UNIVERSIDAD ADOLFO IBAÑEZ

TESIS DE MAGÍSTER

Biblioteca de simulación en C++ para codificar algoritmos de asignación de recursos en redes ópticas elásticas multibanda

---

Autor:
David Lara

Profesor guía:
Danilo Bórquez Paredes

Tesis realizada acorde a los requerimientos para el grado de
Magíster en Ciencias de la Ingeniería, mención Tecnologías de la Información

de la

Facultad de Ingeniería y Ciencias

15 de diciembre de 2023

UAI | FACULTAD DE INGENIERÍA Y CIENCIAS

UNIVERSIDAD ADOLFO IBAÑEZ

Resumen

Facultad de Ingeniería y Ciencias

Magíster en Ciencias de la Ingeniería, mención Tecnologías de la Información

Biblioteca de simulación en C++ para codificar algoritmos de asignación de recursos en redes ópticas elásticas multibanda

por David Lara

En los últimos años, se ha presentado con mucha fuerza una tecnología para aumentar la capacidad de la fibra: La multiplexación por división de banda (BDM) o multibanda. En multibanda, el objetivo es utilizar de manera eficiente el espectro no utilizado en cada fibra. De este modo, las conexiones realizadas ya no se harían sólo en la banda C- (Banda utilizada actualmente), sino que se ampliaría el uso de las bandas O-, E-, S- y L-. La ventaja que tiene multibanda es que no necesita la instalación de nuevos cables. Por otro lado, las nuevas bandas tienen una influencia más significativa de efectos no lineales en la señal, lo que afecta a aspectos críticos de la comunicación.

El desafío de utilizar multibanda se relaciona con diversos problemas que han surgido en el ámbito de las redes ópticas. Uno de ellos es el de asignación de rutas y espectro (RSA o *Routing and Spectrum Assignment*). Donde se busca encontrar las rutas y el espectro para cada demanda de forma que se eviten las interferencias entre las señales ópticas y se optimice algún criterio, como la probabilidad de bloqueo o el uso del espectro. Un problema más complejo es el de Enrutamiento, Nivel de Modulación y Asignación de Espectro (RMLSA *Routing, Modulation Level and Spectrum Assignment*). Además de considerar las rutas y el espectro, tiene en cuenta el alcance óptico y los formatos de modulación de las señales, que determinan la calidad de transmisión y la capacidad de cada canal óptico.

Aunque en la actualidad existen herramientas para trabajar los problemas previamente descritos en ambientes de redes ópticas elásticas, no son lo suficientemente flexibles como para trabajar sólo la parte algorítmica, obligando al investigador a saber programar simuladores, o bien paradigmas específicos de programación. A su vez, estas herramientas aun no son adecuadas al contexto multibanda. Es por esto que se busca desarrollar una nueva herramienta de simulación para facilitar la validación de algoritmos de asignación de recursos en redes ópticas elásticas con multibanda.

Palabras clave: Redes ópticas, Multibanda, Biblioteca, Simulación Orientada a Eventos.

Índice general

Capítulo 1

Introducción

Los dispositivos de telecomunicación permiten intercambiar información desde un emisor a un receptor a través de largas distancias. En los comienzos, cuando los mensajes eran simples y las distancias cortas, las personas utilizaban señales de fuego o humo. Con el aumento de las distancias y la complejidad de los mensajes, se fueron inventando y utilizando distintos dispositivos, como por ejemplo, las cartas o el telégrafo, el cual transformó la comunicación, la cultura y la política en el siglo XIX [1]. Los dispositivos fueron aumentando cada vez más su complejidad hasta llegar a la telefonía cableada, que permite transmitir voz entre dos lugares fijos distantes entre sí. Por otro lado, esta la telefonía celular que permite una comunicación móvil entre dos dispositivos (pero siempre dentro de la red de cobertura) [2]. Luego llegan las redes de computadores, que permitían utilizar servicios como el *Email* para mandar mensajes electrónicos a largas distancias [3].

En el año 1960 existían alrededor de 10.000 computadoras en todo el mundo, pero su función estaba enfocada principalmente en el procesamiento de información y no en ser un canal para permitir las telecomunicaciones [4]. Esto fue cambiando con la llegada de los protocolos de transmisión. Uno de estos fue el protocolo de control de transmisión o *transmission control protocol* (TCP), que permitía conexiones a través de la red. TCP es un conjunto de normas sobre cómo los computadores se comunican, y permite interconectar redes y computadores [5]. A su vez, IP es un protocolo sin conexión que se encarga del direccionamiento y enrutamiento de paquetes de información. El conjunto TCP/IP permite, por ejemplo, que los usuarios puedan mandar *Emails* o transferir archivos electrónicamente sin la necesidad de preocuparse por las diferencias que puedan tener las redes físicamente. Por lo tanto, TCP/IP se convirtió en el estándar para la conexión entre computadores desde 1983 [6].

Con estas herramientas ya se podía tener acceso a un internet que funcionaba principalmente dentro de una red Local o *Local Area Network* (LAN), debido a las limitaciones físicas que implicaba tener cables eléctricos para la transmisión de la información a largas distancias [7]. El tipo de cable más común que se utilizaba (y se sigue utilizando hasta el día de hoy) es el par trenzado. Este cable permite una velocidad de transmisión máxima de 10 *Gigabit* por segundo (*Gbps*), y tiene un alcance máximo de 100 *m* [8]. Otro cable comúnmente utilizado para este fin es el cable coaxial, que permite una velocidad de transmisión máxima de 1 *Gbps* y tiene un alcance de unos 100 *m* sin una pérdida de señal significativa [9]. Dada las limitaciones físicas de estos cables, no se pueden utilizar de manera eficiente para permitir conexiones a nivel mundial, esto porque tendría que pasar corriente a través del mar y colocar una cantidad considerable de repetidores [10]. Un repetidor es un dispositivo utilizado para aumentar el alcance de la red recibiendo una señal a través de un cable

por un extremo, la cual es reenviada por el otro extremo. En este caso, el repetidor sirve para llegar a una distancia mayor con los cables de par trenzado o coaxial que tienen un alcance limitado [11]. Por otro lado, aunque se lograra realizar una conexión mundial con estos cables, existe el problema de que la velocidad de transmisión sería insuficiente, dado que una sola gran empresa privada podría necesitar hasta 10 *Gbps* para sus operaciones [12]. En consecuencia se hace necesario otro tipo de cable de transmisión para poder soportar altos volúmenes de información enviados a grandes distancias.

Un informe de *Strategy Analytics* predice que la demanda de ancho de banda crecerá modestamente durante la primera mitad de la década, pero a partir de 2026, las necesidades de ancho de banda se dispararán, creciendo aproximadamente 300 *Mbps* por año [13]. Mientras que el informe anual de Internet de Cisco (2018-2023) proyecta que la velocidad promedio de banda ancha global se duplicará de 45,9 Mbps a 110,4 Mbps de 2018 a 2023. A su vez la cantidad de dispositivos inteligentes aumentará de 18,4 a 29,3 mil millones entre el año 2018 y 2023 [14]. Además, la cantidad de dispositivos domésticos conectados a internet va en aumento [15], generando como consecuencia un mayor consumo del ancho de banda disponible [16].

En otro estudio, Cisco afirmó que en América Latina el tráfico de IP se multiplicaría por tres desde 2017 al 2022 [17], a una tasa de crecimiento anual compuesta del 21 %. También se mencionó que el tráfico de Internet en hora punta aumentaría 3,9 veces desde el 2017 al 2022, a una tasa de crecimiento anual compuesta del 31 % en América latina, por lo que, se puede ver una clara tendencia al alza en el consumo de internet y el aumento en el tráfico de datos.

Para enfrentar las limitaciones de los cables antes mencionados se utiliza fibra óptica en el núcleo de la red, logrando así cubrir las grandes distancias presentes en, por ejemplo, los enlaces de internet trans-oceánicos [18]. Los cables de fibra óptica permiten transportar información en forma de pulso de luz llegando a una velocidad de transferencia teórica mayor a 50.000 Gbps [19][1]. Otra característica importante de la fibra óptica es que tiene un alcance de varios cientos de kilómetros [20], por lo que, la utilización de energía eléctrica (utilizada sólo en los repetidores de señal, dado que la fibra transporta pulsos de luz) también es necesaria cada algunos cientos de kilómetros. Es por esto que se utilizaron grandes cables de fibra óptica que atravesaban el mar para conexiones inter-continentales [21].

La fibra está hecha de un material altamente transparente que está especialmente diseñado para aprovechar la propiedad de reflexión de la luz [19], de forma que, se pueda mandar una señal utilizando pulsos de luz desde un extremo al otro. La potencia de esta señal puede tener pérdidas en distancias grandes [22] por lo que, si se quiere hacer una conexión de un alto alcance es necesario tener puntos donde se devuelve la potencia a la luz. Esto ocurre en la unión submarina que conecta a los distintos países [18], donde se recupera la potencia de luz través de un amplificador óptico que convierte la señal óptica en una señal eléctrica y luego la vuelve a convertir en señal óptica, pero amplificada [23].

La fibra óptica puede dividirse en dos categorías: la monomodo o *Single Mode Optical Fiber* (SMF) y la multimodo o *Multi Mode Fiber* (MMF) [24]. La principal diferencia es que la primera tiene un diámetro pequeño, cercano a $9\mu m$, mientras que la segunda tiene un diámetro mayor de alrededor de $62,5\mu m$. Estas diferencias afectan las

[1] A pesar de todos los avances que se han tenido en el ámbito de las telecomunicaciones, la velocidad de transferencia teórica de la fibra óptica aún se ve lejana.

características funcionales de la fibra: en SMF existe sólo un tipo de transmisión y longitud de onda, lo que se puede traducir en que la luz interactúa menos con las paredes de la fibra, reduciendo así la pérdida de intensidad de la luz. En consecuencia, este tipo de fibra puede llegar a una mayor distancia de manera efectiva. Por otro lado, en MMF la luz tiene una mayor interacción con las paredes de la fibra (mayor que en SMF), lo que produce mayores pérdidas en la potencia de la luz, reduciendo así su distancia efectiva. Por lo tanto, la diferencia fundamental es que MMF es utilizada comúnmente para redes LAN, es decir, redes de dispositivos que comparten una línea de comunicaciones común donde existen distancias cortas, dado que los sistemas de MMF son más económicos, mientras que SMF se utiliza para redes globales o *Wide Area Network* (WAN) que son distancias más largas.

Dentro de la fibra la luz puede propagarse con diferentes longitudes de onda [25], al igual que el sonido se puede propagar con distintas frecuencias. En el caso de las redes ópticas se utiliza la unidad de medida nanómetro (nm), para las distintas longitudes de onda. Estas longitudes de onda se dividen en unidades llamadas *Frecuency Slots Unit* (FSU) que son la fracción más pequeña de espectro que se puede asignar en una conexión. Igualmente pueden usarse las frecuencias, haciendo la respectiva conversión mostrada en la Ecuación 1.1.

$$c = \lambda f \tag{1.1}$$

Donde c corresponde a la velocidad de la luz, λ a la longitud de onda usada y f a la frecuencia de la respectiva longitud de onda utilizada. Considerando que la velocidad de la luz utilizada en la fibra óptica para EONs es $2{,}99792458 \cdot 10^8 \frac{m}{s}$ [26].

