UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Aprendizaje por refuerzo en redes ópticas pasivas (PON)

Presentado por David Pérez Moreno
en Universidad de Burgos — 8 de julio de 2024
Tutor: Rubén Ruiz González
Cotutora: Noemí Merayo Álvarez

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Rubén Ruiz González, profesor del Departamento de Digitalización, área de Ingeniería de Sistemas y Automática.

Dña. Noemí Merayo Álvarez, en calidad de tutora externa y profesora de la ETSIT de la Universidad de Valladolid.

Exponen:

Que el alumno D. David Pérez Moreno, con DNI 71363734R, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado "Aprendizaje por refuerzo en redes ópticas pasivas (PON)".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección de quienes suscriben, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 8 de julio de 2024

Vº. Bº. del Tutor:	Vº. Bº. de la co-tutora:
D. Rubén Ruiz González	Dña. Noemí Merayo Álvarez

Resumen

El aprendizaje por refuerzo (RL) es una técnica de aprendizaje automático donde un agente aprende a través de recompensas y penalizaciones para tomar decisiones óptimas en un entorno dinámico. Por su parte, las redes ópticas pasivas (PON) suponen una cuota de mercado de redes de acceso sin competencia en la actualidad ni en las previsiones a corto plazo. En las redes PON, donde la asignación de anchos de banda de manera dinámica es crítica para una operación efectiva de la red, el aprendizaje por refuerzo puede mejorar la gestión de recursos y la calidad del servicio ofrecido a los clientes.

En este contexto, este TFG tiene como objetivo desarrollar un entorno simplificado de aprendizaje por refuerzo para la simulación de redes PON. Este entorno va a ser programado en lenguaje Python haciendo uso de la biblioteca Gymnasium. Así mismo, un segundo objetivo principal de este TFG es la evaluación del entorno y su entrenamiento en diversos escenarios de tráfico dinámico y cambiante.

Como resultados principales de este TFG se ha desarrollado un entorno de simulación funcional así como se ha evaluado el mismo con resultados satisfactorios, donde la red es capaz de hacer una asignación de ancho de banda a cada usuario que cumpla las velocidades contratadas al tiempo que minimiza la congestión de la red.

Como conclusión principal, la aplicación de técnicas de aprendizaje por refuerzo es útil a la hora de optimizar el tráfico, minimizar la congestión y ajustar la asignación de ancho de banda de forma dinámica. Esto contribuye a reducir la latencia, mejorar la eficiencia energética y mantener altos niveles de calidad del servicio.

Aunque este trabajo se centra en el uso de aprendizaje por refuerzo para algoritmos de planificación dinámica para mejorar la asignación de ranuras de tiempo, disminuyendo la latencia, numerosas líneas futuras se abren como respuesta al trabajo desarrollado en este TFG. La mejora incremental del entorno para realizar simulaciones más próximas a la realidad destaca por encima del resto.

Descriptores

Aprendizaje por refuerzo, redes ópticas pasivas (PON), optimización del tráfico, gestión de recursos, calidad del servicio, planificación dinámica, latencia, congestión, Gymnasium, Python, Stable-Baselines3.

Índice general

Índice de figuras

1. Introducción

El presente proyecto se centra en el **uso del aprendizaje por refuerzo** para **optimizar redes ópticas pasivas** (PON, Passive Optical Networks). **Dada la creciente demanda de ancho de banda y servicios de alta velocidad**, es fundamental contar con **soluciones que permitan gestionar el tráfico y los recursos de manera eficiente**, manteniendo al mismo tiempo **altos niveles de calidad del servicio**. [5]

Este trabajo se divide en varias secciones en las que cubrimos desde **conceptos fundamentales** hasta **aplicaciones concretas del aprendizaje por refuerzo en redes PON**. Se aborda cómo el **RL puede ser utilizado para reducir la latencia**, **controlar la potencia óptica**, **optimizar la asignación de ranuras de tiempo** y **gestionar errores y recuperaciones en tiempo real**. [12]

Además, se analiza la **estructura de la red PON** y los **desafíos asociados**, explorando cómo el RL puede ser una **herramienta clave para mejorar la eficiencia y la calidad del servicio**. También se incluyen **ejemplos prácticos y estudios de casos** que demuestran la aplicación del aprendizaje por refuerzo en este contexto.

Finalmente, se discuten las **conclusiones** y se ofrece una **visión sobre el futuro de las redes ópticas pasivas con el uso de técnicas de aprendizaje automático** como el RL. También se proporciona una **guía para futuros trabajos y mejoras potenciales** en la gestión de redes PON.

2. Objetivos del proyecto

El objetivo de este proyecto es **desarrollar y validar un marco de trabajo basado en el aprendizaje por refuerzo** (RL, Reinforcement Learning) **que optimice la gestión y operación de redes ópticas pasivas** (PON) **con el fin de mejorar la eficiencia del ancho de banda y la calidad del servicio** para todos los dispositivos de la red. Este objetivo encapsula varios aspectos clave: **el desarrollo de una solución técnica** (el marco de trabajo basado en RL), **la aplicación específica** (en redes PON), y **los resultados deseados** (mejora en eficiencia, reducción de latencia, y mejora en la distribución de recursos, en concreto del ancho de banda disponible lo que conlleva una mejora de la eficiencia y posible reducción de retardo especialmente en usuarios con altos requisitos de calidad de servicio). Además, plantea una **dirección clara para la investigación y el desarrollo**, y establece **criterios claros para la evaluación del éxito del proyecto**.

3. Conceptos teóricos

3.1. Conceptos Teóricos

Introducción al Aprendizaje por Refuerzo (RL)

El Aprendizaje por Refuerzo (RL, por sus siglas en inglés) es una rama fascinante del aprendizaje automático que se centra en enseñar a los agentes cómo actuar en un entorno para maximizar alguna noción de recompensa acumulativa. A diferencia de otros tipos de aprendizaje automático, en el RL no se le dice al agente qué acciones tomar, sino que debe **descubrir por sí mismo cuáles acciones le generan la mayor recompensa**, a través de un proceso de prueba y error. Este enfoque tiene sus raíces en la teoría del aprendizaje psicológico y ha ganado un gran impulso en las últimas décadas debido a su éxito en diversas aplicaciones prácticas, desde juegos hasta la robótica y más allá. [14]

El concepto de RL se puede trazar hasta el problema de los procesos de decisión de Markov, y ha sido formalizado y estudiado en la inteligencia artificial desde la década de 1980. Sin embargo, no fue hasta la introducción de métodos que combinan el RL con técnicas de aprendizaje profundo —**como el Deep Learning**— que esta área realmente comenzó a ver su potencial pleno, resolviendo problemas que antes parecían inaccesibles. [8]

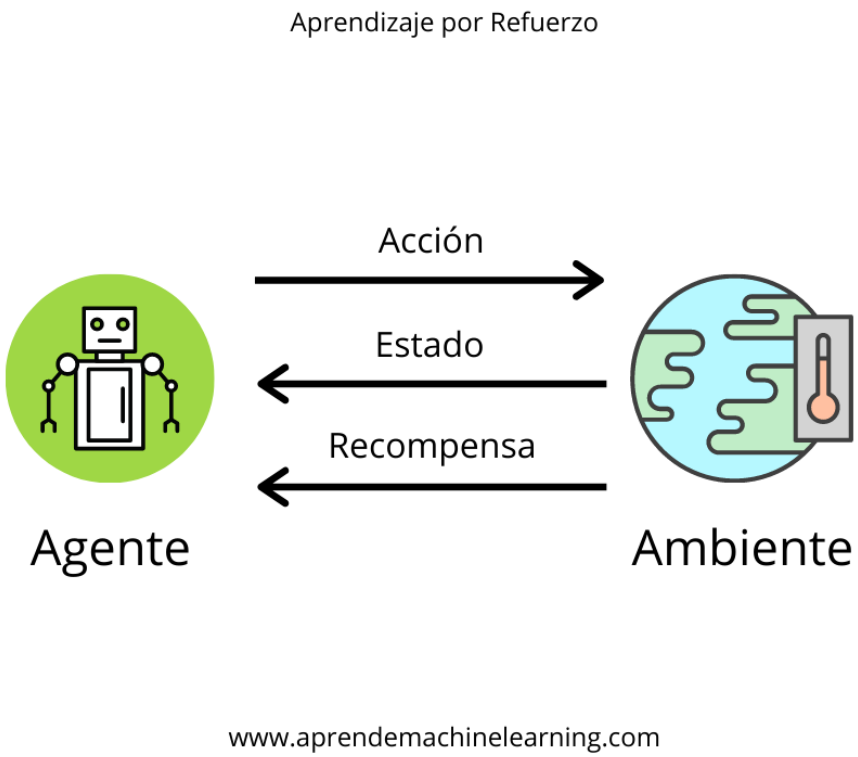

Figura 3.1: Representación de lo que es el Aprendizaje por Refuerzo. [1]

Conceptos Clave del Aprendizaje por Refuerzo

El aprendizaje por refuerzo se centra en la **interacción entre un agente y su entorno con el objetivo de maximizar una señal de recompensa acumulativa**. Este proceso se basa en varios conceptos fundamentales que estructuran cómo el agente aprende y toma decisiones.[7] A continuación, se detallan estos conceptos:

Agente

El agente es el núcleo del aprendizaje por refuerzo. Es una entidad autónoma con la capacidad de percibir su entorno a través de sensores y actuar sobre ese entorno a través de actuadores. En el contexto del aprendizaje por máquina, un agente podría ser un software que juega a un videojuego, un robot recogiendo objetos, o incluso un sistema de

recomendación que decide qué productos ofrecer a los usuarios. El agente necesita evaluar su rendimiento basándose en la recompensa acumulada y ajustar sus acciones futuras para mejorar continuamente.

Entorno

El entorno puede ser cualquier sistema con el que el agente interactúa y que reacciona a las acciones del agente. Este entorno podría ser completamente virtual, como en los juegos o simulaciones, o físico, como en el caso de robots en el mundo real. El entorno proporciona estados al agente, que son esencialmente instantáneas de la situación actual del entorno, incluyendo cualquier cambio que resulte de las acciones anteriores del agente.

Acción

Las acciones son las decisiones tomadas por el agente, seleccionadas de un conjunto de posibles acciones llamado espacio de acción. En juegos como el ajedrez, las acciones serían los diferentes movimientos legales. En la conducción autónoma, las acciones podrían incluir acelerar, frenar o girar. La elección de la acción adecuada en el momento adecuado es crucial, ya que determina la eficacia con la que el agente puede alcanzar su objetivo.

Estado

El estado del entorno es una descripción formal de la situación actual en la que se encuentra el agente. En muchos problemas de RL, el agente no tiene acceso a toda la información sobre el entorno, lo cual se conoce como un entorno parcialmente observable. Los estados deben contener suficiente información para que el agente tome decisiones informadas; sin embargo, deben ser lo suficientemente compactos para ser gestionables desde el punto de vista computacional.

Recompensa

La recompensa es una señal clave en el aprendizaje por refuerzo. Se da después de cada acción y indica qué tan bien está haciendo el agente. El objetivo del agente es maximizar la suma total de recompensas, lo que se conoce como retorno. La función de recompensa debe ser diseñada cuidadosamente para alinear los incentivos del agente con los objetivos a largo plazo deseados.

Estos elementos interactúan en un **ciclo continuo de observación, decisión y actuación**, donde el agente observa el estado del entorno, toma una decisión sobre qué acción realizar y recibe una recompensa basada en el nuevo estado resultante. Este ciclo se repite con el objetivo de acumular la mayor cantidad de recompensas a lo largo del tiempo, lo que eventualmente lleva al agente a aprender una política óptima de acciones.

Elementos Principales de un Modelo de Aprendizaje por Refuerzo

1. **Política (Policy): La política es el núcleo de la estrategia de un agente para tomar decisiones.** Define cómo el agente debe actuar en diferentes situaciones. Puede ser simple como una tabla de búsqueda que mapea estados a acciones, o compleja como una red neuronal que toma decisiones basadas en la percepción del estado actual del entorno. La política puede ser determinista, donde un estado dado siempre resultará en la misma acción, o estocástica, donde la acción es seleccionada según una distribución de probabilidad.

2. **Función de Recompensa (Reward Function): La función de recompensa es crucial porque motiva al agente proporcionando retroalimentación sobre la efectividad de sus acciones.** Es un reflejo directo de los objetivos que el sistema de aprendizaje debe cumplir, y por lo tanto, debe ser diseñada cuidadosamente para asegurar que guíe al agente hacia el comportamiento deseado. Un buen diseño de la función de recompensa ayudará a hacer el aprendizaje más eficiente y efectivo.

3. **Función de Valor (Value Function):** Mientras que la función de recompensa proporciona una medida instantánea de la recompensa obtenida después de una acción específica, **la función de valor estima lo bueno que es un estado (o una acción en un estado dado) a largo plazo.** La función de valor ayuda al agente a evaluar y comparar los estados o acciones basándose no solo en las recompensas inmediatas, sino también en las recompensas futuras esperadas. Esto permite al agente planificar estratégicamente y tomar decisiones que maximizan las recompensas a lo largo del tiempo.

4. **Modelo del Entorno (Model of the Environment):** En algunos enfoques de RL, como el aprendizaje basado en modelos, el agente tiene acceso a un modelo del entorno que puede predecir el próximo

estado y la recompensa resultante de sus acciones. **Este modelo permite al agente planificar al simular posibles futuros antes de tomar una decisión**, lo cual puede ser especialmente útil en entornos complejos y dinámicos. Sin embargo, construir un modelo preciso puede ser desafiante y, en muchos casos, los agentes operan sin un modelo explícito, en un marco llamado aprendizaje sin modelo.

Tipos de Aprendizaje por Refuerzo

El aprendizaje por refuerzo puede clasificarse en varias categorías basadas en cómo el agente interactúa con el entorno y cómo aprende a partir de esa interacción. Aquí se describen los principales tipos:

Aprendizaje basado en Modelos vs. Aprendizaje sin Modelos

En el aprendizaje basado en modelos, **el agente tiene o construye un modelo del entorno que describe la dinámica del entorno.** Este modelo se usa para prever los resultados de las acciones, permitiendo una planificación más profunda y la posibilidad de considerar futuros hipotéticos antes de actuar. El aprendizaje sin modelos no utiliza un modelo del entorno. En lugar de ello, el agente aprende directamente de la experiencia acumulada a través de la interacción con el entorno, generalmente mediante el método de prueba y error. Esta aproximación es generalmente más simple pero puede requerir más experiencias para aprender a actuar eficazmente.

Métodos Monte Carlo

Los métodos Monte Carlo en RL son utilizados para **estimar las funciones de valor basándose en muestras completas de episodios.** No requieren un modelo del entorno y se actualizan al final de cada episodio, lo que los hace particularmente adecuados para tareas con horizontes temporales definidos y claros.

Aprendizaje por Diferencias Temporales (Temporal-Difference Learning)

El aprendizaje por diferencias temporales, o TD, **combina ideas de los métodos Monte Carlo y el aprendizaje dinámico de programación.** TD puede aprender directamente de la experiencia cruda sin necesidad de un modelo del entorno y actualiza sus estimaciones de valor basado parcialmente en otras estimaciones aprendidas, sin esperar a que finalice un episodio.

Deep Reinforcement Learning (DRL)

DRL extiende el RL tradicional utilizando técnicas de aprendizaje profundo. Esto permite al agente aprender políticas y funciones de valor a través de redes neuronales, lo cual es efectivo en entornos con un alto grado de complejidad y gran cantidad de estados y acciones. Ejemplos notables incluyen DQN, PPO, y métodos de gradiente de política.[10]

Cada uno de estos tipos de aprendizaje por refuerzo ofrece ventajas y desafíos únicos, y la elección de uno sobre otro dependerá en gran medida de las especificidades del problema y del entorno en el que el agente debe operar.

Algoritmos en el Aprendizaje por Refuerzo

Esta parte del manual se centrará en describir algunos de los algoritmos clave utilizados en el aprendizaje por refuerzo, proporcionando una visión general de su funcionamiento y aplicaciones.

