy between the observed/true value and the anticipated value; the lower the error, the better.

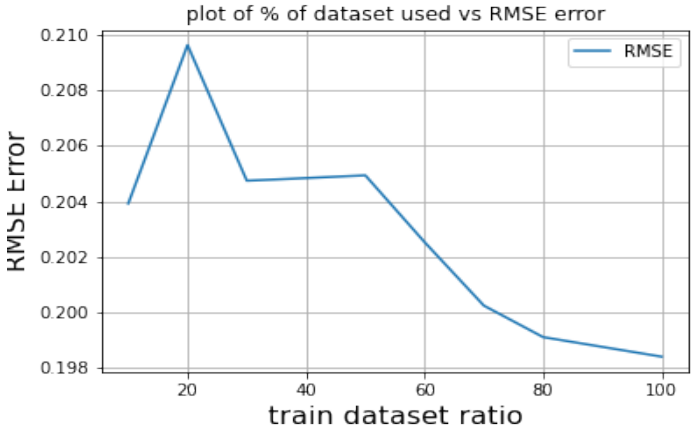

Figure 5.3: RMSE error against the test ratio of the dataset

The dataset is initially randomly divided with a ratio of 20/80, and that 20% dataset is further randomly divided into training and testing dataset in a ratio of 80/20, i.e., 80% data for training the model and other 20% for validating the model. This allows us to observe the error when the total amount of data used for training varied from 10% to

100% of its supposed size. It is evident from the figure 5.3 that as the amount of dataset used for training the model increases, the error also get decreases.

5.2.2 Evaluation of neural network model

	Neural Network		
probability	score	RMSE	MAE
p=0	0.999	0.020	0.014
p=0.32	0.999	0.020	0.016
p=0.50	0.999	0.022	0.018
p=0.75	0.999	0.023	0.017

Table 5.3: The observed outcome of the neural model for each topology.

The neural network is built based on the previously discussed architecture 4.6.2. The table 5.3 shows the observed results for each topology configured with different probability value β. The outcome from the neural network is the same with the regression model, the probability $\beta = 0$ provides the best accuracy and low error from both RMSE and MAE.

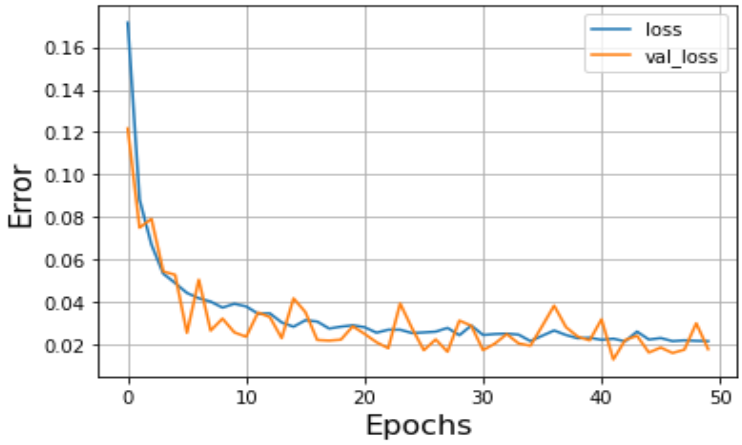

Figure 5.4: Line Plots of Mean Squared Error Over Training Epochs

The loss functions are used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next evaluation. For regression tasks, the mean of squared loss/differences between the observed and predicted value is employed, which is known as the Mean Squared Error (MSE) loss function. Because MSE is sensitive to outliers, it is often utilized when a goal value is typically distributed around a mean value and it is crucial to punish the outliers. In this investigation, the mean value and the target value are most frequently found together and calculate more accurately as per the

dataset, custom MSE loss function were used 4.6.2 to calculate the error more accurately. For both the training (blue) and validation (orange) sets, a line plot 5.4 is made to display the mean squared error loss over the training epochs. The model converged rather rapidly, and both test and train performance stayed about equal, however the contour of the error surface is not smooth, where modest changes to the weights are leading to slightly different changes in loss. The performance and convergence behavior of the model suggest that a neural network learning this problem might benefit from using mean squared error.