Los distintos rangos de longitudes de onda que se pueden utilizar en SMF para transmitir información son llamados bandas. Existen 5 bandas principales utilizadas en las comunicaciones ópticas [27]: Banda O, Banda E, Banda S, Banda C y Banda L. La Banda Original (Banda O) va desde los $1260nm$ hasta los $1360nm$, siendo la primera banda utilizada en la comunicación óptica. La banda extendida (*Extended*, Banda E) va desde los $1360nm$ hasta los $1460nm$ y es una extensión de la Banda O. La banda Convencional (*Conventional*, Banda C) va desde los $1530nm$ hasta los $1565nm$, esta es la banda que se utiliza convencionalmente en los sistemas de fibra óptica debido a que tiene una pérdida de información baja a largas distancias. La banda de Longitud de onda corta (*Short wavelength*, Banda S) va desde los $1460nm$ hasta los $1530nm$, esta banda esta compuesta por las longitudes de onda menores a la banda C y mayores a la banda E. La banda de Longitud de onda larga (*Long wavelength*, Banda L) va desde los $1625nm$ hasta los $1675nm$. En cuanto al resto de longitudes de onda, bajo los $1260nm$ SMF se comienza a comportar como si fuera MMF y pasando los $1625nm$ sube exponencialmente la atenuación, que es la pérdida de potencia o intensidad de la señal óptica que se produce al propagarse por la fibra óptica. La Tabla 1.1 contiene un resumen del rango de cada banda.

Banda	Nombre español	Nombre inglés	Rango de longitud de onda (nm)
O	Original	Original	1260 - 1360
E	Extendida	Extended	1360 - 1460
S	Longitud de onda corta	Short wavelength	1460 - 1530
C	Convencional	Conventional	1530 - 1565
L	Longitud de onda larga	Long wavelength	1565 - 1625

TABLA 1.1: Bandas utilizadas en redes ópticas [27].

En los últimos años, se han presentado con mucha fuerza dos tecnologías para aumentar la capacidad de la fibra: La multiplexación por división de espacio o *Space-division multiplexing* (SDM) y la transmisión óptica multibanda (MB de *Multi-Band optical transmission*). La primera consiste en sustituir las actuales SMF por otras nuevas que admiten la transmisión de información a través de múltiples pares de SMF. En multibanda, lo que se intenta aumentar es el espectro utilizado en cada fibra. De este modo, las conexiones realizadas ya no se harían sólo en la banda C, sino que se ampliaría el uso de las bandas O, E, S y L. De este modo, el uso del espectro pasaría de estar en el rango de 1530nm - 1565nm a estar en el rango de 1260nm - 1625nm. La ventaja que tiene multibanda sobre SDM es que no necesita la instalación de nuevos cables. Sin embargo, las nuevas bandas tienen una influencia más significativa de efectos no lineales en la señal [28]. Estos efectos son una clasificación dentro de la transmisión de señales y dependen de la intensidad de la señal. Esta influencia afecta a aspectos críticos de la comunicación como el alcance óptico de la conexión [29]. Por lo que, la señal en un sistema multibanda puede alcanzar una distancia menor de manera efectiva. También se pueden aumentar las demandas que son denegadas por la red [30], lo que se refiere a las solicitudes de conexión o transmisión de datos no pueden ser satisfechas o completadas exitosamente.

Otros efectos que perjudican en la transmisión de señales son la dispersión cromática y la atenuación [31]. La dispersión cromática causa un ensanchamiento del pulso de luz, lo que implica que el pulso aumente su duración temporal a medida que se propaga por la fibra. Esta variación se mide en $ps/(nm * Km)$ y define cuantos picosegundos se ensanchará un pulso de 1nm por cada kilometro [32]. La atenuación es la pérdida de la potencia de la señal óptica. Esta pérdida se mide en dB por Km, lo que es decibeles por kilómetros. El decibel es una unidad logarítmica que se utiliza para medir la relación entre dos cantidades, en este caso, la atenuación de la señal óptica en la transmisión de fibra óptica.

En la Figura 1.1, se puede ver un gráfico donde se muestra la atenuación y la dispersión cromática dependiendo de las distintas longitudes de onda, junto con las divisiones de las distintas bandas. El eje *Wavelenght*, representa la longitud de onda en nanómetros. Las longitudes de onda son diferentes para las distintas bandas (*O-band*, *E-band*, *S-band*, *C-band* y *L-band*) que se muestran en el gráfico. Por su parte, el eje *Loss* es la atenuación en decibelios. La atenuación se refiere a la pérdida de potencia de una señal en la fibra óptica, que se representa con la línea inferior en el gráfico. Por último, el eje *Chromatic Dispersion* es la dispersión cromática en picosegundos por nanómetro, que se representa con la línea superior en el gráfico. La dispersión cromática es una medida de cómo diferentes longitudes de onda de luz se propagan a diferentes velocidades en la fibra óptica.

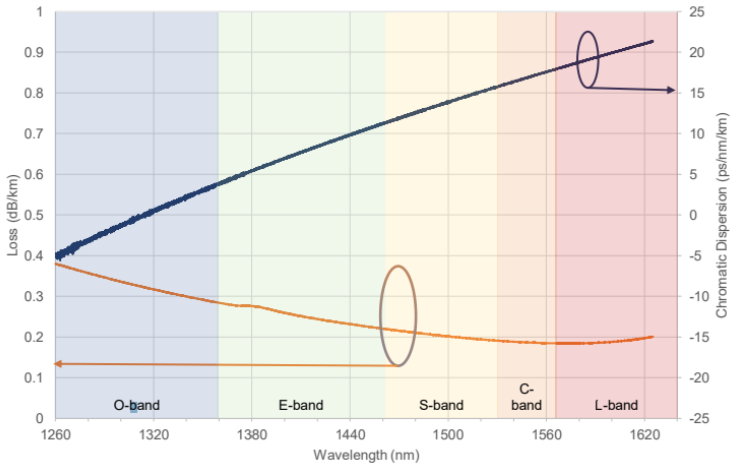


FIGURA 1.1: Atenuación y dispersión de un pulso de luz en función de la longitud de onda utilizada [27].

Se puede ver que las bandas que tienen una menor atenuación son la C y la L. En la banda C la atenuación esta por debajo de los 0,2 dB/Km y la dispersión cromática está entre los 15 y 20 $ps/(nm * Km)$. En esta banda se presenta una baja atenuación y dispersión cromática, siendo la razón de su utilización en los sistemas de fibra óptica. En la banda L la atenuación se encuentra por debajo de los 0,2 dB/Km hasta los 1600 nm de longitud de onda aproximadamente, después de esto ya se encuentra en los 0,2 dB/Km de atenuación, mientras que la dispersión cromática fluctúa los 20 $ps/(nm * Km)$. En esta banda se ve una atenuación un poco mayor a la banda C, pero con una dispersión cromática considerablemente mayor. En cuanto al resto de las bandas cuentan con una atenuación entre los 0,2 y 0,4 dB/Km lo que es considerablemente mayor a las bandas C y L. En cuanto a la dispersión cromática poseen una ventaja dado que varía entre -5 y 15 $ps/(nm * Km)$, por lo que, se presenta el desafío de lidiar con otras atenuaciones y dispersiones cromáticas si se quiere utilizar multibanda en redes ópticas.

Capítulo 2

Definición del problema

Comprender la problemática de la implementación de multibanda en redes ópticas requiere previamente familiarizarse con el concepto de red óptica. Una red óptica es un conjunto de nodos conectados por un conjunto de enlaces ópticos. Matemáticamente esta definida como un grafo dirigido $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, donde $\mathcal{N}$ representa al conjunto de nodos y $\mathcal{L}$ representa el número de enlaces. Cada nodo puede representar computadores, servidores, potenciadores ópticos, etc. Mientras que los enlaces representan cables de fibra óptica SMF o MMF.

La Figura 2.1 muestra una representación de una red óptica que está compuesta por 7 nodos y 9 enlaces. Cada nodo es representado por un icono de un router, e identificado por una letra (A, B, C, D, E, F y G). En cuanto a los enlaces, estos están representados con una línea negra que es la unión entre 2 nodos. Las rutas se definen como un camino compuesto por un conjunto de enlaces para llegar de un nodo a otro. Se puede notar que hay nodos que no están conectados directamente, como el A y el E, pero se pueden comunicar gracias a las rutas que se crean en las redes. Estas rutas también son llamadas *lightpath*, que es un conjunto de enlaces que forman una vía de comunicación entre dos nodos. En el ejemplo, una posible forma en la que podrían comunicarse estos nodos es por la ruta A-C-E que sería la más directa. De todos modos, existen más rutas posibles para realizar la misma acción, como por ejemplo las rutas A-C-D-E, o la A-B-C-E, teniendo múltiples posibilidades para una misma conexión.

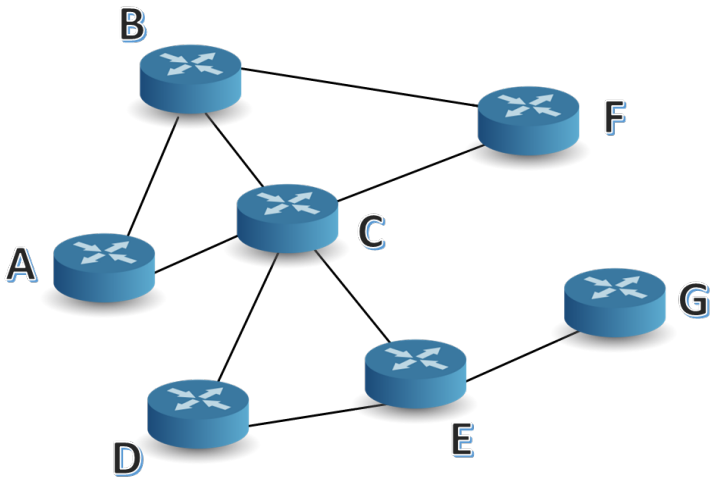


FIGURA 2.1: Representación de red óptica con nodos y enlaces.

Otro término importante dentro de una red, es el *Bit rate*. Este término se refiere a la tasa de *Bits*, es decir, el número de *Bits* que se transmiten por unidad de tiempo. El *Bit rate* es relevante para determinar la capacidad y eficiencia de una red óptica, dado que influye en la cantidad de datos que se transmite. Un *Bit rate* más alto permite transmitir más información en un período dado, sin embargo, puede requerir más ancho de banda al tener una mayor cantidad de datos en menos tiempo.

Si se quiere enviar información desde los nodos A y B de la Figura 2.1 al nodo E, se podrían usar los enlaces que los conectan al nodo C y utilizar la conexión C-E para llegar al nodo esperado. Sin embargo, esto podría llegar a superar la capacidad que tiene el enlace C-E para transportar información. Si bien esta es una red pequeña, pueden existir cientos de conexiones que funcionan cada segundo, por lo que, para poder hacer esta operación habría que utilizar distintas rutas. En este caso, para el nodo A se puede utilizar la conexión A-C-E como se ve en la Figura 2.2 con la ruta 2, mientras que para el nodo B se puede utilizar la conexión B-C-D-E que sería la ruta 1, de forma en que la carga de la información no recaiga toda en el enlace C-E. En redes complejas se utilizan sofisticados algoritmos para realizar estos cálculos y así asegurar que el transporte de información sea efectivo y no sobrecargar la red [33].