- **Q-Learning:** Q-Learning es un método de aprendizaje sin modelo que busca aprender una política óptima de forma indirecta, a través de la estimación de la función de valor Q, que da el valor de realizar una acción en un estado particular. Uno de los aspectos más significativos de Q-Learning es que puede comparar la utilidad esperada de las acciones disponibles sin necesitar un modelo del entorno. Este algoritmo es particularmente útil en entornos estocásticos y para tareas de decisión secuenciales.

- **SARSA (State-Action-Reward-State-Action):** SARSA es también un método de aprendizaje por diferencias temporales y es similar a Q-Learning. La principal diferencia es que SARSA es un método .ºn-policy", lo que significa que el aprendizaje ocurre sobre la política actual que el agente está siguiendo, mientras que Q-Learning es .ºff-policyז aprende basándose en lo que sería óptimo hacer, no necesariamente lo que el agente elige hacer. Esto hace que SARSA sea menos propenso a realizar acciones riesgosas en comparación con Q-Learning.

- **Deep Q-Network (DQN):** El DQN es una extensión de Q-Learning que utiliza redes neuronales profundas para aproximar la función de valor Q. Este método fue pionero en combinar técnicas de aprendizaje profundo con RL, permitiendo que los agentes manejen percepciones

de alta dimensionalidad, como imágenes directamente de videojuegos. DQN también incorpora técnicas como la repetición de experiencias y la iteración de objetivos para estabilizar el aprendizaje.

- **Policy Gradient Methods:** Los métodos de gradiente de política optimizan la política de acciones directamente como una función de los parámetros de una red neuronal. Estos métodos son útiles en entornos donde las acciones son numerosas o complejas y donde las funciones de valor no pueden ser fácilmente estimadas. Un ejemplo prominente es REINFORCE, que actualiza los parámetros de la política en una dirección que mejora la recompensa esperada.

- **Proximal Policy Optimization (PPO):** PPO es un método de gradiente de política que ha ganado popularidad por su balance entre simplicidad y eficacia. **Utiliza técnicas para mantener los cambios en la política dentro de un margen controlado, lo que ayuda a evitar las caídas de rendimiento durante el entrenamiento.** PPO es notablemente efectivo en una variedad de entornos de simulación y ha sido un algoritmo de elección para muchos problemas prácticos de RL.[13]

Estos algoritmos representan una parte fundamental de las estrategias modernas de aprendizaje por refuerzo y son cruciales para desarrollar sistemas inteligentes que pueden aprender y adaptarse de manera efectiva.

Desafíos y Consideraciones en el Aprendizaje por Refuerzo

- **Exploración vs. Explotación:** Uno de los dilemas centrales en el aprendizaje por refuerzo es el equilibrio entre exploración (probar nuevas acciones para descubrir sus recompensas) y explotación (usar las acciones que se sabe generan las mayores recompensas). Encontrar el balance adecuado es crucial, ya que una exploración insuficiente puede llevar a soluciones subóptimas, mientras que una explotación excesiva puede prevenir el descubrimiento de mejores estrategias.

- **Recompensa Diferida:** En muchos problemas de RL, la recompensa puede estar significativamente retrasada desde el momento de la acción. Esto hace que sea difícil para el agente determinar cuál de sus acciones anteriores fue realmente responsable del resultado obtenido. Resolver

este problema a menudo requiere sofisticadas técnicas de atribución de crédito y un diseño cuidadoso de la función de recompensa.

- **Escalabilidad y Eficiencia:** A medida que el tamaño del entorno y el número de posibles estados y acciones aumenta, los algoritmos de RL tradicionales pueden volverse computacionalmente inviables. La escalabilidad sigue siendo una gran barrera para la aplicación práctica del RL en entornos grandes y complejos. Mejorar la eficiencia tanto en términos de computación como de uso de datos es un área activa de investigación.

- **Estabilidad y Convergencia:** Los métodos de aprendizaje por refuerzo pueden ser propensos a problemas de estabilidad y convergencia, especialmente cuando se utilizan aproximadores de función, como las redes neuronales. Las oscilaciones en las estimaciones de valor y las políticas pueden llevar a fluctuaciones significativas en el aprendizaje, haciendo difícil alcanzar o mantener un rendimiento óptimo.

- **Generalización:** La capacidad de un modelo de RL para generalizar desde su experiencia de entrenamiento a situaciones nuevas y no vistas es fundamental para su éxito en aplicaciones reales. Sin embargo, la generalización sigue siendo un desafío, particularmente en entornos que difieren significativamente de aquellos vistos durante el entrenamiento.

Aplicaciones del Aprendizaje por Refuerzo

El aprendizaje por refuerzo ha encontrado aplicaciones en una variedad de dominios, demostrando su capacidad para manejar tareas complejas y dinámicas. Algunos de los ejemplos más destacados incluyen:

- **Robótica:** En la robótica, el aprendizaje por refuerzo se utiliza para enseñar a los robots habilidades como caminar, manipular objetos, o realizar tareas complejas de manera autónoma. Estos sistemas aprenden a optimizar sus acciones basadas en la retroalimentación directa del entorno físico en el que operan.

- **Juegos:** El aprendizaje por refuerzo ha alcanzado logros notables en el ámbito de los juegos, desde juegos clásicos de mesa como el Go, hasta videojuegos complejos. Por ejemplo, sistemas basados en RL como AlphaGo y OpenAI Five han demostrado habilidades superiores a las humanas en juegos respectivamente como Go y Dota 2.

- **Automoción Autónoma:** Los vehículos autónomos utilizan el aprendizaje por refuerzo para tomar decisiones en tiempo real sobre la conducción, como el cambio de carriles, la aceleración y la frenada, aprendiendo de las consecuencias de sus acciones en entornos simulados antes de aplicar el conocimiento en el mundo real.

- **Sistemas de Recomendación:** Los sistemas de recomendación modernos utilizan RL para personalizar las sugerencias de productos, servicios o contenido a los usuarios. Al maximizar la interacción del usuario, estos sistemas pueden mejorar continuamente la relevancia de sus recomendaciones.

- **Salud:** En el sector de la salud, el aprendizaje por refuerzo puede ser usado para optimizar tratamientos y procedimientos médicos. Algoritmos de RL están siendo explorados para personalizar las dosis de medicamentos en tratamientos prolongados o para automatizar ciertos procedimientos diagnósticos. Estas aplicaciones ilustran la amplia gama de problemas que el aprendizaje por refuerzo puede ayudar a resolver, gracias a su capacidad para aprender políticas óptimas en entornos inciertos y dinámicos.

- **Redes Ópticas Pasivas (Redes PON):** Mediante el uso de RL, los sistemas pueden **aprender a distribuir recursos de red de manera más eficiente entre los usuarios, adaptándose dinámicamente a los cambios en la demanda de tráfico y las condiciones de red.** Esto no solo mejora la experiencia del usuario final sino que también incrementa la eficiencia operativa de la red.

Introducción a Redes Ópticas Pasivas (Redes PON)

Las Redes Ópticas Pasivas, comúnmente conocidas como Redes PON, son una **tecnología fundamental en la infraestructura de telecomunicaciones.** Este tipo de red se caracteriza por su **capacidad para proporcionar conectividad de banda ancha a múltiples usuarios mediante un único hilo de fibra óptica que se divide para servir a varios puntos de terminación sin activación electrónica entre ellos.** Las redes PON son esenciales debido a su eficiencia en costos y energía, facilitando una cobertura amplia y fiable, especialmente en áreas urbanas densamente pobladas.[6]

Los componentes principales de una Red PON incluyen:

1. **Terminal de Línea Óptica (OLT):** Ubicado en el proveedor de servicios, el OLT es el punto de inicio de la red PON. Coordina la multiplexación de señales, la asignación de ancho de banda y la comunicación entre los diferentes usuarios.

2. **Unidades de Red Óptica (ONU):** Estos dispositivos están instalados en los lugares de los usuarios finales. Convierten las señales ópticas en señales eléctricas adecuadas para su uso en edificios residenciales o empresariales.

3. **División de Fibra Óptica:** La red utiliza divisores pasivos que no requieren alimentación eléctrica para dividir una única señal óptica en múltiples señales que se distribuyen a los usuarios finales.

4. Estos elementos trabajan en conjunto para proporcionar un servicio eficiente y de alta capacidad, aprovechando las ventajas de la fibra óptica, como la baja atenuación y la alta ancho de banda.

Las redes PON presentan una **topología en árbol** donde el OLT, ubicado en la oficina central del proveedor de servicios, se comunica con múltiples ONUs a través de divisores ópticos. Esta topología permite que una única señal del OLT se divida y se distribuya a varias ONUs, optimizando así el uso de la infraestructura de fibra óptica. [17]

En el **canal descendente** (desde el OLT hacia las ONUs), la red opera en una configuración punto-multipunto. El OLT transmite la información hacia todas las ONUs mediante una técnica de difusión (broadcast), utilizando **Multiplexación por División en Tiempo** (TDM). Cada ONU filtra y recibe solo el contenido destinado a ella.

Por otro lado, en el **canal ascendente** (desde las ONUs hacia el OLT), la red opera en una configuración punto a punto. Las ONUs envían datos al OLT utilizando **Acceso Múltiple por División en Tiempo** (TDMA) para evitar colisiones y asegurar una transmisión ordenada.

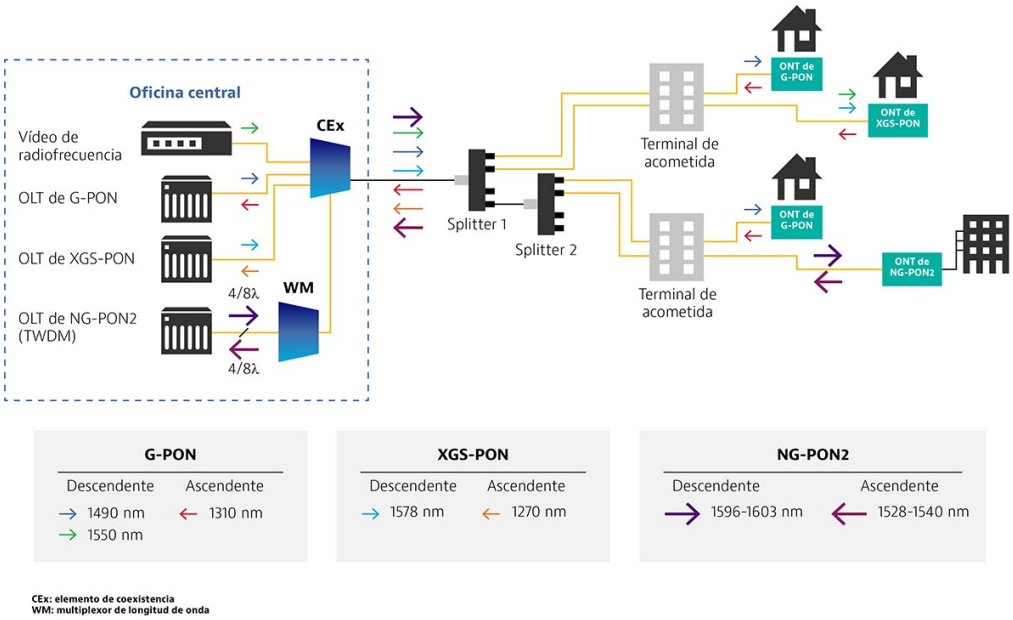

Figura 3.2: Redes PON[4]

Para gestionar eficientemente el ancho de banda disponible, se utilizan **algoritmos de asignación dinámica de ancho de banda** (DBA). Estos algoritmos, gestionados por el OLT, asignan ancho de banda a las ONUs en función de la demanda y otros factores como el tráfico y los requisitos de calidad de servicio (QoS). Los algoritmos DBA pueden ser de dos tipos principales: [17]

- **Algoritmos centralizados:** Asignan el ancho de banda a cada ONU al final de cada ciclo de tiempo, tras conocer las demandas de todas las ONUs.
- **Algoritmos de polling:** Asignan el ancho de banda a cada ONU de forma independiente, basándose en la demanda recibida de cada ONU sin esperar a que finalice el ciclo completo. En nuestro programa nos centraremos en estos algoritmos.

En resumen, las Redes Ópticas Pasivas (PON) son una solución avanzada y eficiente para proporcionar servicios de banda ancha, aprovechando la fibra óptica y técnicas de multiplexación y asignación dinámica de recursos para optimizar la transmisión de datos entre el proveedor de servicios y los usuarios finales.

Introducción a la Distribución de Pareto

La Distribución de Pareto, también conocida como el Principio de Pareto o la regla del 80/20", es una **teoría utilizada en economía y estadística que describe la distribución desigual de recursos o resultados.** Fue nombrada así por el economista italiano Vilfredo Pareto, quien observó que el 80 % de la propiedad en Italia estaba en manos del 20 % de la población. La distribución de Pareto se ha aplicado en diversos campos para describir fenómenos donde una pequeña proporción de causas o esfuerzos suelen producir la mayoría de los beneficios o resultados. [16]

Características de la Distribución de Pareto:

- **Asimetría:** La distribución de Pareto es conocida por su forma asimétrica, donde hay una alta frecuencia de valores bajos y una baja frecuencia de valores altos, lo que indica que los recursos o beneficios no están distribuidos equitativamente.
- **Aplicabilidad:** Se utiliza en análisis de calidad, gestión de negocios y ciencias sociales para identificar las çausas vitales"que necesitan más atención y optimización.

Aplicación de la Distribución de Pareto en Redes PON

En el contexto de redes ópticas pasivas, **la Distribución de Pareto puede aplicarse para la simulación del tráfico de usuarios.** Esta distribución permite modelar de manera precisa cómo una pequeña cantidad de usuarios genera la mayoría del tráfico en la red. Al utilizar esta simulación, los operadores de red pueden prever y analizar los patrones de tráfico, lo que facilita la optimización de la infraestructura y la asignación de recursos, mejorando así la eficiencia general de la red y la calidad del servicio.

4. Técnicas y herramientas

Esta parte de la memoria tiene como objetivo **presentar las técnicas metodológicas y las herramientas de desarrollo que se han utilizado para llevar a cabo el proyecto.** Si se han estudiado diferentes alternativas de metodologías, herramientas, bibliotecas, se puede hacer un resumen de los aspectos más destacados de cada alternativa, incluyendo comparativas entre las distintas opciones y una justificación de las elecciones realizadas. No se pretende que este apartado se convierta en un capítulo de un libro dedicado a cada una de las alternativas, sino comentar los aspectos más destacados de cada opción, con un repaso a los fundamentos esenciales y referencias bibliográficas, para que el lector pueda ampliar su conocimiento sobre el tema.

4.1. Microsoft Visual Studio Code

Visual Studio Code es una herramienta para la creación, desarrollo y mantenimiento de código en diversos lenguajes de programación. Gracias a esta puedo estar trabajando con diversos lenguajes sin problemas de compatibilidad. Ha sido empleado para la creación del código para los archivos Python y los Python Notebooks.

4.2. Scrum

Scrum es un marco de trabajo para el desarrollo de software englobado dentro de las metodologías ágiles. Gracias a esta metodología he podido dividir mi trabajo de forma iterativa gracias al planteamiento

de sprints y revisiones y según ello ir poniendo determinadas fechas para la resolución de los objetivos.[3]

4.3. GitHub

GitHub es una herramienta usada para el control de versiones de forma colaborativa en la que nosotros como usuarios podemos subir nuestros códigos y que otros puedan participar en nuestro repositorio creando modificaciones externas. Gracias a esta herramienta he podido subir mi código para pedir retroalimentaciones de mis tutores así como para tener un control de versiones y planificación en este repositorio. El control de versiones se puede ver a través de los commits y para ver la planificación del proyecto en la interfaz de proyecto donde se puede ver la cronología de trabajo que se ha realizado.

4.4. Outlook

Outlook es un programa para el sistema gestor de correos electrónicos. Esta herramienta ha sido semanalmente usada para la comunicación de avances a mis tutores, además de dudas casuales o compartición de archivos auxiliares para mi aprendizaje.

4.5. Microsoft Teams

Microsoft Teams es una aplicación que nos permite comunicar a varios usuarios. Esta comunicación se realiza mediante llamadas lo que facilita que las personas se estén comunicando como en una llamada de teléfono, además de poder compartir el escritorio para poder mostrar el trabajo que se esté realizando y poder usar videocámaras las cuales dan un enfoque mas cercano de las personas que estén en la reunión.