5.3 Summary

This chapter evaluated the design and the selection of hyperparameters used in training the model, the correlation between the features and targets, and the performance of each model prepared for four topologies. This research presents two case studies 5.2.1 done on multiple approach taken for training the models which gave a promising results.

Chapter 6

Conclusions & Future Work

The work done in this dissertation is outlined in this chapter, along with its key contributions. This chapter concludes by exploring potential future research topics and directions, as well as any limitations of this study.

6.1 Conclusion

In this thesis, the impact of topologies on the predictability of the Quality of transmission estimation is examined (QoT-E). To simulate optical communication topologies with optical devices including roads, amplifiers, line terminals, and links, Mininet-Optical emulator was utilized. This enabled to simulate optical transmissions and collect the necessary metrics needed as datasets for training machine learning models. Later these generated datasets were preprocessed and used for training the selected models developed for each topology with different Watts-Strogatz probability values. The results that were discovered were addressed in these section 5.2.1 5.2.2. The models were further divided into two categories: one set of models was trained using varying amounts of its total amount of datasets, ranging from 10% to its supposed size 5.3, and the other method used the total amount of datasets for each topology and randomly divided it into 80/20 for training and validating sets. It is observable from the results that with the increase in the probability of the Watts-Strogatz's probability β, the hop counts between the connections decreases as more links between the adjacent roadm nodes are randomly rewired, and the existing modified-linear topology becomes more leaned toward mesh-like topology. It also affected the precision of the models, along with the hop counts. The model's accuracy kept edging down a little bit, and the error it produced was also slightly increased.

6.2 Future Work

Since this research, was able to access the ability of machine learning algorithms in predicting the QoT-E on different topologies, there appears to be great potential for future research along many directions. Some of the limitations and the future possibilities associated with it have been discussed below:

1. Currently, the computations required to build the topology are high and consume a lot of time to successfully make the topology and transmit the signals between the nodes. With the aid of a more reliable setup and tool optimization, topologies including a greater number of nodes can be evaluated without necessitating a significant time investment in studying and troubleshooting the topologies.

2. The metrics gathered from this setup are insufficient to fully comprehend the impact of having multiple possibilities of connections in a single topology, which will have a direct impact on wavelength load as well. This study was conducted by enabling only eight unique connections on topologies of 20 nodes. Currently, 80 distinct connections are being formed for a single connection as a result of a single transmission creating a new unique link for each channel. If several connections are desired, 80xN links must be constructed from a single node, which will take a long time to develop the topology and make managing the connections difficult.

Bibliography

Abedifar, V., Shahkooh, S. A., Emami, A., Poureslami, A., and Ayoughi, S. A. (2013). Design and simulation of a roadm-based dwdm network. In *2013 21st Iranian Conference on Electrical Engineering (ICEE)*, pages 1–4.

Aiello, W., Chung, F., and Lu, L. (2000). A random graph model for massive graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC '00, page 171–180, New York, NY, USA. Association for Computing Machinery.

Barletta, L., Giusti, A., Rottondi, C., and Tornatore, M. (2017). Qot estimation for unestablished lighpaths using machine learning. In *Optical Fiber Communication Conference*, page Th1J.1. Optica Publishing Group.

Bollobás, B. (1985). Random graphs. London-Orlando etc.: Academic Press (Harcourt Brace Jovanovich, Publishers). XVI, 447 p. hbk: £ 52.00; $ 58.50; pbk: £26.00; $ 29.95 (1985).

Bouda, M., Oda, S., Vasilieva, O., Miyabe, M., Yoshida, S., Katagiri, T., Aoki, Y., Hoshida, T., and Ikeuchi, T. (2017). Accurate prediction of quality of transmission with dynamically configurable optical impairment model. In *Optical Fiber Communication Conference*, page Th1J.4. Optica Publishing Group.