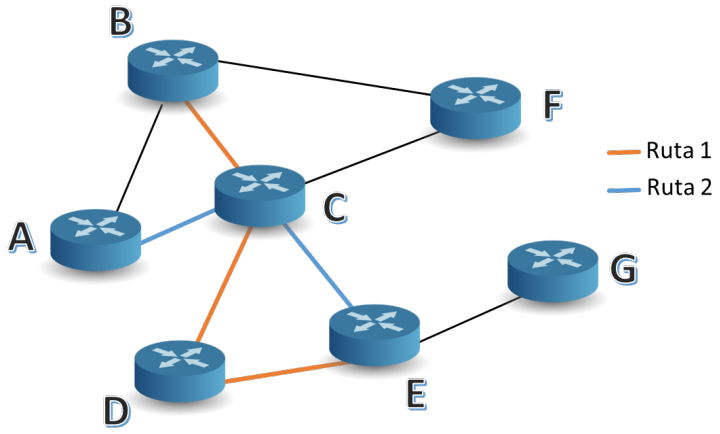


FIGURA 2.2: Representación de red óptica con rutas definidas.

En el ejemplo que se muestra en la Figura 2.2 se muestra una posible solución al problema de asignación de rutas y longitudes de onda o *routing and wavelength assignment* (RWA). En este problema, el objetivo es encontrar las rutas y la asignación de longitudes de onda de manera que una misma longitud de onda no puede ser utilizada por más de una conexión en un mismo enlace, minimizando (o maximizando) alguna métrica, como la probabilidad de bloqueo que corresponde a la tasa de conexiones rechazadas en una red con respecto al número total de conexiones intentadas. Por ejemplo, se puede ver en la Figura 2.3 que la conexión A-E usa la longitud de onda $\lambda 2$, mientras que la conexión B-E usa λ_1. De esta forma, para transmitir la información se tiene que hacer utilizando la longitud de onda respectiva de cada ruta.

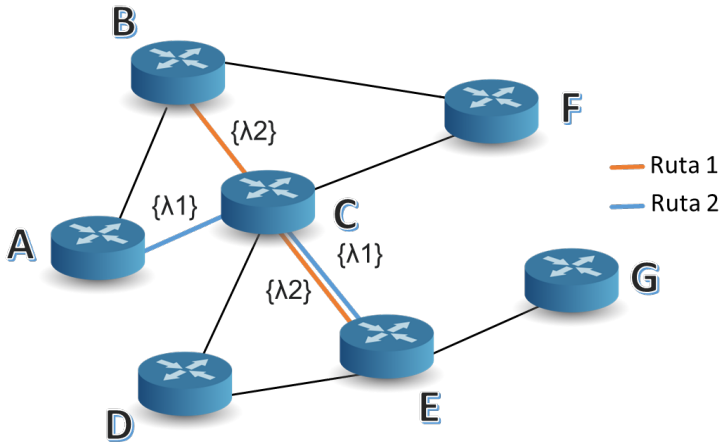


FIGURA 2.3: Representación de red óptica utilizando RWA.

Luego este problema evolucionó con la llegada de las redes ópticas elásticas o *Elastic Optical Network* (EON), cambiando el concepto "longitud de onda" por "espectro". De esta manera el problema fue nombrado como *Routing and Spectrum Assignment* (RSA). En una EON en vez de asignar una longitud de onda definida a un enlace, se asigna un espectro flexible. Este espectro se divide en FSU que son llamados *Slots*. En la Figura 2.4 se muestra la misma red de la Figura 2.2, pero como si fuera una EON. En este caso cada enlace dentro de las rutas tiene 7 *Slots* numerados del 0 al 6. Por esta cantidad de *Slots* se podría asignar el espectro desde el *Slot* 0 hasta el 4 para que fuera utilizado dentro de la red según la demanda de esta. También se podría utilizar un espectro sin límite al utilizar todo los *Slots* disponibles [34].

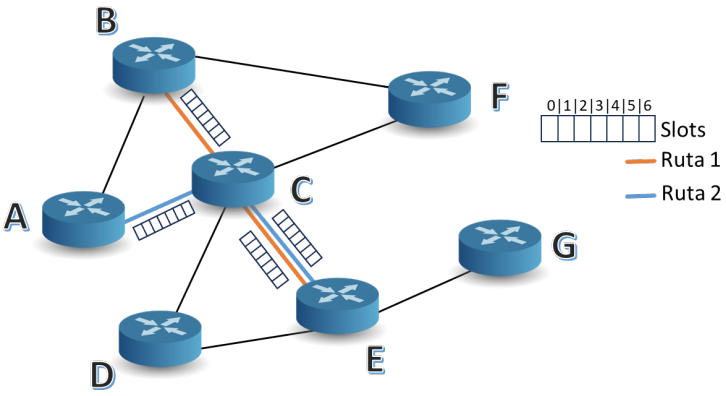


FIGURA 2.4: Esquema de red óptica EON.

Cuando un nodo en una EON necesita enviar datos a otro nodo, el proceso de asignación de rutas y espectro se convierte en un componente esencial de la operación. A diferencia de las redes ópticas tradicionales, donde la elección de una longitud de onda específica para una conexión era la norma, en una EON, se busca primero encontrar una ruta disponible entre los nodos de origen y destino. Una vez que se ha determinado la ruta, el siguiente paso es asignar un espectro adecuado para la transmisión de datos. Este espectro puede ser una parte del espectro total disponible en el enlace o incluso abarcar múltiples longitudes de onda contiguas, dependiendo de la demanda de ancho de banda de la conexión en ese momento. La asignación de

espectro se realiza de manera que no haya interferencia ni conflicto con otros canales o conexiones en el mismo enlace, lo que garantiza una transmisión sin interferencia. Una forma de ver esto es en la Figura 2.5. En este caso, se quiere enviar información equivalente a 2 *Slots* desde el nodo B hasta el E. Si se selecciona la ruta 1, después se seleccionaría la cantidad de *Slots* requeridos, en este caso, 2. Después, se seleccionarían cuáles *Slots* se utilizarían. Suponiendo que están disponibles el *Slot* 0 y el 1, se asumirá que se seleccionan estos (se ven marcados en verde en la Figura 2.5). De esta forma, se establecería una conexión y se enviarían los datos entre estos nodos.

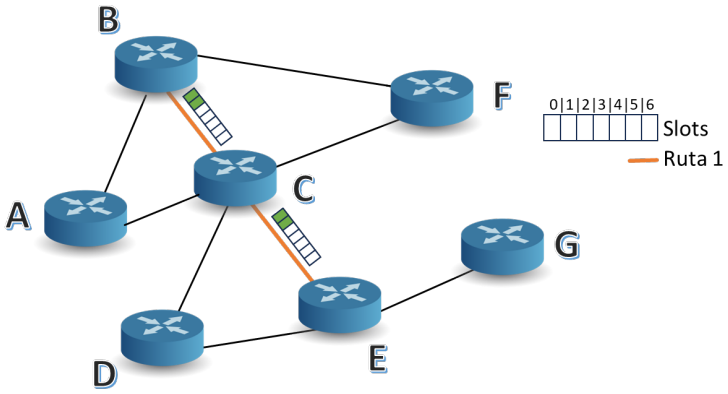


FIGURA 2.5: Ejemplo de transmisión de datos en red óptica EON.

La asignación que se ejemplificó en el párrafo anterior, tiene que cumplir con las restricciones de continuidad y contigüidad [35]. La restricción de continuidad exige que, para una solicitud dada, los mismos segmentos de *Slots* se asignen en todos los enlaces de la ruta, garantizando coherencia y fluidez en la transmisión de datos desde el nodo de origen hasta el destino. Por otro lado, la restricción de contigüidad implica que los *Slots* asignados deben ser adyacentes entre sí en el dominio de la frecuencia. Esta condición asegura una asignación compacta y eficiente del espectro, evitando brechas no utilizadas que podrían comprometer la capacidad de la red. En cuanto a las conexiones, se tiene que cumplir con que sean transparentes, es decir, que no modifican ni alteran los datos que se transmiten a través de estas.

La flexibilidad en la asignación de espectro en las EONs es particularmente valiosa en situaciones donde la demanda de ancho de banda puede variar con el tiempo. Por ejemplo, en momentos de alta demanda, se pueden asignar espectros más amplios para acomodar una mayor cantidad de datos, mientras que en momentos de menor actividad, se pueden liberar recursos al asignar espectros más estrechos o reducir la cantidad de longitudes de onda utilizadas. Esto permite una asignación eficiente de recursos y una adaptación continua a las condiciones cambiantes de la red [36].

Posteriormente, se empezó a trabajar con el problema de asignación de recursos en WAN, por lo que, los términos como alcance óptico y formatos de modulación se volvieron más relevantes. El alcance óptico es la distancia que puede recorrer el pulso de luz sin que se dañe la señal [37]. El formato de modulación se puede definir como el formato que se utilizará para mandar la información a través de la red, tal que no se vea afectada mayormente por factores como la atenuación o la dispersión cromática. Junto con estos términos aparecen conceptos como el enrutamiento y Asignación de Espectro. En consecuencia, el problema pasó a llamarse Enrutamiento, Nivel de Modulación y Asignación de Espectro o *Routing, Modulation Level and*

Spectrum Assignment (RMLSA). Los formatos de modulación son una parte importante en RMLSA. El formato de modulación se refiere al formato que se utilizará para mandar la información a través de la red, tal que no se vea afectada mayormente por factores como la atenuación o la dispersión cromática. En otras palabras, el formato de modulación es la forma en que se codifica la información para ser transmitida a través de la red.

Los formatos de modulación BPSK, QPSK y 8-QAM son algunos de los utilizados en redes ópticas [38]. Cada formato de modulación tiene sus propias ventajas y desventajas, y se utiliza en diferentes situaciones dependiendo de las necesidades de la red. El BPSK (Binary Phase Shift Keying) es un formato de modulación que utiliza dos fases para representar los datos, lo que lo hace adecuado para aplicaciones de baja velocidad. El QPSK (Quadrature Phase Shift Keying) es un formato de modulación que utiliza cuatro fases para representar los datos, lo que lo hace adecuado para aplicaciones de velocidad media. El 8-QAM (8-Quadrature Amplitude Modulation) es un formato de modulación que utiliza ocho estados de amplitud y fase para representar los datos, lo que lo hace adecuado para aplicaciones de alta velocidad. En general, los formatos de modulación se seleccionan en función de la tasa de bits, la distancia de transmisión y la calidad de la señal requerida.

Para representar los formatos de modulación de una forma gráfica se pueden utilizar constelaciones. La constelación se puede visualizar como un diagrama de puntos en el plano complejo, donde cada punto representa un símbolo en la constelación. La distancia entre los puntos en la constelación se conoce como la distancia de símbolo, y es una medida de la separación entre los símbolos en la constelación. Una constelación con una mayor distancia de símbolo puede transmitir más información por unidad de tiempo, pero también es más susceptible a errores de transmisión [39]. En la Tabla 2.1 se pueden ver las constelaciones de los formatos de modulación BPSK, QPSK y 8-QAM según lo que se habló en el párrafo anterior.