4.6. Microsoft OneDrive

La herramienta de Microsoft OneDrive sirve para almacenar los archivos en la nube y poder compartirlos con otros usuarios. He usado esta herramienta para compartir mi trabajo con los tutores y que puedan ver el código de una manera más fácil, además de ellos compartirme documentos auxiliares por este medio.

4.7. SonarCloud

SonarCloud es una herramienta que nos permite detectar vulnerabilidades en nuestro código, así como poder detectar y solucionar errores en nuestro código y poder ver la cobertura de código que ofrecen nuestros test.

5. Aspectos relevantes del desarrollo del proyecto

En este apartado detallaré los aspectos más importantes del desarrollo del proyecto, así como de todas las decisiones tomadas, las implicaciones que han supuesto y los problemas ocasionados y sus resoluciones.

En este apartado se relata el desarrollo del proyecto, así desde los comienzos probando y aprendiendo acerca de los diferentes conceptos que se me incluían, como también los fallos que he ido cometiendo y arreglando, y cómo su evolución ha desarrollado mis capacidades.

5.1. Elección del proyecto

Elegí este trabajo ya que me daba la **oportunidad de trabajar en entornos de Machine Learning y fortalecer así mis conocimientos de redes,** así como trabajar con fundamentos y profundizar en unos temas en los que en la propia universidad no había profundizado y este proyecto me ha brindado la oportunidad de hacerlo. También cuenta el que es un proyecto ambicioso lo que dificulta la implementación futura en un simulador.

5.2. Iniciación al DRL (Deep Reinforcement Learning)

Para la iniciación a los entornos DRL (Deep Reinforcement Learning) busqué unos códigos de aprendizaje por refuerzo con los que estuve experimentando el funcionamiento de los entornos de machine learning, en los que

pude aumentar mis conocimientos y focalizarme en el estudio de las librerías con las que iba a trabajar en un futuro, las cuales son Gymnasium[9] y StableBaselines 3, en el lenguaje de programación de Python y Jupyter Notebook.

5.3. DRL en el entorno de Molinos Eólicos

Una vez ya entendido el funcionamiento de cada uno de los algoritmos de aprendizaje y de las librerías comentadas, hablé con mis tutores para la realización de un entorno experimental en el que fortalecí mis conocimientos de cómo construir el código desde cero.

Este nuevo código consistió en un **entorno de DRL asociado a Molinos Eólicos** y cómo observando las variables de la velocidad del aire, la dirección del viento y la orientación de las turbinas podemos maximizar la potencia que dan cada uno de los molinos.

Como podemos ver en la imagen, sacamos el resultado de la potencia y cómo gracias a este entorno de DRL hemos podido maximizar en todo lo posible y con las variaciones de viento u orientaciones la potencia obtenida.

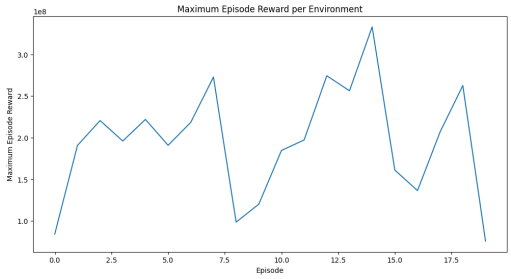

Figura 5.1: Resultados Potencia Molinos Eólicos

Gracias a este trabajo inicial pude terminar de comprender las nuevas librerías y el cómo poder utilizarlas así como la diferencia de códigos usados, teniendo por una parte un entorno en código Python y un script en Jupiter Notebook en el que pongo los casos de uso para la ejecución del programa y desde donde hago la ejecución del propio.

5.4. DRL Versión 1.0 Redes Ópticas Pasivas

Una vez ya teniendo más experiencia en los entornos DRL, empecé a trabajar en el entorno que me habían puesto mis tutores para la realización del trabajo de fin de grado. Para ello tuve que investigar y estudiar acerca de las redes PON y cómo implementarlo en mi código.

Formación en redes PON

Principalmente, no sabía mucho acerca de las redes ópticas pasivas ya que habíamos dado la asignatura de Redes pero no habíamos dado estos conceptos. Para un mayor conocimiento de estos conceptos le pedí ayuda a mi tutora de TFG la cual es una profesora especializada en esta materia y me fue de gran ayuda para mi comprensión de los conceptos.

Una vez comprendidos estos conceptos me puse con el modelado del código.

Desarrollo del programa

Para el desarrollo de este nuevo código, me puse a modelar desde cero el código para que se adaptara a este nuevo entorno de redes ópticas pasivas. Para ello creé la misma estructura que el código de las turbinas eólicas creando un fichero Python con el entorno y un fichero Jupyter Notebook donde ejecuto el programa. El nuevo entorno de Python lo modelé para unas variables las cuales serían el cuerpo del programa las cuales se pasarían por parámetro al entorno y de esta manera que el propio usuario pudiera cambiarlas cuando quisiera. Estas serían las siguientes:

- num_ont: Número de unidades ópticas que tendremos en la red.
- Bmax: Ancho de banda máximo en el que las redes pueden llegar a transmitir como máximo.
- BGarantizado: Ancho de banda garantizado en el que las redes deben retransmitir.

Lo que buscaremos en esta transmisión es que todas las unidades ópticas retransmitan el ancho de banda garantizado que se pasa. Por ello deberemos de tener más variables para tener un control de los valores que tenemos y como van variando para saber su valor:

- band_onus: vector que representa el ancho de banda actualmente asignado a cada ONU.
- previous_band_onus: vector que representa el ancho de banda asignado a cada ONU en el ciclo anterior.
- OLT_capacity: ancho de banda total que el OLT puede distribuir entre todas las ONUs.

Mi objetivo en este entorno es la minimización de la desviación de el ancho de banda que se va calculando a lo largo de los ciclos sea la menor posible, para ello ajusto la recompensa a un inverso del sumatorio de las desviaciones respecto al ancho de banda garantizado de todas las unidades ópticas, consiguiendo así que cuanta mayor sea la desviación menor será el valor del reward y por ello el algoritmo aprenderá que estos resultados no son correctos y deban de ser los más cercanos posibles. La fórmula de la recompensa en la siguiente en la que adaptamos la desviación a la mínima posible.

$$\text{average\_deviation} = \frac{1}{n}\sum_{i=1}^{n} |\text{band\_onus}_i - \text{bgarantizado}_i|$$

En la siguiente imagen se puede ver que la desviación respecto al ancho de banda garantizado es la mínima posible.

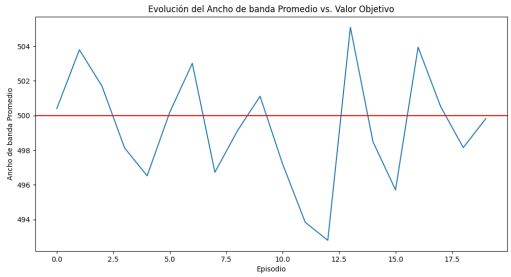

Figura 5.2: Gráfica de anchos de banda

Problemas

También debo de hablar de los problemas que tuve con este código. **El planteamiento que hice no fue correcto debido a las siguientes consideraciones.** El entorno no se correspondía a lo requerido, esto es

debido a la retransmisión que realizaba no era la correcta y la toma de los valores en el ancho de banda garantizado no era lo correcto y por ello debía corregirlo.

Gracias a mis tutores me ayudaron a afrontar el código de otra forma y realicé otra versión nueva del código; a pesar de ello remarco la importancia de este debido a la primera puesta en marcha del entorno de machine learning a las redes ópticas pasivas.

5.5. DRL Versión 2.0 Redes Ópticas Pasivas

Después de estar hablando con mis tutores decidimos dictaminar de una manera más concreta las variables con las que iba a estar trabajando además de la implementación más realista de este entorno. Para ello debía volver a documentarme acerca de estas medidas más realistas como era la distribución de pareto y su funcionamiento en el código.

Desarrollo del programa

Para este nuevo código establecí un nuevo enfoque a representar, empezando desde cero para tener una idea más clara de como debía de representar correctamente el programa. Para ello volví a seguir la misma estructura antes usada de las que voy a explicar un poco más en detalle ya que es el resultado final.

Entorno

Partí desarrollando un fichero Python en el que fui detallando más a fondo las funcionalidades de las variables con las que estaba trabajando y de una manera más en especifico para que se acercara al estudio de trabajo que deseábamos. Para ello volví a determinar unas variables que establecían el cuerpo del programa:

- **num_ont:** Número de unidades ópticas
- **v_max_olt:** Velocidad máxima de la transmisión de la OLT(lo que seria la velocidad máxima de transmisión de entrada)
- **Vt_contratada:** Velocidad máxima de transmisión que cada ont puede transmitir(lo que seria la velocidad máxima de transmisión de salida)

Una vez desarrollado el cuerpo de lo que serían mis variables, supuse que también debía guardar otros valores que me fueran a ayudar en el desarrollo del programa:

- **OLT_Capacity:** Máximo de bits de la transmisión de la OLT.
- **Max_bits_ONT:** Máximo de bits que se pueden transmitir en un ciclo en cada ont por la limitación de la velocidad contratada.
- **step_durations:** Lista para guardar la duración del tiempo del ciclo
- **trafico_entrada:** Lista para guardar el tráfico de entrada generado para cada ont.
- **trafico_pareto_futuro:** Lista para guardar el tráfico de pareto futuro, ya que debemos tomarlo en cuenta para futuros ciclos.
- **trafico_salida:** Lista para guardar el tráfico de salida generado para cada ont.
- **trafico_pareto_actual:** Lista para guardar el trafico de pareto generado en el ciclo actual para cada ont
- **trafico_pendiente:** Lista que guarda el tráfico pendiente(trafico_entrada-trafico_salida) para cada ont. Esta es una variable muy importante ya que es con la que trabajaremos el reward.

Estas variables son las que más vamos a utilizar en nuestro entorno teniendo sobre todo en cuenta los valores de **trafico_entrada, trafico_salida y trafico_pendiente**; siendo como hemos dicho el trafico_pendiente=trafico_entrada-trafico_salida; esta lógica de buscar el menor valor posible es la base en la que vamos a estar trabajando durante todo el programa para que el algoritmo PPO aprenda correctamente.[11]

En este trabajo **se ha elegido el algoritmo Proximal Policy Optimization (PPO)** para el aprendizaje por refuerzo en la simulación y optimización del tráfico en redes ópticas. La elección de PPO se basa en varias ventajas clave que lo hacen especialmente adecuado para este tipo de aplicaciones:

- **Estabilidad y Fiabilidad**: PPO introduce una técnica de actualización que limita los cambios drásticos en la política del agente, lo que mejora la estabilidad del entrenamiento. Esto es crucial para evitar comportamientos erráticos y asegurar un rendimiento consistente.

- **Eficiencia de Muestras**: PPO es altamente eficiente en el uso de muestras, permitiendo múltiples actualizaciones por cada lote de datos recolectados. Esto acelera el proceso de aprendizaje y permite obtener buenos resultados con menos datos de entrenamiento.
- **Manejo de Políticas Estocásticas**: PPO maneja políticas estocásticas de manera efectiva, lo cual es ventajoso en entornos complejos y dinámicos como las redes ópticas, donde la variabilidad en las decisiones puede mejorar el desempeño.
- **Flexibilidad en Espacios de Acción**: PPO es capaz de manejar tanto espacios de acción discretos como continuos, proporcionando una mayor flexibilidad en la modelación de diferentes escenarios de tráfico y transmisión en redes ópticas.
- **Facilidad de Implementación**: PPO es relativamente sencillo de implementar y ajustar, lo que permite concentrar los esfuerzos en la optimización del modelo sin necesidad de ajustes excesivos de hiperparámetros.

En comparación con algoritmos como Deep Q-Network (DQN), **PPO ofrece una mayor estabilidad y eficiencia, lo que resulta en un aprendizaje más rápido y robusto para la tarea específica de gestionar el tráfico en redes ópticas.** Estas características hacen de PPO la elección óptima para el desarrollo del presente proyecto.

Una vez explicado por qué hemos usado el algoritmo PPO, también debemos explicar los demás métodos, cada uno de ellos tiene una funcionalidad importante.

En el ámbito del aprendizaje por refuerzo, los entornos son componentes cruciales que simulan el entorno en el que un agente toma decisiones. Para estandarizar la interacción entre los agentes y estos entornos, se utilizan librerías como Gymnasium (anteriormente conocido como Gym) y Stable-Baselines3. Estas librerías definen una API estándar que los entornos deben seguir, facilitando así la implementación y evaluación de diversos algoritmos de aprendizaje por refuerzo.

A continuación, se describen algunos de los métodos clave que forman parte de esta API estándar, permitiendo una interacción estructurada y consistente con los entornos de aprendizaje por refuerzo:

- **__get__obs():** obtiene las observaciones del entorno

- **__get__info():** obtiene la información de las variables que le especifiquemos

- **calculate__pareto():** calcula el tráfico de entrada asignados unos valores de ON y de OFF determinados

- **calculate__reward():** Halla la recompensa, basada en que el tráfico pendiente sea el menor posible.

- **step():** Función más importante, en ella **se detallan las acciones que el programa realiza para su aprendizaje.** En ella se le pasan acciones para que el programa cambie y respecto a estos cambios que el programa pueda tener un aprendizaje correcto. En esta, realizamos todas las comprobaciones y acciones que el programa debe de hacer para el desarrollo correcto del proceso. Posteriormente en los escenarios detallaré más como funciona.

- **reset():** Este método resetea el entorno a valores default para una correcta inicialización de los valores cuando queramos borrar todo el trabajo realizado y volver a ejecutar el programa desde el inicio correctamente.

En el entorno básicamente se realizan todas las operaciones internas, pero realmente necesitamos un script que las ejecute.

Script

Este documento es un fichero **Jupyter Notebook**, el cual realizará primero unas declaraciones de variables, las cuales podemos asignar a nuestro gusto y después ejecutar el código. Pero vamos a empezar desde el principio para explicar un poco lo que realmente realiza este script y como el algoritmo PPO realiza el ejercicio de aprendizaje.

Primero tenemos la inicialización de las variables, las cuales podemos modificar su valor para que cambie el estudio del funcionamiento del programa, estas primeras variables son para la variación del entorno:

- **seed:** Esta variable es una semilla la cual se establece en el programa, la ponemos aleatoria entre 0 y 10 para que nunca de el mismo valor.

- **num__ont:** Número de unidades ópticas de la red.

- **v_max_olt:** Velocidad máxima de la transmisión de la OLT(lo que seria la velocidad máxima de transmisión de entrada). La unidad usada es la de Bps, para la representación en gráficas es en MBps, su valor usado por defecto es el de 10e9 lo cual se corresponde con 10GBps.
- **T:** tiempo de transmisión en segundos de cada ciclo(2 milisegundos).
- **OLT_CAPACITY:** Máximo de bits de la transmisión de la OLT. Esta variable tiene las unidades en Bits. Se calcula con la siguiente fórmula:

$$\text{OLT\_Capacity} = v_{\text{max\_olt}} \times T$$

- **Vt_contratada:** Velocidad máxima de transmisión que cada ont puede transmitir(lo que seria la velocidad máxima de transmisión de salida). Esta variable tiene las unidades en Bps. Su valor por defecto es el de 600e6 lo cual se corresponde con 600Mbps.
- **Max_bits_ONT:** Máximo de bits que se pueden transmitir en un ciclo en cada ont por la limitación de la velocidad contratada. Esta variable tiene las unidades en Bits.

A parte de estas variables también debemos definir más variables pero para el comportamiento del algoritmo PPO y como se va a ejecutar:

- **n_ciclos:** Número de ciclos que queremos ver retransmitidos.
- **vec_env:** Vector con los entornos establecidos.
- **n_steps:** Número de steps por actualización.
- **batch_size:** Tamaño del mini-batch, debe de ser múltiplo de n_steps (16384 es múltiplo de 256)
- **model:** Variable del algoritmo de aprendizaje por refuerzo PPO el cual crea un modelo con las variables que le introducimos.
  - **MlpPolicy:** Esta es la política que se utiliza para tomar decisiones y aprender del entorno. Esta política procesa los estados del entorno para generar acciones.
  - **vec_env:** Vector con los entornos establecidos
  - **verbose=1:** Este parámetro controla la cantidad de información que el algoritmo imprimirá mientras se entrena. Un valor de 1 significa que se imprimirá información básica como los registros de progreso del entrenamiento.