Calvert, K., Doar, M., and Zegura, E. (1997). Modeling internet topology. *IEEE Communications Magazine*, 35(6):160–163.

de Miguel, I., Durán, R. J., Jiménez, T., Fernández, N., Aguado, J. C., Lorenzo, R. M., Caballero, A., Monroy, I. T., Ye, Y., Tymecki, A., Tomkos, I., Angelou, M., Klonidis, D., Francescon, A., Siracusa, D., and Salvadori, E. (2013). Cognitive dynamic optical networks. *J. Opt. Commun. Netw.*, 5(10):A107–A118.

Diaz-Montiel, A. A., Aladin, S., Tremblay, C., and Ruffini, M. (2019). Active wavelength load as a feature for qot estimation based on support vector machine. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6.

Doar, M. (1996). A better model for generating test networks. In *Proceedings of GLOBECOM'96. 1996 IEEE Global Telecommunications Conference*, volume MiniConfInternet, pages 86–93.

Dods, S. and Anderson, T. (2006). Optical performance monitoring technique using delay tap asynchronous waveform sampling. In *2006 Optical Fiber Communication Conference and the National Fiber Optic Engineers Conference*, pages 3 pp.–.

Díaz-Montiel, A. A., Lantz, B., Yu, J., Kilper, D., and Ruffini, M. (2021). Real-time qot estimation through sdn control plane monitoring evaluated in mininet-optical. *IEEE Photonics Technology Letters*, 33(18):1050–1053.

Geisler, D. J., Proietti, R., Yin, Y., Scott, R. P., Cai, X., Fontaine, N. K., Paraschis, L., Gerstel, O., and Yoo, S. J. B. (2011). The first testbed demonstration of a flexible bandwidth network with a real-time adaptive control plane. In *2011 37th European Conference and Exhibition on Optical Communication*, pages 1–3.

Ghobadi, M., Gaudette, J., Mahajan, R., Phanishayee, A., Klinkers, B., and Kilper, D. (2016). Evaluation of elastic modulation gains in microsoft's optical backbone in north america. In *Optical Fiber Communication Conference*, page M2J.2. Optica Publishing Group.

Hongsuda, Govindan, Tangmunarunkit, R., Jamin, S., Shenker, S., and Willinger, W. (2002). Network topology generators: Degree-based vs. structural. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '02, page 147–159, New York, NY, USA. Association for Computing Machinery.

Jin, X., Giddings, R., and Tang, J. (2011). Experimental demonstration of adaptive bit and/or power loading for maximising real-time end-to-end optical ofdm transmission performance. In *2011 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference*, pages 1–3.

Jong, K.-C. (2008). Q-factor monitoring of optical signal-to-noise ratio degradation in optical dpsk transmission. *Electronics Letters*, 44:761–763(2).

Khan, F. N., Lau, A. P. T., Li, Z., Lu, C., and Wai, P. K. A. (2010). Osnr monitoring for rz-dqpsk systems using half-symbol delay-tap sampling technique. *IEEE Photonics Technology Letters*, 22(11):823–825.

Khan, F. N., Lau, A. P. T., Lu, C., and Wai, P. K. A. (2011). Chromatic dispersion monitoring for multiple modulation formats and data rates using sideband optical filtering and asynchronous amplitude sampling technique. *Opt. Express*, 19(2):1007–1015.

Kilper, D., Bach, R., Blumenthal, D., Einstein, D., Landolsi, T., Ostar, L., Preiss, M., and Willner, A. (2004a). Optical performance monitoring. *Journal of Lightwave Technology*, 22(1):294–304.

Kilper, D. C., Bach, R., Blumenthal, D. J., Einstein, D., Landolsi, T., Ostar, L., Preiss, M., and Willner, A. E. (2004b). Optical performance monitoring. *J. Lightwave Technol.*, 22(1):294.