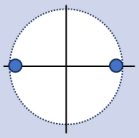
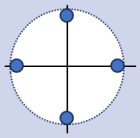
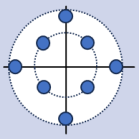
Formatos de modulación	BPSK	QPSK	8QAM
Constelación			

TABLA 2.1: Tabla de constelaciones de formatos de modulación.

Volviendo a la asignación de espectro en las EONs, hoy en día la evolución incorpora aspectos de multibanda. Esto lleva a expandir el espectro, por lo que se debe lidiar con pérdidas mayores a las que se tenían típicamente en la banda C, teniendo en consecuencia una menor distancia efectiva para transmitir información.

En el contexto de la evolución de las EONs y la complejidad asociada a la asignación de espectro, la gestión eficiente del tráfico en una red se convierte en un aspecto que hay que tener en consideración. El tráfico en una red corresponde a la información de las conexiones que se realizarán en ella, como por ejemplo qué pares de nodos serán conectados, cuándo lo harán y por cuánto tiempo [40]. Los distintos tráficos de red son otro punto importante que se puede trabajar en el problema RMLSA. En

el caso del tráfico estático, que es una cantidad fija de información, se trabaja definiendo antes de comenzar a operar la red el problema RMLSA. En cuanto al tráfico incremental, que es cuando la cantidad de tráfico va aumentando con el paso del tiempo, se van dejando fijas las rutas que se asignaron previamente. Esto quiere decir, que no se sabe el orden en que llegan las conexiones, pero que una vez asignadas no serán liberadas. En el caso de que exista un tráfico dinámico, se asignan lo recursos a medida de que las demandas van llegando, de manera que las rutas que son elegidas son las menos congestionadas [41]. En este último caso, no se sabe cuando las conexiones llegarán, ni cuando serán liberadas. La solución del problema RMLSA en estos tráficos son los algoritmos de asignación de recursos [42]. Dependiendo del tráfico en el cual se trabaja en la red es qué algoritmo usar. Por otro lado, está la capacidad de peticiones que puede recibir la red, los parámetros que se deben utilizar, entre varios otros factores. Por lo que, para realizar pruebas, en especial en presencia de tráfico dinámico, se utilizan simuladores de redes ópticas. Los simuladores se utilizan dado que presentan un bajo costo al no tener que comprar componentes, y se pueden probar distintos tipos de tráficos en una red. Además, se pueden probar y modificar distintas topologías sin hacer cambios físicos en un tiempo de ejecución considerable, entre otras ventajas.

El crecimiento exponencial de la demanda de ancho de banda y el aumento en el tráfico de datos plantean desafíos significativos para las redes ópticas actuales. Aunque la utilización de cables de fibra óptica ha permitido superar las limitaciones de los cables tradicionales, aún se requieren mejoras para satisfacer las necesidades de transmisión a nivel mundial. En este contexto, la transmisión óptica multibanda emerge como una oportunidad prometedora. Al aprovechar diferentes bandas de longitud de onda en la fibra óptica, se puede ampliar el espectro utilizado y aumentar la capacidad de transmisión de la red. Sin embargo, en el contexto de la asignación de recursos, la transmisión óptica multibanda presenta desafíos adicionales. Actualmente, no existen herramientas que permitan trabajar con algoritmos de asignación de recursos en ambientes multibanda para trabajar estos desafíos. Por lo tanto, es necesario desarrollar un módulo que permita trabajar con algoritmos de asignación de recursos en ambientes dinámicos en la transmisión óptica multibanda.

Debido a la necesidad de probar diferentes algoritmos y modulaciones para tener una EON que funcione de manera eficiente para diferentes demandas se utilizan simuladores. Los simuladores pueden ser utilizados para probar sistemas complejos que no se pueden evaluar de manera analítica, permitiendo realizar diferentes pruebas en distintos escenarios a bajo costo debido a que no se invierte en suministros físicos más allá del equipo computacional. Se pueden realizar complejas simulaciones como funcionamiento de bancos, determinación de requisitos o protocolos de hardware para redes de comunicaciones, o análisis de operaciones mineras [43]. En la siguiente sección, se explorará el estado del arte en simuladores de redes óptica para algoritmos de asignación de recursos en ambientes.

Capítulo 3

Estado del arte

En un ambiente de asignación de recursos dinámicos, los recursos se van asignando a medida de que las demandas llegan a la red. Esta asignación contempla la ruta que se asigna, el espectro de longitud de onda que se utiliza y la modulación. Todos estos parámetros pueden variar según el tráfico que llegue en cada momento, lo que implica que al momento de resolver el problema de asignación de recursos en una EON se tienen que considerar múltiples casos dependiendo de las distintas demandas que pueden llegar a la red.

A la fecha, no se encontraron simuladores que ayuden a resolver el problema con multibanda. Sin embargo, sí existen simuladores de redes ópticas, que están principalmente enfocados en EON y en WDM. Estos simuladores están escritos en lenguajes de programación como Java y C++, los cuales tienen distintos enfoques y funcionalidades. A continuación se presenta una lista de simuladores que se utilizan hoy en día en la academia para analizar su funcionamiento y características.

3.1. FlexGridSim

Simulador desarrollado en Java y que está basado en WDMSim [44] el cual se define como un simulador simple que está igualmente escrito en Java y que permite escenarios de Grooming (permite combinar varios tráficos de baja velocidad), y RSA en redes ópticas WDM [45]. FlexGridSim posee poca y desactualizada documentación, lo que dificulta su uso. Se facilita un repositorio en Github para su descarga y se recomienda el uso de Gradle para hacer la construcción, pero esto puede generar errores ocasionados por la falta de actualización de los archivos. Una vez descargado e instalado, se pueden proveer archivos del tipo *xml* para definir la red.

En las figuras 3.1 y 3.2 se puede ver la estructura básica que se tiene que entregar en el simulador para poder realizar las simulaciones, que serían los nodos y las conexiones de estos respectivamente. Esta estructura sigue el formato de *xml*.

```
<physical-topology name="USA" cores="7" slots="180" slotsBandwidth="12.5">
    <nodes>
        <node id="0" />
        <node id="1" />
        <node id="2" />
        <node id="3" />
        <node id="4" />
        <node id="5" />
        <node id="6" />
        <node id="7" />
        <node id="8" />
        <node id="9" />
        <node id="10" />
        <node id="11" />
        <node id="12" />
        <node id="13" />
        <node id="14" />
        <node id="15" />
        <node id="16" />
        <node id="17" />
        <node id="18" />
        <node id="19" />
        <node id="20" />
        <node id="21" />
        <node id="22" />
        <node id="23" />
    </nodes>
```

FIGURA 3.1: Archivo xml con nodos de ejemplo en Flexgrimsim.

```
    <links>
        <link id="0" source="0" destination="1" delay="4" bandwidth="192" weight="800" distance="800" />
        <link id="1" source="1" destination="0" delay="4" bandwidth="192" weight="800" distance="800" />
        <link id="2" source="0" destination="5" delay="5" bandwidth="192" weight="1000" distance="1000" />
        <link id="3" source="5" destination="0" delay="5" bandwidth="192" weight="1000" distance="1000" />
        <link id="4" source="1" destination="2" delay="5.5" bandwidth="192" weight="1100" distance="1100" />
        <link id="5" source="2" destination="1" delay="5.5" bandwidth="192" weight="1100" distance="1100" />
        <link id="6" source="1" destination="5" delay="4.75" bandwidth="192" weight="950" distance="950" />
        <link id="7" source="5" destination="1" delay="4.75" bandwidth="192" weight="950" distance="950" />
        <link id="8" source="2" destination="3" delay="1.25" bandwidth="192" weight="250" distance="250" />
        <link id="9" source="3" destination="2" delay="1.25" bandwidth="192" weight="250" distance="250" />
```

FIGURA 3.2: Archivo xml con *Links* de ejemplo en Flexgrimsim.

Se puede ver que no se trata de una estructura compleja para su construcción. Los parámetros que se pueden entregar para poder utilizar el simulador son el nombre del archivo *xml* y el número de simulaciones, dejando de manera opcional el parámetro trace si se quiere generar un archivo de seguimiento, o *verbose* si se quiere mostrar más información con un propósito de *debugging*.

Para el algoritmo de asignación se pueden utilizar las implementaciones que vienen por defecto en el simulador o crear un propio algoritmo. Para implementar un algoritmo se tiene que implementar el método RSA según la estructura base que se ve en la Figura 3.3. Por lo anterior, se tienen que saber manejar conceptos de herencia e interfaces, los cuales son propios del paradigma orientado a objetos en la programación.

```
public interface RSA {

    public void simulationInterface(Element xml, PhysicalTopology pt, VirtualTopology vt, ControlPlaneForRSA cp, TrafficGenerator traffic);

    public void flowArrival(Flow flow);

    public void flowDeparture(Flow flow);

}
```

FIGURA 3.3: Estructura para implementar RSA en FlexGridSim.

Este simulador al ser simple puede ser útil para hacer pruebas básicas de algoritmos ya implementados bajo distintas topologías. En cuanto a simulaciones más robustas (necesidad de mayor control de parámetros y aumento de tamaño de las topologías y peticiones) podría no cumplir con las necesidades del usuario. Por otro lado, la estructura para las topologías puede dificultar el uso del usuario respecto al simulador, dado que se puede tornar compleja si no se ha trabajado con archivos *xml*. Por último, este simulador no soporta multibanda, lo que es una barrera para usuarios que quieran probar sus algoritmos en entornos de este tipo. Retrasando la codificación al requerir de una intervención mayor del código para que se adapte a estas necesidades y obteniendo resultados en un periodo mayor de tiempo.

3.2. Complex Elastic Optical Network Simulator, CEONS

Simulador desarrollado en Java, con la peculiaridad de que facilita al usuario una interfaz gráfica de usuario o *Graphic User Interface* (GUI) para poder realizar las distintas simulaciones, utilizando archivos con la extensión *.eon* para manejar las redes de las simulaciones [46]. Permite escoger los algoritmos de asignación predefinidos *Adaptive Modulation and Regenerator-Aware* (AMRA), *Shortest Path First* (SPF), *DLBestScorePerPath* y *DLTotalScorePerPath*. Junto a esto, permite definir los parámetros: *Traffic Generators*, *Traffic data*, *Erlang*, *Seed*, *Alpha*, *Number of requests*, entre otros. En la Figura 3.4 se puede ver la interfaz que se utiliza, así como las configuraciones de los parámetros que se permiten dentro de la interfaz.