    - **n_steps:** Número de steps por actualización
    - **batch_size:** Tamaño del mini-batch, debe de ser múltiplo de n_steps (16384 es múltiplo de 256)
    - **learning_rate=0.00025:** Este es el ritmo de aprendizaje del optimizador. Un ritmo de aprendizaje más bajo puede hacer que el entrenamiento sea más estable, pero posiblemente más lento.
    - **gamma=0.99:** El factor de descuento. Un valor alto como 0.99 significa que las recompensas futuras son casi tan importantes como las recompensas inmediatas, lo que fomenta estrategias a largo plazo.
    - **gae_lambda=0.95:** Este parámetro se utiliza para el cálculo de la ventaja generalizada (Generalized Advantage Estimation)

- **num_test_episodes:** Número de episodios de prueba. Sólo realizo un episodio debido a la naturaleza del entorno pero se podría adaptar para varios episodios.
- **episode_info:** Lista para guardar la información de cada episodio.
- **list_ont:** Lista donde guardo el tráfico de entrada de cada ciclo, la variable guardada es el tráfico en cada ciclo de todas las unidades ópticas.
- **list_ont_2:** Lista donde guardo el tráfico de salida de cada ciclo, la variable guardada es el tráfico en cada ciclo de todas las unidades ópticas.
- **list_pendiente:** Lista donde guardo los valores de los bits del tráfico pendiente de cada ciclo, la variable guardada es el valor de los bits del tráfico en cada ciclo de todas las unidades ópticas.
- **estados_on_off_recolectados:** Lista donde guardo los valores de ON y de OFF del tráfico de entrada en cada ciclo de cada unidad óptica.

Una vez tenidas en cuenta estas variables, vemos como se desarrollan en el programa. Lo primero de todo que se debe de hacer es un **entrenamiento del modelo PPO**, la característica principal de los machine learnings es tener principalmente un momento de aprendizaje donde el programa aprenda el comportamiento del algoritmo gracias a los valores de reward que se le asocian y aprenderá respecto a esta variable.

Por ello establezco principalmente el método **model.learn() para el aprendizaje del modelo** y que gracias a este aprendizaje luego el test sea el deseado.

Posteriormente, este script esta modelado para en un futuro la persona que siga este trabajo **pueda trabajar con varios episodios y varios entornos**, solo que no pude terminar de implementar estas funciones.

Hay varios bucles for para dejar la base para una futura implementación de esto y una mejora significativa del rendimiento de su funcionamiento.

Lo primero que se debe de hacer es una inicialización de la observación haciendo un reset de los valores para posteriormente empezar a predecir el modelo.

Es diferente la función de learn() y la de predict(), el primero es para el proceso de entrenamiento del modelo y el segundo para una vez que el modelo ha sido entrenado, se utiliza para obtener las acciones óptimas dadas nuevas observaciones/estados del entorno.

Posteriormente en cada ciclo realizaremos el step() lo cual nos dará la variable **info** la cual es la información detallada de las variables que hayamos definido con anterioridad, gracias a esta podremos ver ciclo a ciclo como se ha comportado el programa.

Por ultimo algunas variables daban la lista de datos desorganizada, por lo que realicé las transpuestas de estas listas para una menor complicación de los datos para la representación de estos y tener en cada posición del array la unidad óptica correspondiente con todos los valores del tráfico correspondientes a los ciclos asociados.

Por ultimo, realicé gráficas para entender más visualmente el comportamiento del código y dar un resultado más agradable.

Esta explicación es una base del algoritmo, posteriormente detallo 3 escenarios concretos donde se ve el funcionamiento del programa.

Funcionamiento del programa

Una vez ya visto cuales son las variables que implementa mi algoritmo, haré una explicación un poco más detallada de como al estar variando estas podemos modificar el funcionamiento del aprendizaje de nuestro algoritmo al modificar alguna de las variables. Para ello incluiré 3 apartados los cuales son los más importantes en el aprendizaje de nuestro algoritmo.

Definición del modelo y Fase de entrenamiento Esta parte es una de las más importantes, **esto es debido a que los algoritmos de aprendizaje por refuerzo necesitan un entrenamiento previo a la ejecución para que aprendan el funcionamiento del programa.** El aprendizaje del funcionamiento lo adquieren al estar probando valores, estos valores conformarán el valor de la recompensa de según como la tengamos definida, cuando la recompensa sea positiva el algoritmo dará valores más relacionados a esta respuesta correcta. Si los valores de la recompensa son negativos, el algoritmo dará otros valores más alejados de estos para encontrar siempre los valores lo más positivos posibles. En la siguiente imagen podemos ver un ejemplo simple de como si realizamos un mal movimiento en un tres en raya, habiendo asociado la recompensa a realizar bien una jugada para ganar, el algoritmo penalizará a la inteligencia artificial por su mal resultado; por otro lado si realiza bien su movimiento dará una recompensa positiva haciendo que el algoritmo intente mejorar en su funcionamiento.

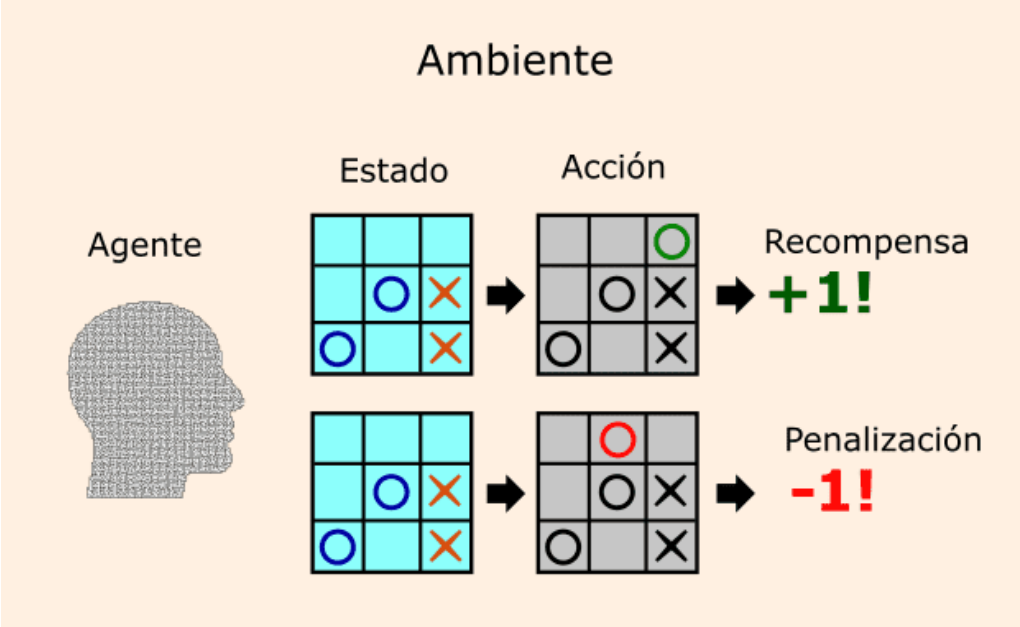

Figura 5.3: Aprendizaje Por Refuerzo[2]

En nuestro algoritmo el aprendizaje del programa lo tenemos un poco más complejo, siendo así primeramente la definición del modelo que vamos a usar y como va a aprender el programa y posteriormente la fase del aprendizaje.

Definición del modelo En la definición de nuestro modelo explicaré un poco más a fondo como realmente puede variar nuestro programa al cambiar estas variables.

Como hemos visto antes tendremos en cuenta las variables que podemos ver en la siguiente imagen:

```
n_steps = 16384  # Steps por actualización
batch_size = 256  # Tamaño del mini-batch (16384 es múltiplo de 256)

# Definir el modelo PPO con los parámetros ajustados
model = PPO(
    "MlpPolicy",
    vec_env,
    verbose=1,
    n_steps=n_steps,  # Steps por actualización
    batch_size=batch_size,  # Tamaño del mini-batch
    learning_rate=0.00025,
    gamma=0.99,
    gae_lambda=0.95
)
```

Figura 5.4: Definición del Modelo

Estas variables definen el como se va a comportar el modelo, ahora mismo están definidas de una manera correcta de la cual el programa puede funcionar correctamente aunque sacrificando el tiempo de ejecución debido al amplio número de steps que se realiza definido en el n_steps.

Modificaré estas variables con valores erráticos para que el programa se comporte de una manera no correcta. Estos serán:

- Reducir lo máximo posible la variable n_steps y batch_size, **cuantos menos steps haga menor será el aprendizaje que realice el algoritmo.**
- Aumentar el learning_rate, **al aumentar este el aprendizaje del algoritmo se volverá más errático.**
- Si se reduce el valor de gamma, en nuestro caso a 0, **el agente solo considera la recompensa inmediata sin tener en cuenta ninguna recompensa futura.**

- Si se reduce gae_lambda, en nuestro caso a 0, la estimación de ventaja solo considera la recompensa inmediata y el valor estimado en el siguiente estado, haciendo que las ventajas sean más variables y menos estables.

```
n_steps = 2  # Steps por actualización
batch_size = 2  # Tamaño del mini-batch (16384 es múltiplo de 256)

# Definir el modelo PPO con los parámetros ajustados
model = PPO(
    "MlpPolicy",
    vec_env,
    verbose=1,
    n_steps=n_steps,  # Steps por actualización
    batch_size=batch_size,  # Tamaño del mini-batch
    learning_rate=1,
    gamma=0,
    gae_lambda=0
)
```

Figura 5.5: Mala definición del Modelo

Al modificar estas variables **podemos ver en el resultado del modelo unos valores totalmente erráticos e incorrectos**, lo cual es normal ya que hemos modificado estos valores y hacen que nos den valores incorrectos aunque al poner pocos steps, que son los pasos del programa, el programa se ejecuta muy rápido. El tráfico de entrada está representado con la línea azul y el tráfico de salida está representado con la línea naranja, y esta toma valores en situaciones en las que no debería de tomar valores.

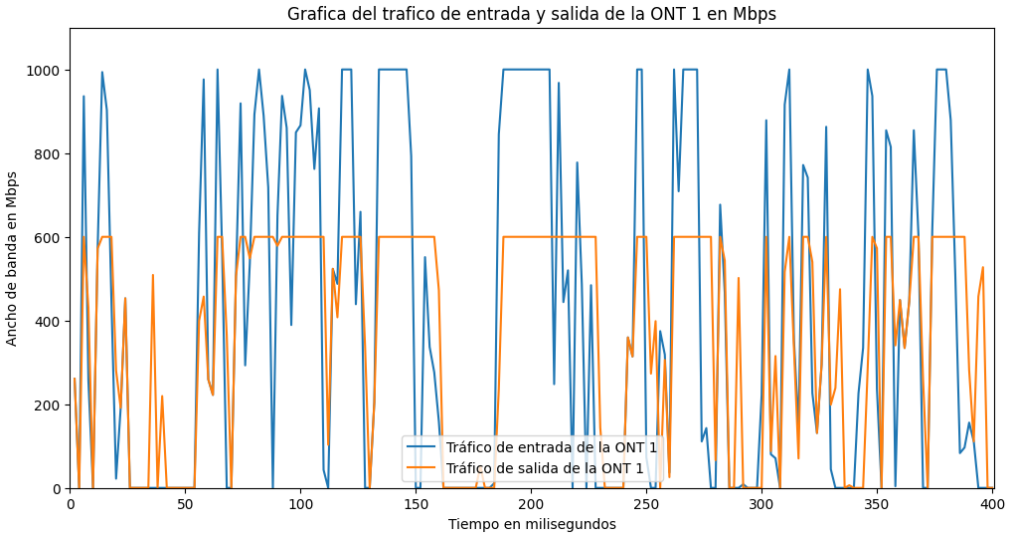

Figura 5.6: Resultado Incorrecto Modelo

La diferencia con un modelo bien entrenado es notable. En un modelo no entrenado, los valores se toman de manera aleatoria sin considerar el tráfico de salida. Por el contrario, en un modelo bien entrenado, los valores ya reflejan el tráfico de salida. En la gráfica del modelo entrenado, se observa que si no hay retransmisión de entrada y no hay tráfico pendiente, entonces no hay tráfico de salida, lo cual es el comportamiento correcto y esperado. Posteriormente en los escenarios comentaré más detalladamente como funciona cada uno.

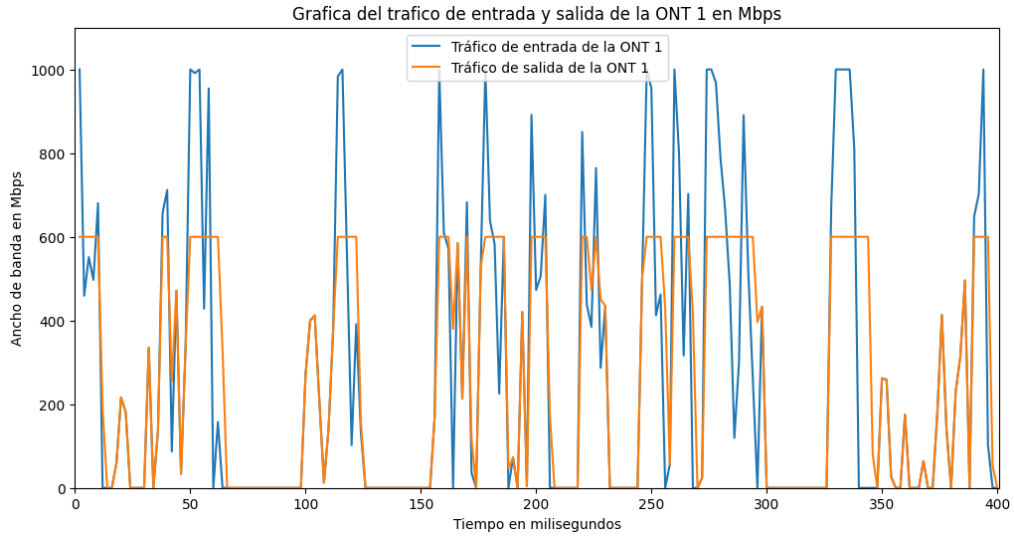

Figura 5.7: Resultado Correcto Modelo

En este resultado podemos ver un resultado más estable gracias a la implementación correcta de las variables.

Fase de entrenamiento En este apartado explicaré un poco más a fondo como realmente funciona en nuestro algoritmo la fase de entrenamiento. Tendremos en cuenta dos funciones las cuales son el **model.learn(timesteps)** y la de **__calculate__reward()**, explicaré un poco más el funcionamiento de cada una.

El método de **model.learn(timesteps)** entrena el modelo de aprendizaje por refuerzo durante el número de pasos(timesteps) que le definamos y durante este proceso el agente interactúa con el entorno, recolecta experiencias y actualiza sus políticas para maximizar la recompensa acumulada.

```
1    model.learn(total_timesteps=1000)
```

Un aprendizaje no entrenado puede desencadenar que el programa no funcione como nosotros deseemos, esto es ya que el agente no aprenderá a tomar las decisiones óptimas en el entorno.

El tiempo de entrenamiento es el siguiente:

```
El tiempo de entrenamiento fue de 16.556080102920532 segundos.
```

Figura 5.8: Tiempo de Entrenamiento timesteps=1000

A continuación pongo un ejemplo de que pasaría al poner un valor bajo de pasos.

```
1    model.learn(total_timesteps=100)
```

El algoritmo se muestra muy inestable aunque el tiempo de ejecución se reduce al tener menos pasos a ejecutar, pero de la misma manera el programa no aprende el funcionamiento de una manera correcta.