Kozicki, B., Takuya, O., and Hidehiko, T. (2008). Optical performance monitoring of phase-modulated signals using asynchronous amplitude histogram analysis. *Journal of Lightwave Technology*, 26(10):1353–1361.

Li, Z., Lu, C., Wang, Y., and Li, G. (2005). In-service signal quality monitoring and multi-impairment discrimination based on asynchronous amplitude histogram evaluation for nrz-dpsk systems. *IEEE Photonics Technology Letters*, 17(9):1998–2000.

Mata, J., Miguel Jiménez, I. d., Durán Barroso, R. J., Aguado Manzano, J. C., Merayo Álvarez, N., Ruiz Pérez, L., Fernández Reguero, P., Lorenzo Toledo, R. M., and Abril Domingo, E. J. (2017). A svm approach for lightpath qot estimation in optical transport networks.

Meng, F., Yan, S., Wang, R., Ou, Y., Bi, Y., Nejabati, R., and Simeonidou, D. (2017). Robust self-learning physical layer abstraction utilizing optical performance monitoring and markov chain monte carlo. 3rd European Conference and Exhibition on Optical Communication : ECOC 2017 ; Conference date: 17-09-2017.

Mininet-Optical Project (2022). Mininet-optical emulator. `https://mininet-optical.org/`. Last checked on Aug 15, 2022.

Mo, W., Gutterman, C. L., Li, Y., Zhu, S., Zussman, G., and Kilper, D. C. (2018). Deep-neural-network-based wavelength selection and switching in roadm systems. *J. Opt. Commun. Netw.*, 10(10):D1–D11.

Rosenfeldt, H., Clarke, I., Frisken, S., Dash, G., Huang, X., Li, H., Cui, W., Zhang, J., Chen, J., Kong, Z., and Poole, S. (2015). Miniaturized heterodyne channel monitor with tone detection. In *Optical Fiber Communication Conference*, page W4D.7. Optica Publishing Group.

Rottondi, C., Barletta, L., Giusti, A., and Tornatore, M. (2018). Machine-learning method for quality of transmission prediction of unestablished lightpaths. *J. Opt. Commun. Netw.*, 10(2):A286–A297.

Shake, I., Takara, H., Uchiyama, K., and Yamabayashi, Y. (2001). Quality monitoring of optical signals influenced by chromatic dispersion in a transmission fiber using averaged q-factor evaluation. *IEEE Photonics Technology Letters*, 13(4):385–387.

Tomlinson, W. J. (2008). Evolution of passive optical component technologies for fiber-optic communication systems. *Journal of Lightwave Technology*, 26(9):1046–1063.

Wang, Y., Hu, S., Yan, L., Yang, J.-Y., and Willner, A. E. (2007). Chromatic dispersion and polarization mode dispersion monitoring for multi-level intensity and phase modulation systems. *Opt. Express*, 15(21):14038–14043.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442.

Waxman, B. (1988). Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622.

Wu, X., Jargon, J. A., Wang, C.-M., Paraschis, L., and Willner, A. E. (2010). Experimental comparison of performance monitoring using neural networks trained with parameters derived from delay-tap plots and eye diagrams. In *2010 Conference on Optical Fiber Communication (OFC/NFOEC), collocated National Fiber Optic Engineers Conference*, pages 1–3.

Yu, Y., Zhang, B., and Yu, C. (2014). Optical signal to noise ratio monitoring using single channel sampling technique. *Opt. Express*, 22(6):6874–6880.

Zhao, J., Lau, A. P. T., Qureshi, K. K., Li, Z., Lu, C., and Tam, H. Y. (2009). Chromatic dispersion monitoring for dpsk systems using rf power spectrum. *Journal of Lightwave Technology*, 27(24):5704–5709.

Zhenhua, Khan, Dong, F. N., Sui, Q., Zhong, K., Lu, C., and Lau, A. P. T. (2016). Optical performance monitoring: A review of current and future technologies. *Journal of Lightwave Technology*, 34(2):525–543.