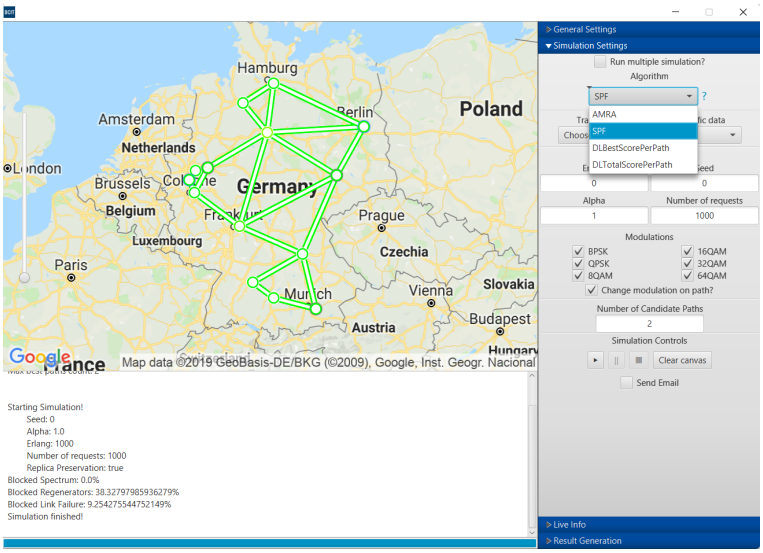


FIGURA 3.4: Interfaz gráfica del simulador CEONS.

En este simulador se puede apreciar que se intenta mostrar una red lo más realista posible mediante su integración con google maps, lo que puede ayudar al usuario para obtener simulaciones visuales para ciertos usos específicos. Un gran problema surge al momento de que se quieran utilizar algoritmos propios en este simulador. Esto debido a que, como se dijo anteriormente y se aprecia en la Figura 3.4, los algoritmos vienen predefinidos y no existe una opción para utilizar uno propio, por lo que, se puede utilizar para ver el comportamiento de los algoritmos propuestos en diferentes tipos de redes y utilizando diferentes parámetros. Por último, al igual que el anterior, este simulador no soporta multibanda, lo que es una barrera para

usuarios que quieran probar sus algoritmos en entornos de este tipo. Por esta razón, el proceso de codificación se pospone al exigir una intervención más extensa en el código para que se ajuste a estas necesidades, lo que conlleva a la obtención de resultados en un período prolongado.

3.3. OMNeT++

Simulador desarrollado principalmente en C++, con un enfoque que permite agregar distintos módulos [47]. Posee una arquitectura abierta y cuenta con una GUI como se puede ver en la Figura 3.5.

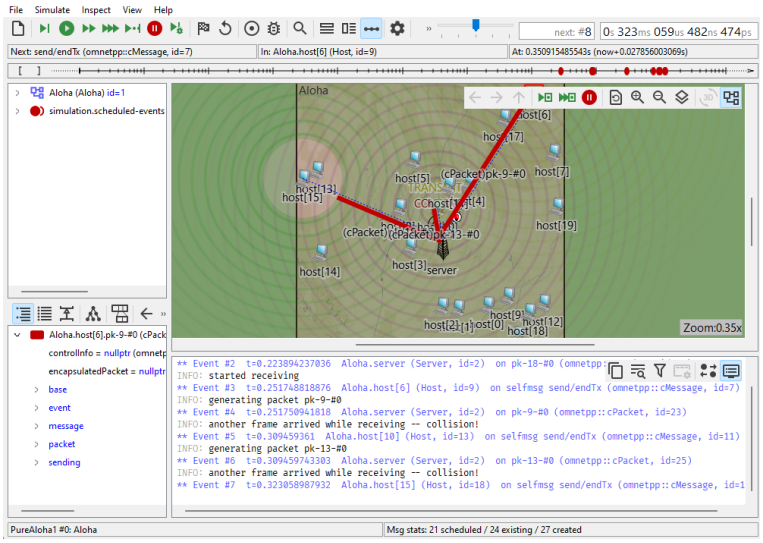


FIGURA 3.5: Interfaz gráfica del simulador OMNeT++.

Este simulador tiene como área principal de aplicación las redes de comunicación, junto con áreas secundarias como en sistemas TI, redes de colas, arquitecturas de hardware y procesos comerciales. No se tiene como enfoque directo las EON, por lo que, programar algoritmos de asignación para estas redes se torna complejo a nivel de programación.

Para realizar una simulación se necesitan 2 archivos, un archivo *ned* que contenga la descripción de la red a utilizar y un archivo *ini* que proporcione los valores que se quieren utilizar en la simulación.

Al ser un simulador muy extenso, se puede convertir en una barrera para utilizarlo. Se tiene un alta curva de aprendizaje y abarca bastantes áreas de las telecomunicaciones y otras. En cuanto a la documentación, es extensa y con actualizaciones constantes para mantener el sistema, lo que se vuelve una clara ventaja. Por último, al igual que los anteriores, este simulador no soporta multibanda, lo que es una barrera para usuarios que quieran probar sus algoritmos en entornos de este tipo. Esto implica una codificación más compleja para adaptarse a necesidades específicas, con resultados que requieren más tiempo.

3.4. Optical Network Simulator, ONS

Simulador desarrollado en Java que tiene por enfoque entregar una forma sencilla de implementar nuevos algoritmos para RWA, RSA y RMLSA [48]. Viene con estadísticas predefinidas y que no pueden ser modificadas. Permite simular tráfico mediante eventos discretos en redes WDM y EON. Se tiene el requisito de que el programador tiene que saber los conceptos de herencia e interfaces.

Tanto para entregar los nodos como sus respectivos *Links* se utilizan archivos del tipo xml. Para implementar un algoritmo se tiene que implementar el método RA según la estructura base de la clase. Además se entregan otros parámetros dentro de estos archivos como se puede ver en la Figura 3.6.

```
<physical-impairment
    physical-distance="no" SNRaware="no" checkQoT="no"
    activeAse="yes" activeNli="yes" NF="6" centerFrequency="193.0E+12" alfa="0.2" beta2="16.0E-24" gama="1.22" C="1.0"
    XTaware="no" dynamicNeighborNumber="no" XTonOthers="no"
    k="3.16E-5" r="0.055" beta="4.0E6" w_tr="45.0E-6" >
    <!-- 'modulation name' BPSK, QPSK, 8QAM, 16QAM, 32QAM, 64QAM, 128QAM, 256QAM -->
    <!-- 'XTthreshold' Crosstalk threshold. Only for SDM-EON. -->
    <!-- 'SNRthreshold' SNR threshold. -->
    <!-- 'PC' power consumption in W. -->
    <!-- 'maxReach' in km -->
    <modulation name="BPSK" XTthreshold="-14.0" SNRthreshold="6.0" PC="112.374" maxReach="8000"/>
</physical-impairment>
```

FIGURA 3.6: Archivo xml en ONS.

Entre los parámetros más relevantes que se pueden ver en la Figura 3.6 está que permite escoger entre distintos niveles de modulación y personalizar a distintos niveles el tráfico en la red. Al terminar una simulación se obtiene un resultado como en la Figura 3.7.

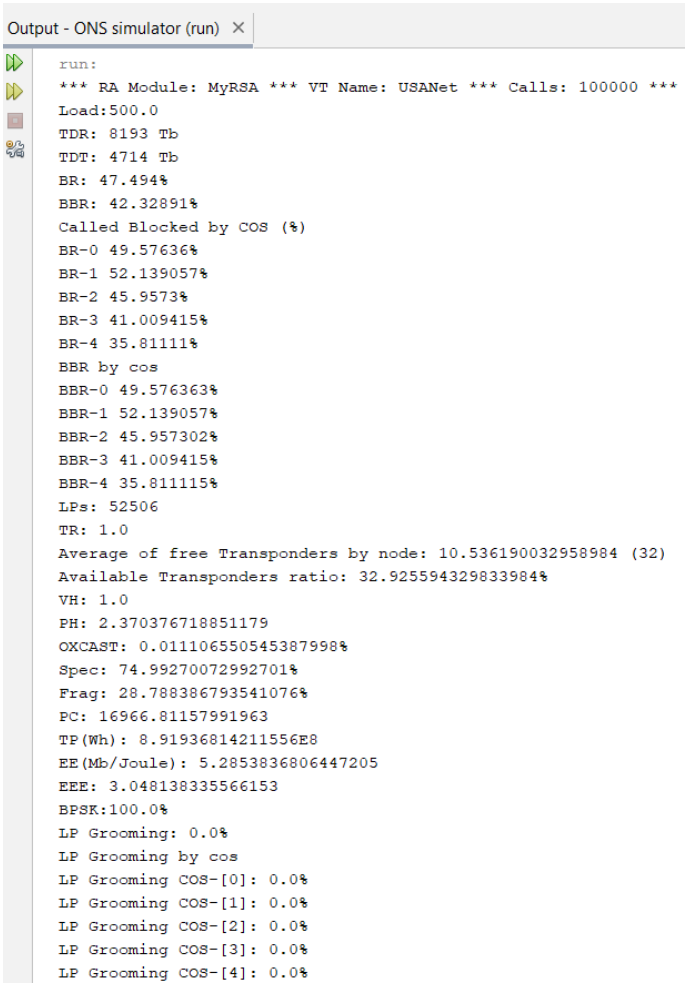


FIGURA 3.7: Salida en simulación ONS.

Este simulador cuanta con bastantes parámetros que se pueden personalizar ayudando al usuario a probar escenarios específicos dependiendo de lo que se desea probar. En casos donde se quiera realizar simulaciones simples puede suponer un problema y no se cuenta con apoyo visual para ver la información de la simulación. Los resultados finales de la simulación pueden ser abrumadores dada la cantidad de *Data* que se entrega, como se ve en la Figura 3.7 donde se puede perjudicar al usuario al buscar entre tantos resultados la información que necesita. Por último, al igual que los anteriores, este simulador no soporta multibanda, lo que es una barrera para usuarios que quieran probar sus algoritmos en entornos de este tipo. Esto demora la codificación con cambios en el código para adaptarse a necesidades específicas, lo que resulta en un periodo de tiempo más largo para obtener resultados.

3.5. ElasticO++

Software flexible que es un *framework* para Omnet++ [49]. Esta pensado principalmente para EON y tiene una estructura predefinida para los algoritmos de asignación de recursos que se puede ver en la Figura 3.8, por lo que si se quiere crear un algoritmo se tiene que seguir dicha estructura. Al igual ONS, funciona utilizando los conceptos de herencia e interfaces. Actualmente, el repositorio entregado por los

creadores del software no se encuentra disponible por lo que, no está disponible para su descarga.

Al basarse en el simulador OMNeT++ se opera con la misma base, es decir, también se requieren 2 archivos principales ned e ini para poder definir las redes y definir los valores de la simulación. Igualmente es compatible con la GUI que entrega OMNeT++ para sus simulaciones.

```
1  struct AllocationResult {
2  public:
3      string allocation_log;
4      long int connection_id;
5      int lower_slot_index;
6      string modulation;
7      int number_of_slots;
8      bool rejection_after_defragmentation;
9      int rejection_cause;
10     int route_id;
11     bool success;
12     bool success_after_defragmentation;
13 };
14
15 struct AllocationRequest {
16 public:
17     int source;
18     int destination;
19     int bitrate;
20 };
21
22 struct AllocationRestriction {
23 public:
24     int lower_slot_index;
25     int higher_slot_index;
26     int num_slots;
27     int route_index;
28     int route_id;
29     double route_longest_link_length;
30 };
```

FIGURA 3.8: Estructura de algoritmos de asignación en Elastico++ [49].