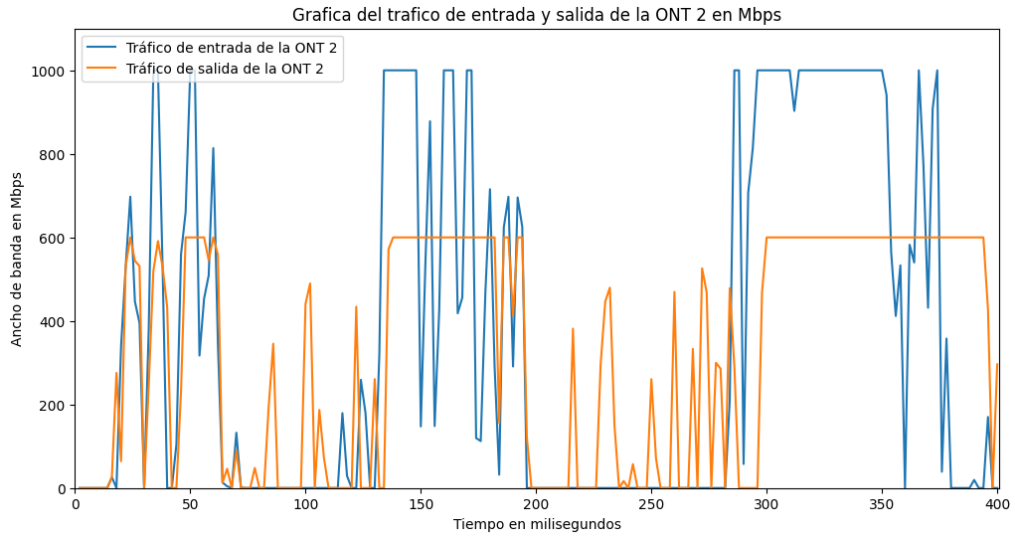

Figura 5.9: Mal funcionamiento al modificar el model.learn(timesteps)

```
El tiempo de entrenamiento fue de 1.5279772281646729 segundos.
```

Figura 5.10: Tiempo de Entrenamiento timesteps=100

Otro método importante en el apartado de la fase de entrenamiento es el comportamiento de la obtención de las recompensas, el cual es calculado por el método `_calculate_reward()`. En este método, basamos el resultado en

la suma del tráfico pendiente de todas las unidades ópticas para que sea el menor posible, lo cual es óptimo en una red, ya que minimiza la cantidad de información relevante sin enviar.

El método es el siguiente:

```
def _calculate_reward(self):
    # Penalizar fuertemente el tamaño de la cola
    para mantenerlo lo más bajo posible
    reward = -sum(self.trafico_pendiente)
    return reward
```

A continuación, se explica en detalle cómo se calcula la recompensa y el impacto de las acciones:

- **Tráfico pendiente (`trafico_pendiente`)**: Este término se refiere a la cantidad de datos (en bits por segundo, Bps) que aún no han sido enviados a través de la red.
- **Cálculo de la recompensa**: La recompensa se calcula como el negativo de la suma del tráfico pendiente en todas las unidades ópticas (ONUs). Matemáticamente, esto se expresa como:

$$\text{reward} = -\sum_{i=1}^{N} \text{trafico\_pendiente}_i$$

  donde N es el número de unidades ópticas (ONUs) y $\text{trafico\_pendiente}_i$ es el tráfico pendiente de la i-ésima unidad óptica.
- **Impacto de las acciones**: Las acciones en el entorno afectan directamente el tráfico pendiente. Acciones que optimizan el uso del ancho de banda y priorizan el envío de datos pendientes reducirán trafico_pendiente, resultando en una recompensa más alta (menos negativa). Por el contrario, acciones ineficientes que dejan acumular tráfico pendiente aumentarán la suma total de trafico_pendiente y, por lo tanto, harán que la recompensa sea más negativa.
- **Unidades usadas**: El tráfico pendiente se mide en bits por segundo (Bps). Esta métrica es crucial para evaluar la eficiencia de la red, ya que un menor tráfico pendiente implica una mejor utilización de los recursos de red y una mayor calidad de servicio.

`_calculate_reward()` es, por tanto, una función crítica que guía al agente a tomar decisiones que optimicen el rendimiento de la red, asegurando que la cantidad de información sin enviar se mantenga en un nivel mínimo.

Fase de predicción La fase de predicción también es muy importante, ya que una vez entrenado el modelo, debemos de predecir para obtener los valores de representación que va a tener, se consideraría la parte final del proceso en el que obtenemos los valores representados.

Anteriormente hemos obtenido valores, pero han sido únicamente para la fase de entrenamiento, en esta parte ya tenemos el modelo entrenado y ya en este apartado es cuando ya empezamos a obtener los resultados finales y con los que trabajaremos en la representación gráfica. Los valores devueltos son las variables de action y states, la variable action es la que realizará las acciones en nuestro entorno afectando al valor de nuestras variables, en nuestro caso al tráfico de salida para que se obtengan los valores deseados.

```
1    action, _states = model.predict(obs, state=_states,
        ↪ deterministic=True) # Usamos el modelo para predecir la
        ↪ acción
```

Implementación de la distribución de Pareto

Para la implementación de la distribución de Pareto establecí, gracias a la ayuda de mi tutor, de un método aparte para el cálculo del tráfico de entrada. Con este método estableceremos los estados de ON y OFF del tráfico de entrada según las variables de ON y de OFF ya establecidos. Gracias a esto podemos establecer la velocidad media de transmisión que ofrece la red.

Detallaré un poco las las fórmulas relacionadas con la gestión del tráfico de Pareto en redes ópticas, junto con una breve explicación de cada una.

Distribución de Pareto

La función de densidad de probabilidad (pdf, *probability density function*) de la distribución de Pareto se define como:

$$p(x) = \frac{\alpha}{(1+x)^{\alpha+1}}, \quad (x > 0, \alpha > 1)$$

Esta distribución se utiliza para modelar el tráfico en las redes ópticas, donde α es un parámetro que determina la forma de la distribución. Concretamente,

los tiempos de actividad (ON) e inactividad (OFF) se modelan como variables aleatorias con esta distribución.

Valor Esperado

El valor promedio o esperado $\bar{x}$ de la distribución de Pareto se calcula como:

$$\bar{x} = \int_0^\infty x \cdot p(x)\, dx = \int_0^\infty \frac{\alpha x}{(1+x)^{\alpha+1}}\, dx$$

Esta integral representa el valor promedio o esperado de un conjunto de datos extraídos con esta distribución. En nuestro caso, los valores son los tiempos que un cliente transmite o no durante el tráfico a ráfagas generado.

Solución de la Integral

La solución de la integral anterior se obtiene como:

$$\int_0^\infty \frac{x}{(1+x)^{\alpha+1}}\, dx = \left[\frac{x}{-\alpha(1+x)^{\alpha}}\right]_0^\infty = 0 + \frac{1}{\alpha - 1}$$

Esta simplificación muestra que el valor esperado está relacionado inversamente con el parámetro α.

Tiempo de Activación (ON)

El tiempo promedio que una unidad óptica (ONT) está activa (ON) se define como:

$$T_{\text{on}} = \frac{1}{\alpha_{\text{on}} - 1}$$

Este valor depende del parámetro α_{on}, que caracteriza la distribución del tiempo de actividad.

Tiempo de Inactividad (OFF)

Similarmente, el tiempo promedio que una ONT está inactiva (OFF) se define como:

$$T_{\text{off}} = \frac{1}{\alpha_{\text{off}} - 1}$$

Este valor depende del parámetro α_{off}, que caracteriza la distribución del tiempo de inactividad.

Relación entre Tiempos de Actividad e Inactividad

La relación entre los tiempos de activación e inactividad se denomina en este contexto carga o duty cycle y se puede expresar como:

$$carga = \frac{\frac{1}{\alpha_{\text{on}}-1}}{\frac{1}{\alpha_{\text{on}}-1}+\frac{1}{\alpha_{\text{off}}-1}} = \frac{\alpha_{\text{off}}-1}{\alpha_{\text{off}}-1+\alpha_{\text{on}}-1} = \frac{\alpha_{\text{off}}-1}{\alpha_{\text{off}}+\alpha_{\text{on}}-2}$$

Esta ecuación muestra que la carga depende tanto de α_{on} como de α_{off}.

Carga de la Red

Finalmente, se puede establecer una relación simplificada cuando $\alpha_{\text{off}} = 2$:

$$\alpha_{\text{off}} = 2 \Rightarrow \frac{1}{\alpha_{\text{on}}} = \text{carga}$$

Esto indica que la carga de la red está inversamente relacionada con el parámetro α_{on}.

Código

Una vez vistas las fórmulas de cómo debemos de gestionar la distribución de Pareto en nuestro código, modelamos el código necesario. Para ello deberemos de primero realizar la inicialización de los parámetros alpha_on y alpha_off que definen las formas de las distribuciones de Pareto para los estados ON y OFF, respectivamente. También se crean listas vacías para almacenar los valores futuros de tráfico (trafico_futuro_valores), el tráfico actual (lista_trafico_act) y una lista de listas para el tráfico actual por ONT (trafico_actual_lista).

La primera parte del código se encarga de generar el tráfico utilizando la distribución de Pareto para cada Optical Network Terminal (ONT):

```
1  if not traf_pas:
2      trafico_pareto = list(self.rng.pareto(alpha_on, size=1))
3      trafico_pareto += list(self.rng.pareto(alpha_off, size=1))
4  else:
5      trafico_pareto = traf_pas[i]

7  suma = sum(trafico_pareto)
8  while suma < 2:
9      trafico_pareto += list(self.rng.pareto(alpha_on, size=1)) +
10     list(self.rng.pareto(alpha_off, size=1))
11     suma = sum(trafico_pareto)
```

El código realiza las siguientes acciones:

- **Condición Inicial:**
  - Si no se proporciona una lista de tráfico pasado (`traf_pas`):
    - Se generan muestras de la distribución de Pareto con parámetros `alpha_on` y `alpha_off`.
    - `self.rng.pareto(alpha_on, size=1)`: Genera una muestra de Pareto con el parámetro `alpha_on`.
    - `self.rng.pareto(alpha_off, size=1)`: Genera una muestra de Pareto con el parámetro `alpha_off`.
    - Las muestras generadas se almacenan en la lista `trafico_pareto`.
  - Si se proporciona una lista de tráfico pasado (`traf_pas`):
    - Se utiliza directamente el tráfico pasado correspondiente a la ONT actual (`traf_pas[i]`).
    - El tráfico se almacena en la lista `trafico_pareto`.
- **Validación del Tráfico Generado:**
  - Se calcula la suma de los valores en `trafico_pareto`.
  - Mientras la suma sea menor que 2, se generan y añaden nuevas muestras de Pareto con los parámetros `alpha_on` y `alpha_off` a `trafico_pareto`.
  - Este proceso asegura que el tráfico generado sea suficiente para ser considerado válido, es decir, que su suma sea al menos 2.

La segunda parte del código se encarga de procesar el tráfico generado y separarlo en tráfico actual y futuro para cada Optical Network Terminal (ONT):

```
1  traf_act = []
2  suma = 0
3  while suma < 2:
4      traf_act.append(trafico_pareto.pop(0))
5      suma = sum(traf_act)

7  traf_fut = [0, 0]
8  if len(traf_act) % 2 == 0:
9      traf_fut[0] = 0
10     traf_fut[1] = suma - 2
```

```
11      traf_act[-1] -= traf_fut[1]
12  else:
13      traf_fut[0] = suma - 2
14      traf_fut[1] = trafico_pareto[-1]
15      traf_act[-1] -= traf_fut[0]

17  trafico_actual_lista[i].append(traf_act)
18  vol_traf_act = sum(traf_act[::2]) * vel_tx_max * 10e-3
19  lista_trafico_act.append(vol_traf_act)
20  trafico_futuro_valores.append(traf_fut)

22  return lista_trafico_act, trafico_actual_lista,
        ↪ trafico_futuro_valores
```

El código realiza las siguientes acciones:

- **Separación del Tráfico Actual**:
  - Se inicializa una lista vacía `traf_act` para almacenar el tráfico actual.
  - Se suma el tráfico de `trafico_pareto` a `traf_act` hasta que la suma de `traf_act` sea al menos 2.
  - Este proceso asegura que el tráfico actual sea suficiente para ser procesado.
- **Determinación del Tráfico Futuro**:
  - Se inicializa una lista `traf_fut` con dos elementos en 0.
  - Si la longitud de `traf_act` es par:
    - `traf_fut[0]` se mantiene en 0.
    - `traf_fut[1]` se establece como la diferencia entre la suma de `traf_act` y 2.
    - El último valor de `traf_act` se ajusta restando `traf_fut[1]`.
  - Si la longitud de `traf_act` es impar:
    - `traf_fut[0]` se establece como la diferencia entre la suma de `traf_act` y 2.
    - `traf_fut[1]` se establece como el último valor de `trafico_pareto`.
    - El último valor de `traf_act` se ajusta restando `traf_fut[0]`.
- **Actualización de Listas**:

- Se añade `traf_act` a la lista correspondiente en `trafico_actual_lista`.
- Se calcula el volumen de tráfico actual (`vol_traf_act`) como la suma de los valores de `traf_act` en los estados ON, multiplicado por la velocidad máxima de transmisión y un factor de tiempo.
- Se añade `vol_traf_act` a `lista_trafico_act`.
- Se añade `traf_fut` a `trafico_futuro_valores`.

▪ **Retorno de Resultados**:

- El método retorna tres listas: `lista_trafico_act`, `trafico_actual_lista` y `trafico_futuro_valores`.

Este método no solo facilita la generación y procesamiento de tráfico en un entorno simulado, sino que también **proporciona una base sólida para realizar estudios y optimizaciones en la gestión de redes ópticas.** La flexibilidad y precisión del algoritmo aseguran que se puedan realizar análisis detallados, contribuyendo así a la mejora continua en el diseño y operación de estas redes. Este enfoque, basado en modelos probabilísticos realistas, es esencial para afrontar los desafíos actuales y futuros en el campo de las telecomunicaciones.

Funcionamiento paso a paso

Una vez ya sabiendo cada método y función, explicaré un poco el funcionamiento paso a paso que realiza el código para la ejecución completa del programa, por donde empieza a ejecutarse y como realiza apropiadamente el aprendizaje por refuerzo.

