LYNX: A GNPy-based web application for multi-vendor optical network planning

LYNX: A GNPy-based web application for multi-vendor optical network planning

Mohammad Saad Raza[(1)], Andrea D'Amico[(2)], Fehmida Usmani[(1)], Sami Mansoor Alavi[(1)], Muhammad Ali Taimoor[(1)], Vittorio Curri[(2)], Arsalan Ahmad[(1)]

[(1)] School of Electrical Engineering and Computer Science (SEECS), National University of Sciences & Technology (NUST), Islamabad, Pakistan,
[(2)] Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129, Torino, Italy
arsalan.ahmad@seecs.edu.pk

Abstract: We demonstrate LYNX, a web-based application for network planning. Built on top of GNPy, it automates the design of user-defined or already built-in network topology, dynamic resource provisioning, network recovery and optimization.

1. Introduction

In the last two decades, continuous advances in optical transmission networks revolutionized the telecommunication industry to meet the rapidly growing internet traffic demands, particularly in backbone networks [1]. This poses several challenges for operators and vendors to come up with new solutions to maximize the capacity and returns on existing network infrastructure and to inter-operate existing and new components during migration processes. In addition, there has been growing interest in other network characteristics, such as dynamic resource provisioning and fault tolerance. All these factors lead to fundamental challenges for network operators. With the use of appropriate technologies and engineering practices, operators need to carry out careful planning, designing network topologies while bearing in mind the future upgrade demands. For this purpose, we developed LYNX, a multi-layer tool based on GNPy [2]. LYNX is a web-based open-source framework, publicly accessible on [3]. It is designed with the objective to provide an interactive Graphical User Interface (GUI) based solution to model networks from scratch or upgrade existing networks according to user requirements. Exploiting the network abstraction provided by GNPy [4], users can test dynamic traffic patterns and draw optimal solutions of network element configurations along with appropriate lightpath selection.

2. LYNX Overview

LYNX has been designed as a Software as a Service (SaaS) Product. The architecture of LYNX is shown in Fig.1. It is structured into frontend and backend code repositories written in JavaScript. The frontend renders views and handles user interaction through LYNX GUI, while all of the business logic is written at the backend running on servers and is hosted as a service. The system follows a mixture of Application Programming Interface (API) driven and layered architectural approaches. The frontend is primarily composed of three layers which are views, service controllers and customized TypeScript Software Development Kit (SDK) written for GNPy integration. Each layer has its own set of responsibilities. Views contain the web components required for gathering relevant information from the user and the WebGL-powered Map Graphics for the geographical representations of network topology. They handle user interaction and User Interface (UI) rendering. The major challenges during interface development are compatibility with GNPy while maintaining data integrity and data completeness. We developed a customized Typescript SDK to overcome these challenges. The "gnpy-typescript" SDK handles serialization and de-serialization of network equipment objects to make them compatible with the GNPy core engine while allowing their use in a web application. The Service controllers communicate with the TypeScript SDK through the APIs to exchange data and interact with views to update the GUI accordingly. They are also responsible for maintaining the state of the application and providing a common ground for publishing events and subscribing to those events. The backend is composed of two layers: main processes and supplementary processes depending on the scale of LYNX. The main process is responsible for handling data exchange to execute business logic and for performing computations. In this layer, the physical topology's auto-design and spectrum assignments are carried out with GNPy API. Additionally, the main process handles user authentication, project data about a particular user and application analytics with the help of supplementary processes. It also communicates back and forth with a persistent data store. The supplementary processes control the execution of curated python scripts, which utilize the python-based GNPy API. The main reason for offloading this work to supplementary processes is to tackle the scalability issue for LYNX. Since the computation times grow larger for increasingly complex network topologies, executing them in the main processes would impede the normal performance of LYNX. Therefore, we need an