Este simulador se presenta como una solución prometedora para utilizar OMNeT++ con un foco en las redes ópticas elásticas. En cuanto a la estructura, puede ser una limitante para usuarios que quieran probar un algoritmo de alojamiento totalmente personalizado. El hecho de que no se pueda acceder al simulador desde su repositorio impide que se pueda evaluar su funcionamiento de manera exhaustiva. Por último, al igual que los anteriores, este simulador no soporta multibanda, lo que es una barrera para usuarios que quieran probar sus algoritmos en entornos de este tipo. Al momento de programar, se retarda la codificación mediante cambios extensos en el código para adecuarse a necesidades particulares, resultando en un tiempo prolongado para obtener resultados.

3.6. Flex Net Sim

Biblioteca de C++ enfocada en diseñar algoritmos de asignación para EON [50]. El enfoque que tiene este simulador es que los investigadores se preocupen de los algoritmos de asignación y no en el simulador en sí. En la Figura 3.9 se puede ver el resultado de una simulación en este simulador.

```
Nodes:              14
Links:              42
Goal Connections:   1000000
Lambda:             5000
Mu:                 40
Algorithm:          FirstFit

+----------+----------+----------+----------+----------+----------+----------+
| progress | arrives  | blocking | time(s)  | Wald CI  | A-C. CI  | Wilson CI|
+----------+----------+----------+----------+----------+----------+----------+
|    5.0%  |   50000  |  3.3e-01 |       6  |  4.1e-03 |  4.1e-03 |  4.1e-03 |
|   10.0%  |  100000  |  3.3e-01 |      12  |  2.9e-03 |  2.9e-03 |  2.9e-03 |
|   15.0%  |  150000  |  3.3e-01 |      19  |  2.4e-03 |  2.4e-03 |  2.4e-03 |
|   20.0%  |  200000  |  3.3e-01 |      25  |  2.1e-03 |  2.1e-03 |  2.1e-03 |
|   25.0%  |  250000  |  3.3e-01 |      31  |  1.8e-03 |  1.8e-03 |  1.8e-03 |
|   30.0%  |  300000  |  3.3e-01 |      37  |  1.7e-03 |  1.7e-03 |  1.7e-03 |
|   35.0%  |  350000  |  3.3e-01 |      43  |  1.6e-03 |  1.6e-03 |  1.6e-03 |
|   40.0%  |  400000  |  3.3e-01 |      50  |  1.5e-03 |  1.5e-03 |  1.5e-03 |
|   45.0%  |  450000  |  3.3e-01 |      56  |  1.4e-03 |  1.4e-03 |  1.4e-03 |
|   50.0%  |  500000  |  3.3e-01 |      62  |  1.3e-03 |  1.3e-03 |  1.3e-03 |
|   55.0%  |  550000  |  3.3e-01 |      68  |  1.2e-03 |  1.2e-03 |  1.2e-03 |
|   60.0%  |  600000  |  3.3e-01 |      74  |  1.2e-03 |  1.2e-03 |  1.2e-03 |
|   65.0%  |  650000  |  3.3e-01 |      81  |  1.1e-03 |  1.1e-03 |  1.1e-03 |
|   70.0%  |  700000  |  3.3e-01 |      87  |  1.1e-03 |  1.1e-03 |  1.1e-03 |
|   75.0%  |  750000  |  3.3e-01 |      93  |  1.1e-03 |  1.1e-03 |  1.1e-03 |
|   80.0%  |  800000  |  3.3e-01 |      99  |  1.0e-03 |  1.0e-03 |  1.0e-03 |
|   85.0%  |  850000  |  3.3e-01 |     106  |  1.0e-03 |  1.0e-03 |  1.0e-03 |
|   90.0%  |  900000  |  3.3e-01 |     112  |  9.7e-04 |  9.7e-04 |  9.7e-04 |
|   95.0%  |  950000  |  3.3e-01 |     118  |  9.5e-04 |  9.5e-04 |  9.5e-04 |
|  100.0%  | 1000000  |  3.3e-01 |     124  |  9.2e-04 |  9.2e-04 |  9.2e-04 |
```

FIGURA 3.9: Salida del simulador *Flex Net Sim*.

En la Figura 3.9 se van entregado a tiempo real los resultados de la simulación, de manera que se logran obtener resultados parciales sin la necesidad de que termine de ejecutarse el código completo. Se puede implementar un algoritmo de asignación de recursos sin seguir una estructura predefinida, además de que se pueden utilizar los algoritmos *First Fit* (FF), *Exact Fit* (EF) o *First-Last-Fit* (FLF) que vienen implementados en los ejemplos del simulador. Para definir los nodos y las rutas que se utilizarán en la simulación se utilizan archivos *Json*.

El formato *Json*, que significa *JavaScript Object Notation*, es un formato de texto ligero para el intercambio de datos, que se compone de pares de claves y valores, donde las claves son siempre cadenas de texto y los valores pueden ser cadenas de texto, números, objetos *Json*, *arrays*, *booleanos* (verdadero o falso) o null. Esto entrega un formato sencillo, más fácil de utilizar en cuanto a la estructura y más rápido si se compara con un *xml* [51]. Por otro lado, permite al usuario programar su propio algoritmo de asignación de recursos sin una estructura definida más allá de las limitaciones del lenguaje de programación. Para utilizar los algoritmos preexistentes hay que agregar código o basarse en los ejemplos que se entregan, lo que puede suponer un inconveniente para usuarios que quieran probar topologías complejas con solo algoritmos ya configurados. Por último, al igual que los anteriores, este simulador no soporta multibanda, lo que es una barrera para usuarios que quieran probar sus algoritmos en entornos de este tipo. En cuanto a la codificación, se ve prolongada a través de modificaciones en el código, lo cual conlleva a una demora considerable en la obtención de resultados.

3.7. Comparación de los simuladores

Considerando que uno de los principales focos de utilización de los simuladores es la asignación de recursos se revisarán los siguientes parámetros:

- Versatilidad: Nivel de personalización en los algoritmos de asignación a implementar por el usuario. Tendrá los siguientes niveles:

  - Baja: Solo permite probar los algoritmos de asignación previamente configurados.
  - Media: Permite al usuario implementar sus propios algoritmos bajo una estructura predefinida por el programa.
  - Alta: Permite implementar algoritmos de asignación sin una estructura fija más allá de las limitaciones del lenguaje de programación.
- Adaptabilidad: Nivel de manejo de los parámetros por parte del usuario. Tendrá los siguientes niveles:
  - Baja: No permite seleccionar la modulación que se desea utilizar y se tiene un bajo control sobre parámetros generales como λ, μ, Número de peticiones, entre otros.
  - Media: Se puede seleccionar la modulación y manejar parámetros generales a nivel medio.
  - Alta: Se puede seleccionar la modulación y manejar parámetros generales a nivel alto.
- Soporte multibanda: Si el simulador soporta redes elásticas multibanda.
- Documentación: Existencia de documentación para utilizar el simulador. Basándose en los siguientes niveles:
  - Bajo: Solo proporciona documentación básica sobre el uso general del simulador.
  - Medio: Ofrece documentación detallada que abarca la configuración y personalización de parámetros estándar del simulador.
  - Alto: Incluye documentación detallada que permite a los usuarios crear y modificar parámetros avanzados, así como desarrollar extensiones y personalizaciones significativas en el simulador.

Simuladores redes ópticas				
Métrica	Versatilidad	Adaptabilidad	Soporte multibanda	Documentación
FlexGridSim	Media	Baja	No	Baja
CEONS	Baja	Media	No	Media
Omnett++	Media	Media	No	Alta
ONS	Media	Alta	No	Alta
ElasticO++	Media	Media	No	Media
Flex Net Sim	Alta	Alta	No	Alta

TABLA 3.1: Comparación de simuladores.

3.8. Conclusiones

Los simuladores existentes al día de hoy para EON son variados y poseen distintos enfoques según las necesidades del investigador, sin embargo, ninguno de los simuladores que se encontraron permiten un soporte para multibanda. Esto implica que los investigadores tengan que programar sus propios simuladores que soporten multibanda para poder probar algoritmos de asignación de recursos o ver el comportamiento de una EON.

El simulador escogido para realizar la implementación de un módulo que soporte multibanda es Flex Net Sim. Debido a que se está trabajando con el equipo desarrollador de esta biblioteca y que permite una alta versatilidad para algoritmos de asignación de recursos. Al tratarse de un simulador ya existente se puede revisar solo el módulo que se desea implementar y no realizar un simulador desde cero que implica otras áreas que ya se implementan en los simuladores revisados. Otra característica de este simulador es que trabaja utilizando módulos, por lo que, se simplifica su modificación sin intervenir de manera abrupta al código base.

Capítulo 4

Hipótesis y objetivos

Hipótesis de Investigación: Un módulo dedicado a multibanda disminuirá el tiempo de codificación de algoritmos de asignación de recursos en ambientes multibanda.

Objetivo General: Desarrollar una nueva herramienta de simulación (biblioteca en C++) para facilitar la validación de algoritmos de asignación de recursos en redes ópticas elásticas con multibanda.

Objetivos Específicos:

- Recolectar el estado del arte de redes ópticas elásticas, algoritmos de asignación de recursos en redes ópticas elásticas, redes ópticas multibanda y simuladores en redes ópticas.
- Diseñar un módulo de redes ópticas multibanda para el simulador *Flex Net Sim*.
- Implementar el módulo de redes ópticas multibanda.
- Realizar pruebas en ambientes multibanda para comprobar el correcto funcionamiento del módulo.
- Crear documentación necesaria para la utilización del módulo.
- Implementar y replicar resultados de algoritmos de asignación de recursos existentes en el estado del arte.

Capítulo 5

Metodología

El plan de trabajo consta de diseñar e implementar un módulo para el simulador *Flex Net Sim*. Además, se realizarán las pruebas necesarias para asegurar su correcto funcionamiento. Luego se procederá a crear la documentación y publicar los resultados obtenidos.

Las principales actividades para realizar serán:

1. Revisión de literatura para conocer el estado del arte de las redes ópticas Multibanda.
2. Revisión de literatura para conocer el estado del arte de simuladores en redes ópticas Multibanda.
3. Comprensión del funcionamiento de la biblioteca de simulación orientada a eventos en C++ *Flex Net Sim*.
4. Diseño del módulo a realizar y pensar en la lógica de este.
5. Implementación de un módulo dentro de la biblioteca *Flex Net Sim*.
6. Diseño de las pruebas para el módulo.
7. Ejecución de las distintas pruebas para comprobar el funcionamiento del módulo.
8. Codificación de un algoritmo de asignación de recursos en ambientes Multibanda utilizando el simulador *Flex Net Sim* con el nuevo módulo incorporado.
9. Realización de la documentación de la biblioteca.
10. Publicación de la biblioteca en una revista o conferencia del área.

En primer lugar, se realizará una revisión exhaustiva de bases de datos académicas y bibliotecas virtuales, utilizando términos de búsqueda relevantes junto con criterios de inclusión y exclusión. En segundo lugar, se seleccionarán los artículos más pertinentes y se analizarán críticamente, extrayendo información clave relacionada con los objetivos planteados. Por último, se revisarán las referencias bibliográficas de los *paper* seleccionados para identificar estudios complementarios. La síntesis de la información obtenida permitirá cumplir el primer objetivo específico y generar un marco teórico sólido para la investigación.