La ejecución es iniciada en el archivo Jupyter Notebook, más en concreto en el siguiente:

```
1    if __name__ == "__main__":
```

Las siguientes lineas de código son las **inicializaciones del valor de las variables que usaremos en el programa.**

```
1    env_id = 'RedesOpticasEnv-v0' # Hay que asegurarse de que este
         ↪ ID coincida con el registrado
2    num_test = 20 #Ponemos el número de test que necesitamos para
         ↪ que el algoritmo de aprendizaje aprenda.
3    seed = np.random.randint(0, 10) #Ponemos seeds aleatorias
4    num_envs = 1 # Número de entornos paralelos
```

```
    num_ont=16
    #Establecemos el v_max_olt
    #10 Gpbs (XGSPON)
    v_max_olt=10e9
    #Transmision de cada ciclo
    T=0.002
    #OLT Capacity
    OLT_Capacity=v_max_olt*T
    #Velocidad de transmision contratada
    vt_contratada=600e6
    #Maximo de bits que se pueden transmitir en un ciclo en cada
        ↪ ont por la limitacion de la velocidad contratada
    Max_bits_ONT=vt_contratada*T
    #número de ciclos que se van a ejecutar
    n_ciclos=int(input("Cuantos ciclos quiere ver: "))
```

Lo siguiente que realizamos es inicializar el vector con los entornos establecidos, llamando al método make_env el cual nos inicializa el entorno con el que trabajará el algoritmo de aprendizaje por refuerzo con las variables que le pasamos por parametro.

```
    vec_env = DummyVecEnv([make_env(num_ont, v_max_olt,
        ↪ vt_contratada,n_ciclos,
    rank=i, seed=42) for i in range(num_envs)])

    def make_env(num_ont, v_max_olt=10e6,vt_contratada=10e6/10,
        ↪ n_ciclos=200,
    rank=0, seed=0):
    def _init():
        env = RedesOpticasEnv(render_mode=None,seed=seed, num_ont=
            ↪ num_ont, v_max_olt=v_max_olt, vt_contratada=
            ↪ vt_contratada,n_ciclos=n_ciclos)
        return env
    return _init
```

La inicialización del entorno podemos ver que es lo que realiza en el fichero Python de redes_opticas_env.py en el metodo de init donde lo que se realiza es la inicialización de todas las variables que se van a ejecutar en el programa.

```
    def __init__(self, render_mode=None, seed=0, num_ont=3,
        ↪ v_max_olt=10e6, vt_contratada=10e6/10, n_ciclos=200):
        self.num_ont = num_ont #número de ont(unidades opticas)
```

```
3         self.v_max_olt = v_max_olt # bits por segundo (bps)
4         self.temp_ciclo = 0.002 # segundos (s)
5         self.OLT_Capacity = v_max_olt * self.temp_ciclo # bits
6         #Velocidad de transmision contratada
7         self.velocidadContratada = vt_contratada
8         #Maximo de bits que se pueden transmitir en un ciclo en
              ↪ cada ont por la limitacion de la velocidad
              ↪ contratada
9         self.Max_bits_ONT=self.velocidadContratada*self.temp_ciclo

11        self.observation_space = spaces.Box(low=0, high=self.
              ↪ Max_bits_ONT,
12        shape=(self.num_ont,), dtype=np.float32)
13        self.action_space = spaces.Box(low=-self.Max_bits_ONT,
14        high=self.Max_bits_ONT, shape=(self.num_ont,), dtype=np.
              ↪ float32)

16        self.step_durations = [] #Guardar duracion de tiempo del
              ↪ ciclo
17        self.trafico_entrada = [] #Guardar el trafico de entrada en
              ↪  cada ont
18        self.trafico_pareto_futuro = [] #Guardar el
              ↪ trafico_pareto_futuro
19        self.trafico_salida = [] #Guardar el trafico de salida en
              ↪ cada ont
20        self.trafico_pareto_actual = [] #Guardar el trafico pareto
              ↪ actual
21        self.trafico_pendiente = np.zeros(self.num_ont) #
              ↪ Inicializar el
22        tráfico pendiente para cada ONT

24        self.rng = np.random.default_rng(seed) # Inicializa el
              ↪ generador de números aleatorios

26        #Variable propia de este escenario donde se cambia el
              ↪ funcionamiento del algoritmo
27        self.instantes=0

29        #Variable con la que decimos el número de ciclos del
              ↪ algoritmo
30        self.n_ciclos=n_ciclos-1

32        self.state = None
```

```
33          self.reset()
```

Una vez inicializado el entorno y establecido en la variable vec_env procedemos a definir el modelo PPO, que como hemos dicho anteriormente nos ofrece una ofrece una mayor estabilidad y eficiencia además de su fácil implementación.

```
1       n_steps = 16384 # Steps por actualización
2       batch_size = 256 # Tamaño del mini-batch (16384 es múltiplo de
            ↪ 256)

4       # Definir el modelo PPO con los parámetros ajustados
5       model = PPO(
6           "MlpPolicy",
7           vec_env,
8           verbose=1,
9           n_steps=n_steps, # Steps por actualización
10          batch_size=batch_size, # Tamaño del mini-batch
11          learning_rate=0.00025,
12          gamma=0.99,
13          gae_lambda=0.95
14      )
15      model.learn(total_timesteps=1000)
```

La siguiente parte es la inicialización de variables auxiliares para la ejecución y guardado de datos del programa.

```
1       # Fase de pruebas

3       #Establecemos un unico episodio a investigar
4       num_test_episodes = 1 # Número de episodios de prueba

6       # Lista para guardar la información de cada episodio
7       episode_info = []

9       # Lista de en cada ont guardar el valor de su capacidad, de
10      entrada salida y del pendiente(entrada-salida)
11      list_ont = []
12      list_ont_2 = []
13      list_pendiente=[]

15      # Guardar los estados de ON y OFF del estado de pareto
16      estados_on_off_recolectados = []
```

```
18      #Capacidad de la OLT
19      tamano_cola=[]
```

En este momento ya empezamos a entrar en el groso del programa, donde primero se llama a la función de **reset()** donde se inicializan los valores de las variables del entorno a sus valores por defecto para una correcta ejecución de los datos desde el principio. Esto se realiza para que **los valores con los que hemos estado entrenando el entorno no nos afecten a la ejecución del programa**, es muy importante diferenciar aquí los valores con los que hemos estado trabajando en el entrenamiento y ahora lo que realizamos es establecer los valores por defecto para volver a ejecutar los datos pero ahora con el entorno entrenado. La demás parte del código son inicializaciones de las variables, el done es la característica que hace que el programa termine de ejecutar los pasos, en nuestro caso esta definido para que una vez llega al ciclo determinado pase a True y detenga la ejecución del programa.

```
1       obs = vec_env.reset() # Resetea el entorno al estado inicial
2       _states = None # Inicializa el estado del modelo
3       for episode in range(num_test_episodes):

5           done = np.array([False]*num_envs) # Inicializa 'done'
6           para todos los entornos
7           step_counter = 0 # Contador de steps para limitar al número
                ↪  de ciclos

9       def reset(self, seed=None, options=None):
10          self.trafico_entrada, self.trafico_pareto_actual,
11          self.trafico_pareto_futuro = self.calculate_pareto(self.
                ↪ num_ont,
12          self.trafico_pareto_futuro)
13          self.trafico_salida = self.rng.uniform(low=self.
                ↪ Max_bits_ONT/10,
14          high=self.Max_bits_ONT, size=self.num_ont).astype(np.
                ↪ float32)

16          self.trafico_pendiente = np.zeros(self.num_ont) #
                ↪ Inicializar el tráfico pendiente para cada ONT

18          self.rng = np.random.default_rng(seed)
```

```
20          observation = self._get_obs()
21          info = self._get_info()
22          return observation, info
```

El siguiente punto será lo más importante del programa, para ello realizo un bucle while el cual no va a acabar hasta que se ejecuten todos los ciclos que hemos puesto. También se encuentra las partes más importantes, **en cada ciclo se debe de ejecutar la predicción del programa con los valores dados**, de primeras el valor de la observación y del estado esta nula y nos devolverán los valores de la acción que tome el programa y el estado en el que se encuentra. Con el valor de la acción recibida realizaremos el **step()**, la función más importante del programa, en la que **contiene todas las lineas de código para que el programa realice la ejecución determinada en ese ciclo,** devolviendo el valor de la observación dada, la recompensa obtenida, el estado del done y la variable info para saber los valores de las variables que nosotros definamos.

```
1       while step_counter < n_ciclos:

3           action, _states = model.predict(obs, state=_states,
                ↪ deterministic=True)
4           # Usamos el modelo para predecir la acción
5           obs, rewards, dones, info = vec_env.step(action)
```

Como es una función importante la explicare un poco más a fondo.

Lo primero que realizamos es la inicialización de una variable tiempo en la que podremos ajustar el tiempo de cada ciclo, luego explicaré como la variable start_time funciona.

```
1       def step(self, action):
2           start_time = time.time()
```

La siguiente parte del código es la obtención de los valores del tráfico de entrada, tráfico de pareto actual y el tráfico de pareto futuro en nuestro programa para poder ver en este ciclo en cada ONT cuales son los valores de cada una de estas variables. También, con la acción que hemos pasado al metodo step() por parametro, **la línea siguiente asegura que el tráfico de salida determinado por el agente se mantenga dentro de límites razonables y prácticos,** contribuyendo a un comportamiento más efectivo y seguro del sistema de redes ópticas.

```
    # Obtener el tráfico de entrada actual
        self.trafico_entrada, self.trafico_pareto_actual,
        self.trafico_pareto_futuro = self.calculate_pareto(self.
            ↪ num_ont,
        self.trafico_pareto_futuro)

        # Considerar el tráfico pendiente en el cálculo del tráfico
            ↪  de salida
        self.trafico_salida = np.clip(action, 0, self.Max_bits_ONT)
```

En las siguientes lineas es donde **calculamos los valores del tráfico pendiente para cada ONT**, se actualiza el tráfico pendiente sumando el tráfico de entrada menos el tráfico de salida del ciclo actual. Esta operación se representa por la siguiente fórmula:

$$\begin{aligned}\texttt{trafico\_pendiente}[i] =& \texttt{trafico\_pendiente}[i] + \\ &+ (\texttt{trafico\_entrada}[i] - \texttt{trafico\_salida}[i])\end{aligned} \tag{5.1}$$

Si el **tráfico pendiente es mayor que cero, se ajusta el tráfico de salida para el siguiente ciclo de la siguiente manera:**

- El tráfico de salida se establece como el mínimo entre el tráfico pendiente y la capacidad máxima de bits que se pueden transmitir en un ciclo (`Max_bits_ONT`).
- Luego, se reduce el tráfico pendiente por la cantidad de tráfico de salida ajustado.

$$\texttt{trafico\_salida}[i] = \text{mín}(\texttt{trafico\_pendiente}[i], \texttt{Max\_bits\_ONT}) \tag{5.2}$$

$$\texttt{trafico\_pendiente}[i] = \texttt{trafico\_pendiente}[i] - \texttt{trafico\_salida}[i] \tag{5.3}$$

```
    # Asegurar que si hay tráfico pendiente, se ajuste
        ↪ adecuadamente el tráfico de salida
        for i in range(self.num_ont):
```

```
            self.trafico_pendiente[i] += self.trafico_entrada[i] -
                ↪ self.trafico_salida[i]
            if self.trafico_pendiente[i] > 0:
                # Asegurarse de que el tráfico de salida en el
                    ↪ siguiente
                ciclo considera el tráfico pendiente
                self.trafico_salida[i] = min(self.trafico_pendiente[
                    ↪ i],
                self.Max_bits_ONT)
                self.trafico_pendiente[i] -= self.trafico_salida[i]
```

En esta parte del código **se verifica si la suma del tráfico de salida para todas las ONTs excede la capacidad del OLT.** También, si la suma excede la capacidad del OLT **se calcula el exceso de tráfico distribuyéndose equitativamente entre todas las ONTs**, ajustando el tráfico de salida de cada ONT para asegurar que el tráfico de salida total no exceda la capacidad del OLT, distribuyendo el exceso de manera uniforme entre todas las ONTs.

```
    # Asegurarse de que la suma del tráfico de salida no supere la
        ↪ capacidad del OLT
        if np.sum(self.trafico_salida) > self.OLT_Capacity:
            exceso = np.sum(self.trafico_salida) - self.
                ↪ OLT_Capacity
            self.trafico_salida -= (exceso / self.num_ont) #
                ↪ Distribuir el exceso entre todas las ONTs
```

En las siguientes lineas se obtiene la recompensa determinada, además de tener en cuenta la finalización de la función del step determinando si la variable de instantes, la cual **lleva el recuento de los pasos que se han ido realizando en la fase de ejecución del programa,** es igual al número de ciclos que nosotros hemos determinado al principio.

```
    # Calcular recompensa
        reward = self._calculate_reward()

        if self.instantes==self.n_ciclos:
            done=True
        else:
            done=False

        self.instantes+=1
```

Por ultimo, una vez ya ejecutado todo lo que debe de hacer el método de step(), determinamos el tiempo que ha tardado en ejecutar este método, indicando a placer de cuanto ha tardado en ejecutar el programa o incluso pudiendo dormir el programa por si la ejecución ha sido más rápida de lo que deseamos. Por ahora lo dejamos comentado ya que la ejecución se realiza correctamente pero para un simulador en el que se necesite obligatoriamente que los ciclos duren una determinada duración se modificaría desde esta variable.

```
        """
        elapsed_time = time.time() - start_time
        if elapsed_time < 0.002:
            time.sleep(0.002 - elapsed_time)
        """

        end_time = time.time()
        step_duration = end_time - start_time
        self.step_durations.append(step_duration)
```

Por último, **guardamos la información de los valores de las variables definidas en la información y devolvemos los valores para que podamos representar los datos de cada ciclo**. Entre estos se encuentran, la capacidad de la OLT, el tráfico de entrada de cada ont, el tráfico de salida de cada ONT, el tráfico de Pareto de cada ONT y el tráfico pendiente de cada ONT.

```
    info = self._get_info()

        return self._get_obs(), reward, done, False, info

    def _get_info(self):
        info = {
            'OLT_Capacity': self.OLT_Capacity,
            'trafico_entrada': self.trafico_entrada,
            'trafico_salida': self.trafico_salida,
            'trafico_IN_ON_actual': self.trafico_pareto_actual,
            'trafico_pendiente': self.trafico_pendiente
        }
        return info
```

Para guardar los valores, guardaremos por si necesitáramos más episodios los valores del info, también sobre cada entorno(por esto el bucle for por si

tuviéramos más entornos paralelos) guardamos la información de cada uno de los valores para luego representarlos en las gráficas.

```
# Guardamos la información del episodio.
    episode_info.append(info)
    for i in range(len(info)): # Itera sobre cada sub-entorno

        suma = 0

        list_ont.append(info[i]['trafico_entrada'])
        list_ont_2.append(info[i]['trafico_salida'])
        list_pendiente.append(info[i]['trafico_pendiente'])
        estados_on_off_recolectados.append(info[i]['
            ↪ trafico_IN_ON_actual'])

    done |= dones # Actualiza 'done' para todos los entornos
    step_counter += 1 # Incrementa el contador de steps
```

```
    # Guardamos la información del episodio.
            episode_info.append(info)
            for i in range(len(info)): # Itera sobre cada sub-
                ↪ entorno

                suma = 0

                list_ont.append(info[i]['trafico_entrada'])
                list_ont_2.append(info[i]['trafico_salida'])
                list_pendiente.append(info[i]['trafico_pendiente'])
                estados_on_off_recolectados.append(info[i]['
                    ↪ trafico_IN_ON_actual'])

            done |= dones # Actualiza 'done' para todos los
                ↪ entornos
            step_counter += 1 # Incrementa el contador de steps
```

Las siguientes líneas son funciones auxiliares para la transposición de las variables para su representación en las gráficas.

Impacto de las Redes PON en el Código

Las redes ópticas pasivas (PON) presentan un desafío particular en la gestión del ancho de banda debido a la **necesidad de compartir eficiente-**

mente una capacidad limitada entre múltiples usuarios. En nuestro proyecto, hemos implementado un algoritmo de aprendizaje por refuerzo profundo (PPO) para optimizar esta asignación de recursos. A continuación, se detalla cómo las características de las redes PON se reflejan en nuestro código y la lógica detrás de la distribución del ancho de banda.

Control del Ancho de Banda Basado en SLA

El principal objetivo es **garantizar que el ancho de banda ofrecido a las ONUs** esté en línea con los acuerdos de nivel de servicio (SLA) establecidos. Esto se logra mediante varias operaciones clave en el código:

- **Ajuste del Tráfico de Salida**: La acción del agente determina el tráfico de salida para cada ONU, asegurándose de que este valor esté dentro de los límites permitidos.

```
self.trafico_salida = np.clip(action, 0, self.Max_bits_ONT)
```

  Esta línea garantiza que el tráfico de salida no supere la capacidad máxima contratada (`Max_bits_ONT`), manteniendo el cumplimiento de los SLA.

- **Consideración del Tráfico Pendiente**: Se ajusta el tráfico pendiente para cada ONU, acumulando cualquier exceso de tráfico de entrada que no haya sido transmitido en el ciclo actual.

```
for i in range(self.num_ont):
    self.trafico_pendiente[i] += self.trafico_entrada[i] - self.
        ↪ trafico_salida[i]
    if self.trafico_pendiente[i] > 0:
        self.trafico_salida[i] = min(self.trafico_pendiente[i],
            ↪ self.Max_bits_ONT)
        self.trafico_pendiente[i] -= self.trafico_salida[i]
```

  Esta lógica asegura que cualquier tráfico pendiente se considera en los ciclos futuros, permitiendo una distribución más equitativa y eficiente del ancho de banda.

- **Capacidad del OLT**: Para evitar sobrecargar el OLT, se verifica que la suma total del tráfico de salida no exceda su capacidad máxima. Si lo hace, se distribuye el exceso de manera uniforme entre todas las ONUs.

```
if np.sum(self.trafico_salida) > self.OLT_Capacity:
    exceso = np.sum(self.trafico_salida) - self.OLT_Capacity
    self.trafico_salida -= (exceso / self.num_ont)
```

Esto garantiza que el tráfico de salida total se mantenga dentro de los límites operativos del OLT, evitando congestiones y asegurando una operación fluida de la red.