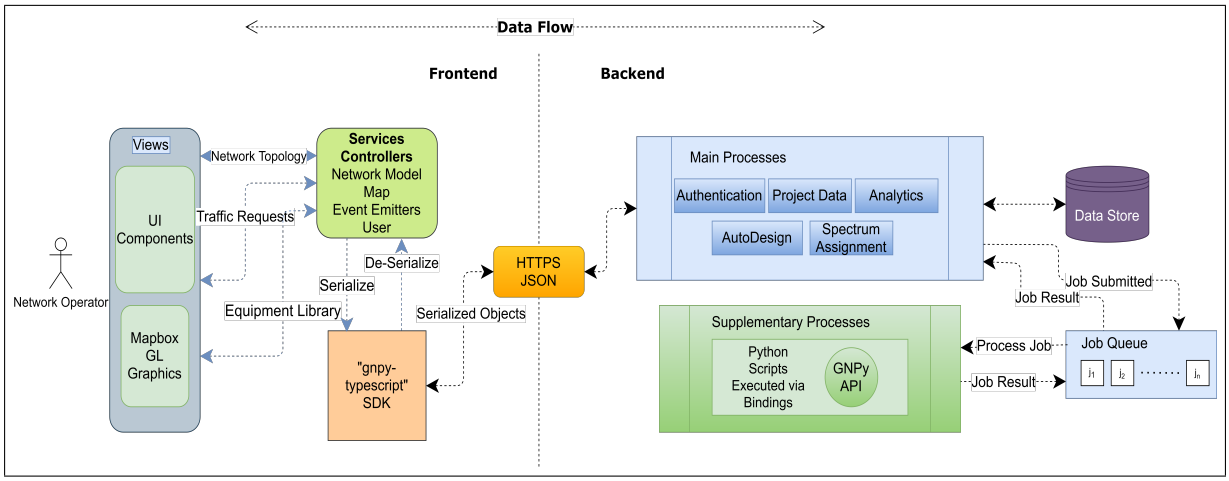


Fig. 1: Architecture of LYNX

asynchronous method for handling the computations. The LYNX offers the use of Job Queues as a solution. The main process, on encountering a physical layer computation, creates a thread and submits a job to the Job Queue where the thread waits for the results to be computed. The worker processes will pick up jobs from the queue and after processing put back the result. The number of supplementary processes depends on the scalability of LYNX, there are enough worker processes available to process submitted jobs and results to be conveyed back to the users with minimal overhead.

The physical topology, the traffic matrix and the network parameters are supplied as input to LYNX, which performs state-of-the-art routing and wavelength assignment with GNPy API to find an optimal solution for a given network state, indicating occupied lightpaths and the number of traffic requests served.

Remarkably, the LYNX framework enables dynamic changes in network configurations allowing an easy analysis of overall critical issues and possible upgrade. Moreover, LYNX provides the following major functionalities:

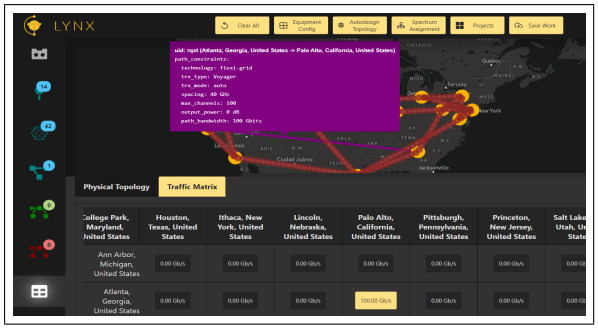
- A persistent state management including user authentication, session management, and project saving and loading functionality. User authentication is done with the token exchange between the frontend, the backend and the OAuth2 service provider, i.e. Google Firebase. Session state is tracked using a cookie and a database. As the session expires, the cookie gets deleted and the user can no longer access the protected resources of the core until he logins again. Cloud Firestore is used to store users' projects. Any backend computation done on the data is not stored in the database as it is redundant and can be acquired again given the project state. The stored record contains physical topology, equipment configurations and traffic requests.

- Fault tolerance and reporting features have been implemented to ease the user experience using LYNX. The former assists the user in finding fault-tolerant lightpaths for a given source to destination nodes, whereas, the reporting feature allows the user to visualize the network design information such as connected nodes, fiber constraints, parameters of traffic requests, assigned lightpaths, lightpath metrics and blocked lightpath requests.

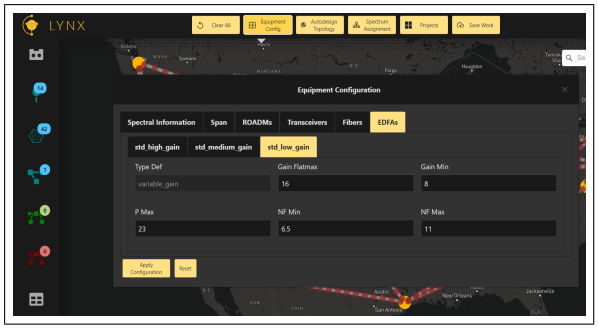
3. LYNX Use Case: Network Upgrade for Ligthpath Deployment

We demonstrate a use case based on the National Science Foundation Network topology. It consist of 14 nodes and 21 links and its default configurations can be retrieved within the example topologies available in LYNX. As per GNPy, the network configurations given in the topology description can be partial and the auto-design GNPy feature can be used in the LYNX framework to build a fully defined network abstraction. Given the nodes positions and the available links, the auto-design feature optimizes the amplifier placements and working points selecting the missing device configurations from a set of available equipment defaults, as shown in Fig. 2c.