La comprensión del funcionamiento del simulador abarcará un enfoque integral, combinando un análisis teórico exhaustivo con una evaluación práctica basada en ejemplos ejecutables. Esta combinación de perspectivas permitirá obtener un panorama completo de las capacidades y limitaciones del simulador, sentando las bases

para su correcta utilización y su futura modificación para la inclusión del módulo de Multibanda.

Se utilizará una metodología ágil, lo que permite adaptar la forma de trabajo a las condiciones cambiantes del proyecto, de manera que se consiga flexibilidad e inmediatez al momento de realizar las actividades propuestas. Existen variadas formas de metodología ágiles, como por ejemplo: Scrum, *Extreme Programming* (XP), *Feature Driven Development* (FDD), *Dynamic Software Development Method* (DSDM), Kanban [52].

La metodología ágil Scrum [53] tiene un enfoque interactivo e incremental para desarrollar proyectos complejos. Se proponen 3 actores principales: *Product owner*,*Scrum master* y *Project team* o *Scrum team*. El *product owner* formula el plan de acuerdo al problema y divide las tareas para obtener resultados de forma funcional. El *Scrum master* es una persona que lidera el equipo de desarrollo y es el puente de comunicación entre el *product owner* y el equipo de desarrollo. Mientras que el *Project team* es un grupo de entre 5 a 9 personas calificadas para ayudar a lograr los objetivos del proyecto. Esta metodología no sería aplicable a este proyecto, debido a que su enfoque interactivo e incremental se sustenta en la colaboración y comunicación entre diversos actores. En ausencia de múltiples participantes, los elementos clave de *Scrum*, como el*Product Owner*, el *Scrum Master* y el equipo de desarrollo, no estarían presentes, limitando la utilidad y eficacia de esta metodología.

Otra metodología ágil es XP [54], donde se tienen seis fases: exploración, planificación, iteraciones para liberar, producción, mantenimiento y muerte. En la fase de exploración se busca la familiarización con la tecnología, herramientas y prácticas que se utilizarán. En la planificación se estima el esfuerzo requerido y se realiza un calendario para liberar el producto. Las fases de iteraciones, tal y como su nombre indica, son distintas iteraciones para producir la primera versión. En la producción se llevan a cabo pruebas y comprobaciones de rendimiento adicionales. En el mantenimiento se producen nuevas iteraciones para implementar cambios y nuevas solicitudes. Por último, la fase de muerte corresponde a completar la documentación y no existe una propuesta de valor para mejorar el sistema. La metodología XP, aunque eficaz para proyectos ágiles, puede no ser adecuada para el desarrollo de un módulo para una biblioteca, esto debido a su enfoque iterativo y su énfasis en la entrega de productos funcionales.

Por otro lado, en FFD [55] existen dos etapas principales, la primera es para descubrir la lista de características que se busca implementar, mientras que la segunda es la implementación de estas características una por una. En la primera etapa, los clientes tienen que actuar activamente y se van creando los diagramas de modelado unificado o *Unified Modeling Language* (UML), que constan de un conjunto integrado de diagramas pensado para ayudar a los desarrolladores a especificar, visualizar, construir y documentar los modelos, bases de datos, *scripts*, llamados *artefactos de sistema de software*. En la segunda etapa, la lista de características tiene que ser lo suficientemente pequeña para desarrollar el *software* en iteraciones cortas. Debido a su enfoque activo por parte de los clientes y la necesidad de crear diagramas de modelado unificado, puede no ser la mejor alternativa para utilizarla en este caso porque puede resultar excesivo y poco eficiente en términos de recursos y tiempo.

Por último, en Kanban se tiene un tablero o tabla dividido en distintas columnas donde se van mostrando los flujos de la producción de *software* [56]. Para esto se

pueden utilizar distintos *softwares* como *Trello* [1] o *Teamwork* [2]. Con la evolución del desarrollo, evoluciona la información de la tabla, agregando tarjetas que contienen toda la información necesaria acerca de las tareas a realizar. De esta forma se asegura tener un orden, tareas establecidas y un correcto seguimiento del proceso del proyecto. Esta metodología tiene un enfoque visual y flexible que permite un seguimiento eficiente del flujo de trabajo, garantizando un orden establecido y una gestión eficaz de las tareas, lo cual es fundamental para el éxito del proyecto.

La tabla 5.1, creada a partir de [57], muestra una comparación de las metodologías ágiles vistas anteriormente. Para la realización de este proyecto se eligió Kanban debido a que se trata de una metodología con un enfoque incremental, lo cual se adecúa a la situación dado que se parte en base a un simulador. También, tanto el tamaño del equipo como del proyecto coinciden con lo propuesto. Por último, la comunicación e intervención del cliente (Equipo de *Flex net sim*) se planea realizar de forma directa y en los lanzamientos incrementales.

Característica	**Scrum**	**XP**	**FDD**	**DSDM**	**Kanban**
Enfoque de desarrollo	Iterativo e incremental	Iterativo e incremental	Iterativo e incremental	Iterativo e Incremental	Incremental
Tiempo para una iteración	2-3 semanas	1-6 semanas	$<$ 2 semanas	2-3 días	Tiempo que toma desde el inicio del trabajo hasta que existen correcciones
Tamaño del equipo	7+/-2	$<$ 20	Muchos miembros (más de un equipo)	Equipos independientes de cualquier tamaño	Cualquier experto o equipo multidisciplinar
Tamaño del proyecto	Cualquier tipo de proyecto	Proyecto pequeño	Proyecto complejo	Cualquier tipo de proyecto	Proyectos pequeños
Comunicación del equipo	Informal	Informal	Basado en documentación	Basado en documentación	Informal cara a cara
Intervención del cliente	*Product owner* actúa sobre nombre de cliente en todo el proceso	A lo largo del proceso	A través de informes	Involucrado en lanzamientos frecuentes	Involucrado en lanzamientos incrementales

TABLA 5.1: Comparación metodologías ágiles.

Continuando con la metodología ágil, se definirán distintos sprints, los cuales corresponden a un ciclo de ejecución corto (puede ser entre una y cuatro semanas) que tienen como objetivo aumentar el valor del producto que se está construyendo. En el caso de este proyecto sería el módulo para la biblioteca *Flex Net Sim* que permite

[1] https://trello.com/es
[2] https://www.teamwork.com/

trabajar con algoritmos de asignación de recursos en redes ópticas elásticas Multibanda.

Para complementar la metodología a utilizar, se incluirá GIT [58] dentro de la realización del proyecto. GIT es un sistema de control de versiones utilizado en la realización de software y distintos proyectos. Una de sus grandes ventajas es que tiene una arquitectura distribuida, lo que implica que el historial de versiones del software completo se almacena en un repositorio que puede ser accedido de distintas maneras y llevar un control de cambios y documentación eficiente y seguro. Esta herramienta va ligada al desarrollo ágil debido a su capacidad de ir actualizando el repositorio e ir guardando las distintas versiones, lo que facilita los cambios y la definición de tareas específicas sin influir en otras partes del software.

Para utilizar GIT es necesario un servidor remoto que se utiliza para alojar y gestionar repositorios de código. Este servidor proporciona un espacio centralizado donde los desarrolladores pueden almacenar y compartir su código fuente, colaborar en proyectos y controlar las versiones de manera eficiente. En este caso, se utilizará *GitLab* como servidor remoto para alojar el repositorio GIT. *GitLab* es una plataforma popular que permite a los equipos colaborar en el desarrollo de software de manera eficiente. Al igual que otros servidores remotos como GitHub y Bitbucket, GitLab ofrece la posibilidad de alojar repositorios GIT de forma remota. La razón de esta elección es que el simulador flex net sim tiene su código fuente alojado en este servidor.

En el proceso de implementación del diseño, se utilizará el código oficial de la biblioteca [3]. Para llevar a cabo las modificaciones y desarrollos necesarios, se hará un *Fork* del repositorio original, que es una acción que implica crear una copia independiente de un repositorio existente. Esta copia se realiza con el propósito de trabajar en el código de manera separada y paralela al repositorio original. Esto permitirá crear una nueva rama específica para el desarrollo en cuestión. Este enfoque de trabajo en ramas separadas asegura una gestión efectiva de las modificaciones y una separación clara entre el código original y las nuevas contribuciones.

Para la codificación de un algoritmo de asignación de recursos en ambientes Multibanda se seguirán los siguientes pasos: investigación y revisión del estado del arte de algoritmos Multibanda, selección del algoritmo, configuración del simulador, implementación del algoritmo, evaluación y validación. Esto se hará en el nuevo módulo para flex net sim y en mismo simulador sin el módulo implementado.

En cuanto a la documentación de la biblioteca se llevará a cabo un análisis del módulo, desglosando sus funciones, características y requisitos técnicos. A continuación, se elaborará una estructura clara y coherente, incluyendo secciones como descripción general, uso y ejemplos de código. Se prestará especial atención a la claridad y concisión, asegurando que los usuarios puedan comprender rápidamente la funcionalidad del módulo y cómo utilizarlo correctamente. Además, se proporcionarán explicaciones detalladas de los parámetros y opciones disponibles. La documentación también incluirá consideraciones de buenas prácticas de uso y posibles problemas conocidos con sus soluciones correspondientes.

[3] https://gitlab.com/DaniloBorquez/flex-net-sim/-/tree/master

Capítulo 6

Resultados y análisis

El simulador *Flex Net Sim* (FNS) se basa en una arquitectura modular que permite integrar diferentes componentes y funcionalidades de acuerdo con las necesidades del usuario. Esta arquitectura se puede ver de forma resumida en la Figura 6.1 donde se encuentran los componentes principales que contiene el simulador. Cada componente se ocupa de su propia función. El componente *Network* contiene la información de la red y está compuesto por las clases *Network*, *Node* y *Link*. Estas se ocupan respectivamente de los componentes de la red general, de los nodos de la red y los enlaces entre los nodos. Dentro de la clase Link se encuentra la cantidad de *Slots*. El tamaño de los *Slots* utilizados en el simulador puede ser definido por el usuario mientras se mantenga el mismo por toda la red. Sumado a lo anterior, existe un conjunto de métodos que permiten cambiar el estado de la red. Todos estos métodos son *atomic*, lo que significa que permiten cambiar el estado de un único enlace o una sola ranura. Por ejemplo, dentro de este componente no existe el concepto de *lightpath*, ya que la red es el componente más primitivo de la biblioteca y opera exclusivamente a nivel de enlaces y nodos.

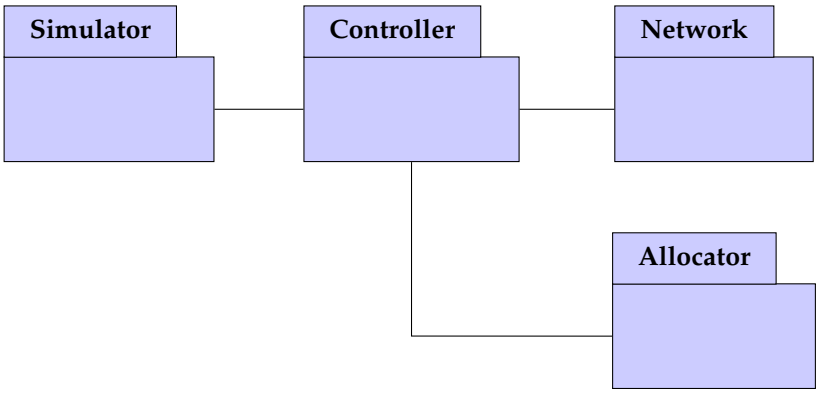


FIGURA 6.1: Componentes del simulador FNS.