- **Cálculo de la Recompensa y Finalización del Episodio**: La recompensa se calcula en función del desempeño del agente, y se verifica si se ha alcanzado el número máximo de ciclos para determinar el final del episodio.

```
reward = self._calculate_reward()

if self.instantes == self.n_ciclos:
    done = True
else:
    done = False

self.instantes += 1
```

Este mecanismo proporciona retroalimentación al agente, guiándolo hacia políticas que maximicen la eficiencia y el cumplimiento de los SLA.

Escenarios de Simulación

Nuestro proyecto **considera diferentes escenarios de simulación para evaluar el desempeño del algoritmo PPO** en la gestión del ancho de banda en redes PON. Estos escenarios varían en términos de demanda de tráfico, condiciones de la red y configuraciones de SLA, permitiendo una evaluación exhaustiva del sistema bajo condiciones realistas y desafiantes.

En cada escenario, el agente de aprendizaje por refuerzo se entrena y prueba para optimizar la asignación de recursos, asegurando un equilibrio entre eficiencia de la red y satisfacción del usuario.

Para ello detallaré en los siguientes apartados cada escenario para que quede lo más claro posible su desempeño.

Escenario 1

En este primer escenario detallo la funcionalidad del algoritmo en unas condiciones específicas. Estas condiciones serán las siguientes:

- Todas las unidades ópticas transmiten la misma carga media de **900 Mbps.** Este valor se establece por los datos de **ON=1.4** y de **OFF=1.2.**
- La velocidad de transmisión máxima de la ONT es de **10 Gbps.**
- El ancho de banda garantizado máximo para cada unidad óptica debe de ser de **600 Mbps.**
- El número de unidades ópticas es de **16.**
- El número de ciclos se puede elegir, yo lo he puesto a **200 ciclos.**

Con todos estos datos y habiendo explicado el código explico un poco los resultados dados por las gráficas.

Inicialmente voy a explicar la representación de cada dato, tomaré como referencia los valores de la primera unidad óptica ya que es interesante los datos para su observación. En la siguiente imagen podemos ver 3 lineas de Output:

- La primera linea corresponde a una **pequeña parte de los valores del tráfico de entrada en cada ciclo de la primera unidad óptica.**
- La segunda linea corresponde a una **pequeña parte de los valores del tráfico de salida en cada ciclo de la primera unidad óptica.**
- La tercera linea corresponde a los **valores de los instantes de ON y de OFF en cada ciclo** de la primera unidad óptica. Con estos podremos ver cada uno de estos en más perspectiva de como actúa la distribución de pareto en las redes ópticas pasivas.

```
Valores de la ONT 0 de entrada son: [962.7109944653586, 669.9607861626982, 793.7102134694197, 0.0, 0.0, 0
Valores de la ONT 0 de salida son: [600.0, 600.0, 600.0, 600.0, 600.0, 600.0, 600.0, 600.0, 600.0, 600.0,
Los valores de los instantes de la ont 1 son de [[0, 0.07457801106928308, 1.925421988930717], [0.04146400
```

Figura 5.11: Escenario 1 Valores

La primera gráfica representada corresponde a la **representación de los valores del tráfico de entrada y de salida.** Podemos ver que el tráfico de entrada(representado con la linea azul) no sobrepasa el máximo permitido de 1000 Mbps. El tráfico de salida(representado con la linea naranja) tampoco sobrepasa su máximo impuesto de 600 Mbps. También para una mejor visualización de los datos, el eje x se ha puesto a milisegundos para la comprensión de nuestro algoritmo. Si hemos decidido 200 ciclos, cada ciclo corresponde a 2ms lo que hace que representemos 400 milisegundos.

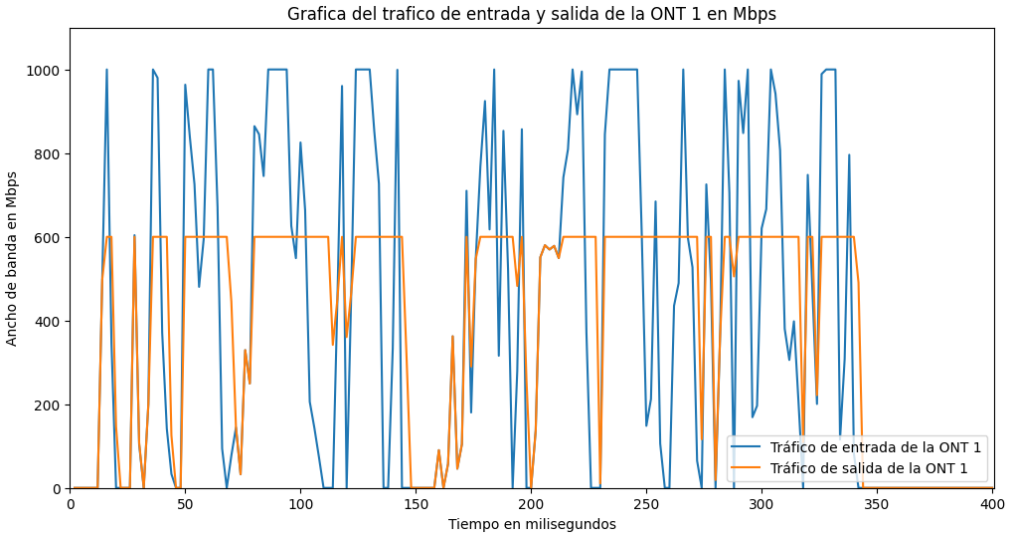

Figura 5.12: Escenario 1 Trafico de entrada y salida

La segunda gráfica corresponde a **los instantes de ON y de OFF.** Aquí vemos como varían los valores respecto a un eje x el cual es un ciclo temporal y como los picos de la anterior gráfica entre los milisegundos de esta gráfica se puede ver en que estados el tráfico de entrada ha retransmitido o no. Los valores de ON es cuando se retransmite y OFF cuando no retransmite nada. **En este ejemplo nos debemos de fijar en los 40 milisegundos de la anterior gráfica para ver como han estado afectando los valores de ON y de OFF para la retransmisión de la OLT.**

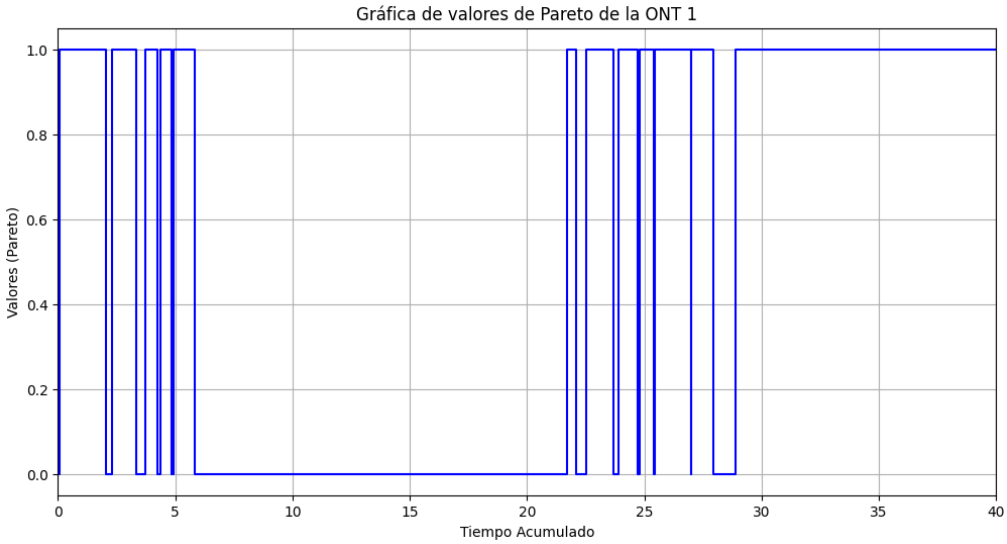

Figura 5.13: Escenario 1 Tráfico de Pareto

La tercera gráfica se corresponde a **como evoluciona la carga del tráfico pendiente a lo largo del tiempo.** Cuanta más carga de tráfico de entrada respecto a la salida haya, el valor del tráfico pendiente aumentará; en cambio si el tráfico de salida es mayor que el tráfico de entrada la carga de bits pendiente será menor. Por ello nos debemos de fijar en la primera gráfica en la diferencia del tráfico de entrada y salida para ver cuanta carga de tráfico hay. Si no hay tráfico pendiente y el tráfico de salida es mayor al de entrada o el tráfico de entrada en ese momento no retransmite, el tráfico de salida no debe de retransmitir, esto lo podemos ver en los últimos instantes donde el tráfico de salida no retransmite y en esta gráfica que no hay tráfico pendiente.

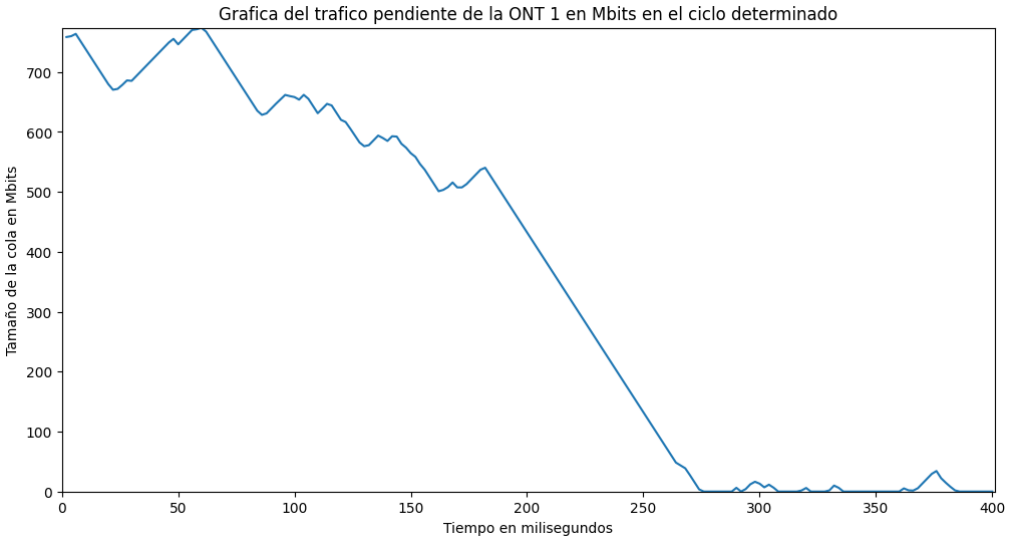

Figura 5.14: Escenario 1 Carga Pendiente

Escenario 2

El escenario segundo es una **variación del primero,** solo que trabajamos con otros valores. Este escenario definimos las siguientes condiciones:

- Las unidades ópticas tienen diferentes cargas, dividiremos las unidades ópticas en **3 grupos.** El primer grupo retransmitirá a **900 Mbps** de media cuya carga será de **ON=0.9**, el segundo grupo retransmitirá a **800 Mbps** de media cuya carga será de **ON=0.8** y el tercer grupo retransmitirá a **700 Mbps** de media cuya carga será de **ON=0.7.**
- La velocidad de transmisión máxima de la ONT es de **10 Gbps.**
- El ancho de banda garantizado máximo para cada unidad óptica debe de ser de **600 Mbps.**
- El número de unidades ópticas es de **16.**
- El número de ciclos se puede elegir, yo lo he puesto a **200 ciclos.**

Este escenario es muy similar al escenario uno, salvo que debemos de adaptar los datos anteriores a los actuales. **En vez de trabajar con valores de ON de tipo Integer, debemos de trabajar con listas.** Para esto estableceremos una lista de valores de ON en el que para cada unidad óptica

se le establecerá según su grupo su correspondiente carga, esta variable es la llamada **lista_resultado.** En la función de calculo de la transmisión de entrada en cada unidad óptica trabajaremos con su correspondiente valor en cada unidad óptica. También debemos de tener en cuenta el definir los valores en el script del valor de carga de cada grupo.

El establecer los grupos lo hago respecto al múltiplo de la longitud de la lista de valores de las cargas que queramos dar. Si queremos dar 3 tipos de cargas a 16 onts se organizarán las onts de la siguiente manera:

- **1º grupo:** ONT 1, ONT 4, ONT 7, ONT 10, ONT 13 y ONT 16
- **2º grupo:** ONT 2, ONT 5, ONT 8, ONT 11 y ONT 14
- **3º grupo:** ONT 3, ONT 6, ONT 9, ONT 12 y ONT 15

En el siguiente ejemplo tendré en cuenta las gráficas de las **unidades ópticas 4,5 y 6** las cuales se corresponden con los **grupos 1,2 y 3**. Nos fijaremos ahora más en la linea azul por como se comporta el tráfico de entrada en cada grupo. La primera gráfica la cual se corresponde con la ONT 4 del grupo 1 de 900 Mbps de media podemos ver que los valores del tráfico de entrada(linea azul) son bastante altos.

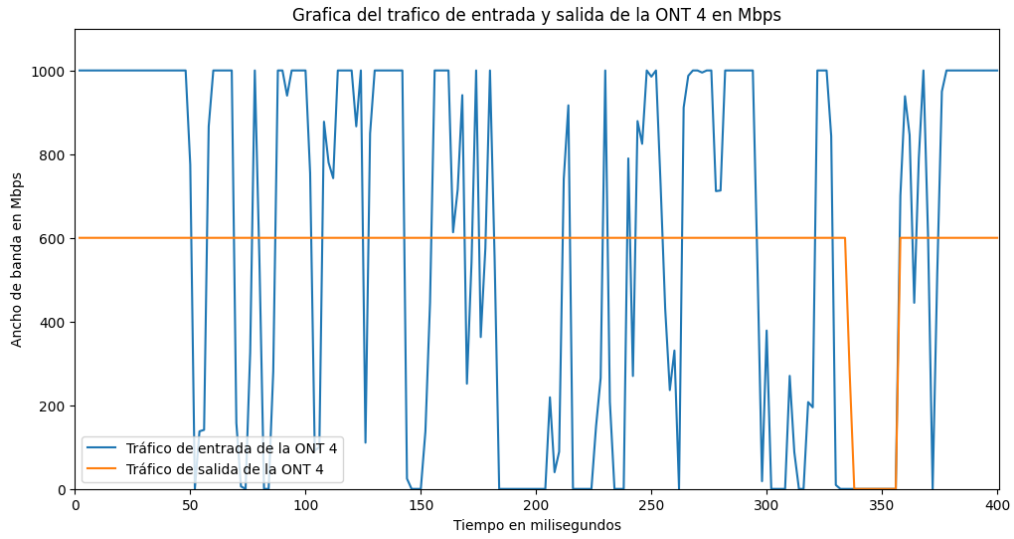

Figura 5.15: Escenario 2 Tráfico de valores de entrada y salida Ont 4

La segunda gráfica la cual se corresponde con la ONT 5 del grupo 2 de 800 Mbps de media podemos ver que los valores del tráfico de entrada (linea azul) son altos pero con más picos que el anterior.

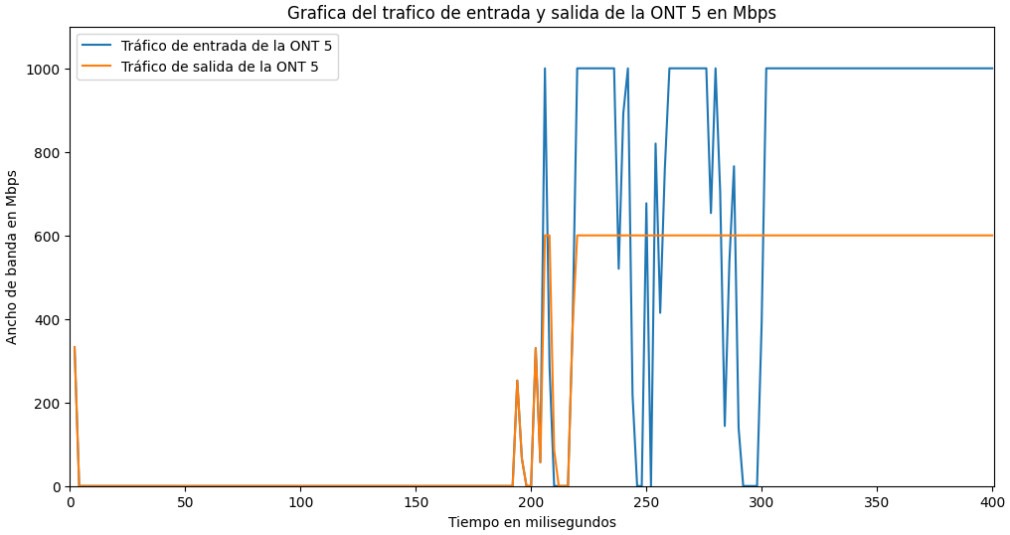

Figura 5.16: Escenario 2 Tráfico de valores de entrada y salida Ont 5

La tercera gráfica la cual se corresponde con la ONT 6 del grupo 3 de 700 Mbps de media podemos ver que los valores del tráfico de entrada(linea azul) son altos pero con más picos que el anterior.