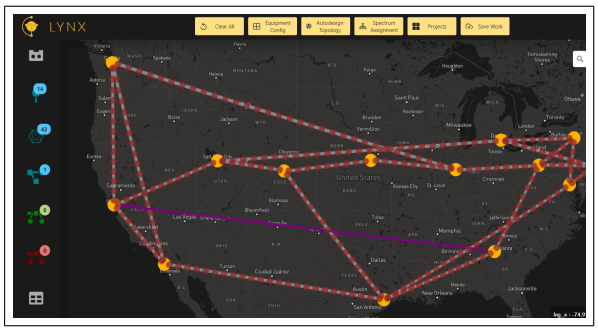
In general, the equipment configurations can be given by the user or easily changed according to users' requirements after the auto-design procedure. In the considered use case, we demonstrate that a given traffic request, a 100 Gbit/s transmission from Atlanta to Palo Alto, which is blocked when the default equipment configurations are taken into account, can be achieved changing the in-line EDFA noise figures (NF). The traffic request can be specified within the traffic matrix feature available in LYNX as shown in Fig 2a. At first, the configurations of the in-line EDFAs along the shortest path connecting Atlanta and Palo Alto are the default values given for the standard low gain EDFA, show in Fig 2b. In particular, the minimum and maximum NF values are 6.5 and 11 dB, respectively. Launching the spectrum assignment feature, the requested connection is blocked, shown in Fig. 2d, as the simulated generalized SNR (GSNR) is below the transceiver threshold. At this point, the user can explore a



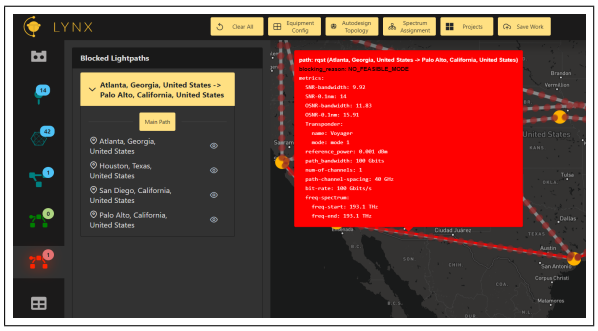
(a) Traffic Matrix



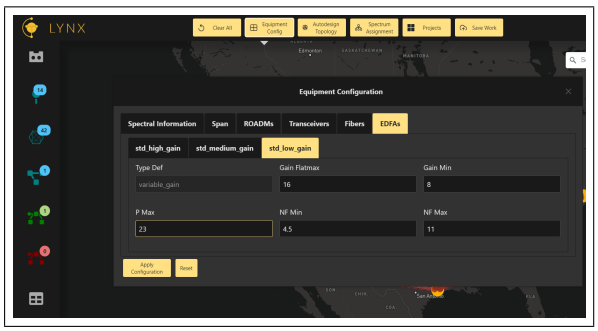
(b) High Noise Figure EDFA Configuration



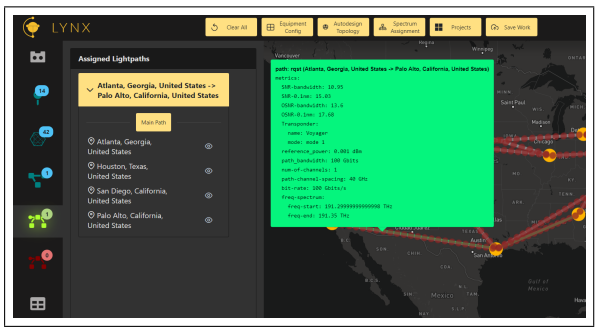
(c) Auto-Designed Topology of NSFNET



(d) Blocked Lighpath for Atlanta to Palo Alto



(e) Low Noise Figure EDFA Configuration



(f) Assigned Lighpath for Atlanta to Palo Alto

Fig. 2: LYNX Use Case

possible network upgrade that enables the given traffic request. For example, the minimum NF values can be easily modified for all the in-line EDFAs as shown in Fig. 2e. Changing the minimum NF from 6.5 to 4.5 dB lowers the total noise due to the in-line amplification and provides an higher GSNR for the selected lightpath. After this upgrade, the traffic request can be achieved and the lightpath is deployed. The resulting transmissions details are reported in Fig. 2d

4. Conclusion

We demonstrate a simple use case where LYNX can be easily used to simulate a network upgrade and its effect on the available traffic. This example represents a limited subset of the many possible effective applications of LYNX, which undoubtedly serves as an accelerator and catalyst for efficient network analysis, both from an academic and industrial standpoint.

References

1. Cisco, “Cisco annual internet report (2018–2023) white paper,” (2020).
2. A. Ferrari *et al.*, “Gnpy: an open source application for physical layer aware open optical networks,” JOCN **12** (2020).
3. https://app-lynx-corenet.herokuapp.com/ (2021).
4. A. Ferrari *et al.*, “Gnpy: an open source planning tool for open optical networks,” (2020), pp. 1–6.