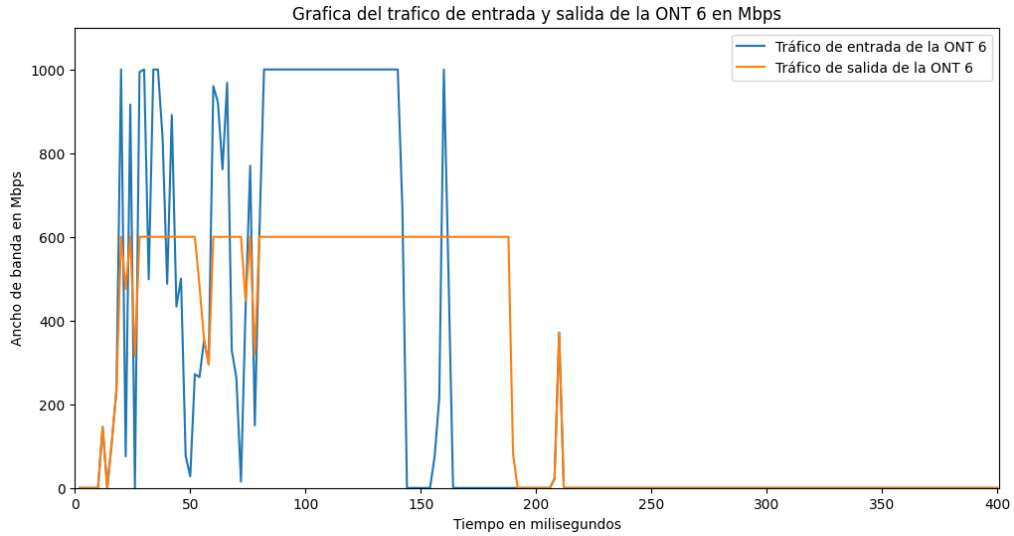

Figura 5.17: Escenario 2 Tráfico de valores de entrada y salida Ont 6

Escenario 3

El tercer escenario fue el más complicado de implementar debido a la lógica detrás de este y como realmente debía de retransmitir y debido a que los resultados no se pueden ver con tanta claridad los cambios realizados. En este escenario definiremos las siguientes condiciones:

- Todas las unidades ópticas transmiten la misma carga media de **900 Mbps.** Este valor se establece por los datos de **ON=1.4** y de **OFF=1.2.**
- La velocidad de transmisión máxima de la ONT es de **10 Gbps.**
- El ancho de banda garantizado máximo para cada unidad óptica debe de ser de **600 Mbps** al inicio del proyecto, pero después de un número de ciclos que **cambie a 400 Mbps.**
- El número de unidades ópticas es de 4 para que se pueda apreciar de una mejor manera el cambio realizado.
- El número de ciclos se puede elegir, yo lo he puesto a **200 ciclos.**

La cuestión de este escenario es la de que en un determinado momento, podamos variar el comportamiento de nuestro código pudiendo cambiar el

ancho de banda garantizado con el que iniciamos a otro y que el programa de machine learning se adapte a estos cambios.

Para ello definiré un entorno donde se deba cumplir lo siguiente: En el ciclo 100 o milisegundo 200, el programa debe de pasar de tener el ancho de banda garantizado de 600 Mbps a 400 Mbps.

Para ello definiremos una variable global el cual deberá de contar cada ciclo que llevamos para cambiar el comportamiento del programa cuando lleguemos al ciclo que pedimos. En nuestro caso debemos cambiar el tráfico garantizado que pedimos y la carga máxima de bits que este puede transmitir.

En la siguiente gráfica podemos ver con claridad que cuando llega al milisegundo 200(100 ciclos) pasamos de transmitir la salida(linea naranja) de su máximo de 600 Mbps a su nuevo máximo de 400 Mbps.

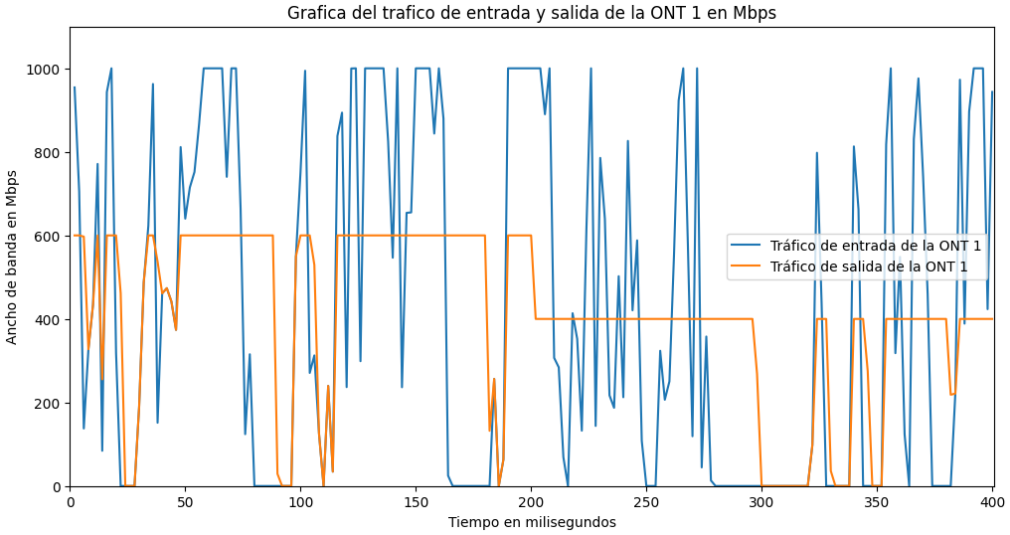

Figura 5.18: Escenario 3 Tráfico de entrada y salida

5.6. Refactorización de los escenarios

La **refactorización** es un proceso crucial en el desarrollo de software que implica **modificar el código para mejorar su estructura,** legibilidad y eficiencia sin alterar su comportamiento funcional. Este proceso es vital para mantener el código fácil de entender y de mantener a lo largo del tiempo.

Mediante el uso de SonarCloud, identificamos varios problemas clave en nuestro código, incluyendo:

- Nomenclatura Inconsistente: Variables y parámetros que no seguían las convenciones de nomenclatura estándar de Python.

- Uso de Funciones Obsoletas: Dependencia en métodos legados de generación de números aleatorios.

- Problemas de Seguridad: Posibles vulnerabilidades relacionadas con la gestión incorrecta de la entrada de datos.

Para ello haremos un primer análisis para ver los defectos de nuestros códigos.

Figura 5.19: Evolución de las Issues antes de la refactorización

Como podemos ver es un código bastante defectuoso, pero gracias a este análisis con la herramienta de SonarCoud podemos ver los defectos y corregirlos.

Figura 5.20: Evolución de las Issues después de la refactorización

La refactorización tuvo un impacto significativo en la calidad general del código. SonarCloud mostró una mejora en la puntuación de calidad, pasando de 52 a 2. Las métricas de legibilidad y mantenimiento mejoraron notablemente, lo que **facilitará la futura extensión y mantenimiento del proyecto.** Además, el código es ahora más seguro y robusto, reduciendo la probabilidad de errores y fallos.

6. Trabajos relacionados

José María Robledo Sáez, graduado en Ingeniería de Telecomunicaciones por la Universidad Autónoma de Madrid, presentó en 2024 su proyecto de fin de carrera titulado "Implementación de un Simulador de Redes de Acceso Ópticas Pasivas en OMNeT++". Este proyecto se enfocó en el desarrollo de un simulador de redes de acceso ópticas pasivas, también conocidas como redes PON (Passive Optical Networks), dentro del entorno de simulación OMNeT++.[15]

El principal objetivo del proyecto fue la **creación de un simulador que fuera lo más genérico y flexible posible,** capaz de simular diversas infraestructuras de redes PON bajo diferentes condiciones y escenarios. Además, se buscaba implementar diferentes estrategias y algoritmos de control de recursos para comparar los resultados obtenidos con los valores teóricos y otros resultados obtenidos en plataformas de simulación similares, como OPNET Modeler.

El proyecto culminó en el diseño y implementación de un simulador que **integra componentes y módulos detallados para el análisis y simulación de diversas estrategias de red.** Estos incluyeron desde la programación de las unidades de red óptica (ONUs) y las unidades de línea óptica (OLT), hasta la gestión del tráfico y la asignación dinámica de ancho de banda. La red PON simulada fue diseñada bajo el estándar Ethernet (EPON), lo que incluyó la gestión de tráfico en un canal bidireccional controlado por un mecanismo de acceso múltiple por división en el tiempo (TDMA).

Esta implementación permite no sólo evaluar la eficacia de diferentes configuraciones y estrategias de red sino también **ajustar y optimizar**

las operaciones de redes PON, contribuyendo así a la investigación y desarrollo en el campo de las telecomunicaciones y redes ópticas pasivas.

7. Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Este trabajo es una base bastante estable para futuros trabajos que se realicen entre estas ventajas que ofrece el código son las siguientes:

- **Funcional:** Todas las partes funcionan por si solas.
- **Base bastante funcional** de la cual no hay códigos en internet para basarse en ellos.
- **Modular:** El código esta organizado en métodos no muy extensos y llamadas a estos para una fácil comprensión del programador.
- **Diversidad de escenarios** y de ejemplos para que en un futuro no haya tantos problemas de asociación de conceptos y de compatibilidades de datos.
- **Comentarios:** El código esta comentado para la comprensión de cada parte.

A pesar de ello no me ha dado tiempo a implementar todos los escenarios y toda la complejidad que se puede introducir en este mundo tan amplio.

Por ello ofrezco una serie de posibles mejoras futuras para la mejora del código.

7.1. Ejecución paralela

El código esta desarrollado con **un solo entorno** debido a los datos que obtengo, esto con una mayor observación de los datos y una mejor optimización de los entornos se podrían ejecutar varios entornos a la vez para que el programa pueda aprender mucho más rápido con las mismas iteraciones.

7.2. Ejecución de varios episodios

El código esta desarrollado para la **ejecución de un solo episodio** para la salida de los valores ya predichos. Pero también estos valores predichos en vez de solo ser uno podrían ser varios episodios y ver que episodios han predicho mejores que otros para una mayor optimización de las cargas de datos.

7.3. Creación de diferentes SLAs

Otra de las posibles implementaciones que se podría hacer es la de **implementar y gestionar diferentes SLAs** en las unidades de red óptica de manera que cada usuario disponga de un servicio con características especificas de ancho de banda y calidad de servicio contratados.

7.4. Crear diferentes modelos de trafico

Se podría implementar **diversos tipos de trafico de datos** en las unidades ópticas para representar variados servicios de usuario, implementando diferentes algoritmos de gestión de colas en las ONUs que puedan manejar de manera eficiente múltiples tipos de trafico.

7.5. Implementación del agente en el simulador XGSPON

La universidad de Valladolid tiene en su acceso el simulador XGSPON de redes ópticas con lenguaje de Pyhton. Una de los objetivos finales del trabajo es la **implementación de este código de DRL en redes ópticas en el simulador para una mejora sustancial de la optimización de las retransmisiones de redes PON.**

El programa esta orientado a la posible implementación en este simulador, para ello solo necesitaríamos poner el valor que necesitemos en las siguientes variables:

- Según el escenario que deseemos, si es un problema simple usaríamos el escenario 1, si es un escenario donde tenemos diferentes grupos con sus correspondientes cargas de ON usaríamos el escenario 2 y si queremos que el programa varíe su ancho de banda garantizado en medio del programa usaríamos el escenario 3.

- Para variar el número de ONTs que tenemos modificaríamos la variable de num_ont, poniendo el valor que más deseemos.

- Si queremos variar el valor de la velocidad máxima de la red deberíamos de modificar la variable v_max_olt, debemos de poner el valor en bps, es decir si queremos poner 10Gbps como velocidad máxima de transmisión deberemos de poner 10*10e9.

- Para variar el valor de la velocidad garantizada para cada ont, deberemos de modificar el valor de vt_contratada, este valor también esta establecido en bps, es decir que si queremos poner 600 Mbps deberemos de poner el valor de 6000000000.

- El número de ciclos los establecemos por teclado, estableciendo los ciclos que queremos representar.

- Si queremos modificar el número de ciclos de entrenamiento modificaríamos la variable de total_timesteps, establecida inicialmente a 1000 iteraciones.

- En el escenario 2, para establecer los valores de la carga deberemos de modificar la variable de valores_ON para dictaminar los grupos y los valores de cada grupo.

El resto del programa está configurado para que se ejecute con las variables dictaminadas, es decir, que solo modificando las explicadas el funcionamiento del algoritmo variará según estas, sin depender de cambios auxiliares cuando queramos probar con otros valores diferentes, de por ejemplo aumentar el número de ONT o cambiar el ancho de banda garantizado.

7.6. Limitar el tamaño de las colas

Se podría estudiar el **establecer límites máximos en las colas de las ONUs** y realizar simulaciones para observar cómo afectan estos limites al trafico bajo con diferentes condiciones de carga.

7.7. Ajustar el tamaño del ciclo

Este código esta definido a 2 milisegundos por cada ciclo, pero otra mejora que se podría **establecer es que se ajusten dinámicamente el tamaño de ciclo de la transmisión en las ONUs** basándose en el trafico en tiempo real.

Bibliografía

[1] Aprendizaje por Refuerzo. https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/.

[2] Aprendizaje por Refuerzo. https://www.ceupe.com/blog/aprendizaje-por-refuerzo.html.

[3] Metodología Scrum: Qué es y cómo funciona. https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html.

[4] Qué es una Red Óptica Pasiva (PON). https://www.viavisolutions.com/es-mx/que-es-una-red-optica-pasiva-pon.

[5] What is PON? https://www.juniper.net/mx/es/research-topics/what-is-pon.html.

[6] Admin. GPON - How it works, its components, benefits & drawbacks. https://stl.tech/blog/everything-about-gpon-gigabit-passive-optical-network/, 2023. Accessed: 16 June 2024.

[7] Amazon Web Services. What is Reinforcement Learning? https://aws.amazon.com/es/what-is/reinforcement-learning/.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Available at http://www.deeplearningbook.org.

[9] Gymnasium Documentation. https://gymnasium.farama.org/, 2024. Accessed: 16 June 2024.

[10] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more*. Packt Publishing Ltd., Birmingham, UK, 2020.

[11] Machine Learning Expedition. An Introduction to Proximal Policy Optimization (PPO) in Reinforcement Learning. https://www.machinelearningexpedition.com/ppo-proximal-policy-optimization/, 2024. Accessed: 16 June 2024.

[12] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. Accessed: 16 June 2024.

[13] John Schulman et al. Proximal Policy Optimization Algorithms. 2017. Available online.

[14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 2020.

[15] José María Robledo Sáez. Memoria del Proyecto de Fin de Carrera: Desarrollo de Simulador de Redes de Acceso Ópticas Pasivas, 2012.

[16] Wikipedia contributors. Pareto distribution — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Pareto_distribution, 2024. [Online; accessed 5-July-2024].

[17] Noemí Merayo Álvarez. Redes de Acceso Ópticas Pasivas (PON). Asignatura de Sistemas de Comunicaciones Ópticas, Universidad de Valladolid.