Conectado a *Network* se encuentra el controlador o *Controller*. Este componente gestiona la información más global de la red, como las rutas existentes entre cada par de nodos y las conexiones establecidas. Aquí se encuentran los métodos para asignar o desasignar una conexión entre cualquier par de nodos. Los métodos aquí expuestos utilizan de manera más amplia los métodos de la red. Es posible cambiar el estado de múltiples *slots* con un solo método del controlador o configurar los *lightpaths* que serán utilizados por las conexiones.

El componente *Simulator* lleva a cabo cada simulación dentro de la biblioteca. En este componente, se generan de manera aleatoria las solicitudes de conexión, las

cuales se representan mediante la tripleta compuesta por el nodo fuente, el nodo destino y el *Bit rate*. El número de solicitudes está configurado como parámetro y es definido por el usuario. Posteriormente, cada solicitud de conexión se transmite al controlador, que se encargará de intentar su asignación utilizando el componente *Allocator*.

En último término, el *Allocator* representa el componente en el cual el usuario desarrolla el algoritmo de asignación que será sometido a prueba mediante una clase heredada, la cual permite a los usuarios extender y personalizar la funcionalidad del *Allocator* permitiendo utilizar el algoritmo del usuario. Este constituye el único punto donde el usuario aporta activamente para la configuración del simulador. Esta flexibilidad en la estructura proporciona a los investigadores la oportunidad de enfocarse exclusivamente en el desarrollo del algoritmo.

6.1. Ejemplo del algoritmo First Fit en Flex Net Sim

En esta sección, se mostrará paso a paso un ejemplo del algoritmo *First Fit* en el simulador *Flex Net Sim* para mostrar el funcionamiento de la programación de algoritmos de asignación de recursos en el simulador. En el Algoritmo 1 se puede ver el pseudocódigo. El algoritmo recorre la lista de los *Slots* totales, que representan los espacios disponibles, y busca un conjunto de *Slots* contiguos vacíos, o que tengan un estado igual a *False*, para asignar recursos. Si encuentra un conjunto de *Slots* que sea igual o mayor al espacio requerido, los asigna y devuelve "*ALLOCATED*". Si no se pueden asignar los recursos, devuelve "*NOT_ALLOCATED*".

Algorithm 1 Algoritmo First Fit

```
1:  procedure FIRST_FIT(total_slots, required_slots)
2:      currentNumberSlots ← 0
3:      for each slot in total_slots do
4:          if slot == False then
5:              currentNumberSlots ← currentNumberSlots + 1
6:          else
7:              currentNumberSlots ← 0
8:          end if
9:          if currentNumberSlots ≥ required_slots then
10:             allocate_slots()
11:             return ALLOCATED
12:         end if
13:     end for
14:     return NOT_ALLOCATED
15: end procedure
```

Al inicio se incluye la biblioteca mediante `#include <fnsim/simulator.hpp>` como se ve en la primera línea del el Código 6.1. Esta biblioteca utiliza Macros para facilitar la programación a los usuarios. En este ejemplo, se hará uso extensivo de estas macros, las cuales se destacan por estar en mayúsculas. A continuación, se procede con la declaración de la función de asignación en la linea 2, identificada como `BEGIN_ALLOC_FUNCTION`. Esta declaración recibe como parámetro el nombre del algoritmo, el cual puede ser elegido libremente. Se sigue declarando variables relacionadas a los *Slots* que se usarán más adelante.

CODE 6.1: Inicialización de variables en *First Fit*.

```
#include <fnsim/simulator.hpp>
BEGIN_ALLOC_FUNCTION(ExactFit) {
    int currentNumberSlots = 0;
    int currentSlotIndex = 0;
    std::vector<bool> totalSlots;
    int numberOfSlots = REQ_SLOTS(0);
```

El estado de cada *Slot* en la ruta se guarda en el vector `totalSlots` mostrado en el Código 6.2. Esta variable es necesaria debido a que la conexión es transparente. Además, opera mediante la "compresión" de los *Slots* de cada enlace en un "enlace representativo". Esta compresión es necesaria para garantizar el cumplimiento de las restricciones de continuidad y contigüidad en la ruta. Para hacerlo, se utiliza `LINK_IN_ROUTE(0, 0)`, donde el primer valor representa la ruta y el segundo el enlace. Las rutas se encuentran precalculadas en un archivo *Json* como el de la Figura 6.2, donde se encuentra el nodo fuente (*Src*), el nodo destino (*Dst*) y las rutas (*Paths*) por las que se conectan. Estas rutas están representadas con una lista que contiene los identificadores de los enlaces por los que se tiene que pasar para llegar desde *Src* a *Dst*. Se cuenta con varias rutas donde cada una tiene su respectiva cantidad de enlaces. Como todos los enlaces tienen la misma cantidad de *Slots* se elige arbitrariamente un enlace representativo, en este caso se escoge el primero, asociado a la ruta 0 y enlace 0. Este objeto de enlace, con el método `getSlots`, proporciona el vector de *Slots* por enlace, y `totalSlots` almacena esta cantidad con el valor predeterminado `false` para indicar que la ranura está disponible.

```
{
    "alias": "BDM_USNet",
    "name": "BDM_USNet",
    "routes": [
        {
            "src": 0,
            "dst": 1,
            "paths": [
                [
                    0,
                    1
                ],
                [
                    0,
                    2,
                    3,
                    1
                ],
            ]
        }
```

FIGURA 6.2: Ejemplo de archivo *Json* para rutas.

CODE 6.2: Definición de `totalSlots` en *First Fit*.

```
    totalSlots = std::vector<bool>(LINK_IN_ROUTE(0, 0)->getSlots(), false);
```

El siguiente paso es actualizar el estado predeterminado de los *Slots* para comprobar que se cumplen las restricciones de continuidad y contigüidad. Para hacerlo, recorreremos cada *Slot* del enlace con un doble bucle mostrado en el Código 6.3. La macro `NUMBER_OF_LINKS(i)` obtiene el número de enlaces en la ruta `i`.

CODE 6.3: Actualización del estado de los *Slots* en *First Fit*.

```
    for (int j = 0; j < NUMBER_OF_LINKS(i); j++) {
        for (int k = 0; k < LINK_IN_ROUTE(i, j)->getSlots(); k++) {
            totalSlots[k] = totalSlots[k] | LINK_IN_ROUTE(i, j)->getSlot(k);
        }
    }
```

Luego, se inicia un bucle que revisa el estado de cada *Slot* en el vector de *Totalslots*. En caso de estar el *Slot* disponible, se suma 1 a la variable de los *Slots* disponibles (*currentNumberSlots*). Si el *Slot* se encuentra ocupado, se reinicia la variable *currentNumberSlots* y se actualiza la variable del índice actual (*currentSlotIndex*). Esto se puede ver en el Código 6.3.

CODE 6.4: Revisión del estado de los *Slots* en *First Fit*.

```
    for (int j = 0; j < totalSlots.size(); j++) {
      if (totalSlots[j] == false) {
        currentNumberSlots++;
      } else {
        currentNumberSlots = 0;
        currentSlotIndex = j + 1;
      }
```

Finalmente se utiliza `ALLOC_SLOTS` para asignar los *slots* en el enlace como se ve en el Código 6.5. Esta macro recibe tres parámetros: el primero es el ID del enlace, el segundo es la posición de inicio para asignar y el tercero es el número de *Slots* a asignar. Al asignar los *Slots* se marcan como ocupados, que es cambiar su estado a *True*.

CODE 6.5: Asignación de *Slots* en *First Fit*.

```
      if (currentNumberSlots == numberOfSlots) {
        for (int j = 0; j < NUMBER_OF_LINKS(i); j++) {
          ALLOC_SLOTS(LINK_IN_ROUTE_ID(i, j), currentSlotIndex, numberOfSlots)
        }
        return ALLOCATED;
      }
    }
  }
  return NOT_ALLOCATED;
}
END_ALLOC_FUNCTION
```

Si el ciclo se completa y no es posible asignar los *Slots* requeridos, se coloca que el algoritmo de asignación retorne `NOT_ALLOCATED`. En caso contrario, se retornará `ALLOCATED`. Estos valores son los que reconoce la biblioteca y utiliza para calcular la probabilidad de bloqueo. La finalización de la función de asignación debe concluir obligatoriamente con `END_ALLOC_FUNCTION`.

En resumen, el algoritmo recorre la lista de los *Slots* totales, que representan los espacios disponibles, y busca un conjunto de *Slots* contiguos vacíos o que tengan un estado igual a *False* para asignar recursos. Si encuentra un conjunto de *Slots* igual o mayor al espacio requerido, los asigna y se retorna `ALLOCATED`. Si no se pueden asignar los recursos, devuelve `NOT_ALLOCATED`. El código completo del algoritmo se ve en el Código 6.6.

CODE 6.6: Código completo de algoritmo *First Fit*.

```
#include <fnsim/simulator.hpp>
BEGIN_ALLOC_FUNCTION(ExactFit) {
    int currentNumberSlots = 0;
```

```
    int currentSlotIndex = 0;
    std::vector<bool> totalSlots;
    int numberOfSlots = REQ_SLOTS(0);
    totalSlots = std::vector<bool>(LINK_IN_ROUTE(0, 0)->getSlots(), false);
    for (int j = 0; j < NUMBER_OF_LINKS(i); j++) {
        for (int k = 0; k < LINK_IN_ROUTE(i, j)->getSlots(); k++) {
            totalSlots[k] = totalSlots[k] | LINK_IN_ROUTE(i, j)->getSlot(k);
        }
    }
    for (int j = 0; j < totalSlots.size(); j++) {
      if (totalSlots[j] == false) {
        currentNumberSlots++;
      } else {
        currentNumberSlots = 0;
        currentSlotIndex = j + 1;
      }
      if (currentNumberSlots == numberOfSlots) {
        for (int j = 0; j < NUMBER_OF_LINKS(i); j++) {
          ALLOC_SLOTS(LINK_IN_ROUTE_ID(i, j), currentSlotIndex, numberOfSlots)
        }
        return ALLOCATED;
      }
    }
  }
  return NOT_ALLOCATED;
}
END_ALLOC_FUNCTION
```

6.2. Clases del Simulador

En la Figura 6.3 se presenta el diagrama de componentes que muestra las clases que contienen los distintos componentes. *Simulator* contiene clases relacionadas a los eventos, variables aleatorias y otras funcionalidades para la generación de la simulación. El componenete *Controller* gestiona las conexiones con la clase *Controller* y contiene la estructura de estas. El componente *Allocator* maneja la asignación de las conexiones con la clase *Allocator* y recibe el *Bit rate* de la conexión respectiva. Por último, *Network* contiene clases relacionadas con la topología de la red, como nodos y enlaces. Dentro del diagrama se representa cómo interactúan cada una de las clases y se representa con líneas negras gruesas la interacción de los componentes.

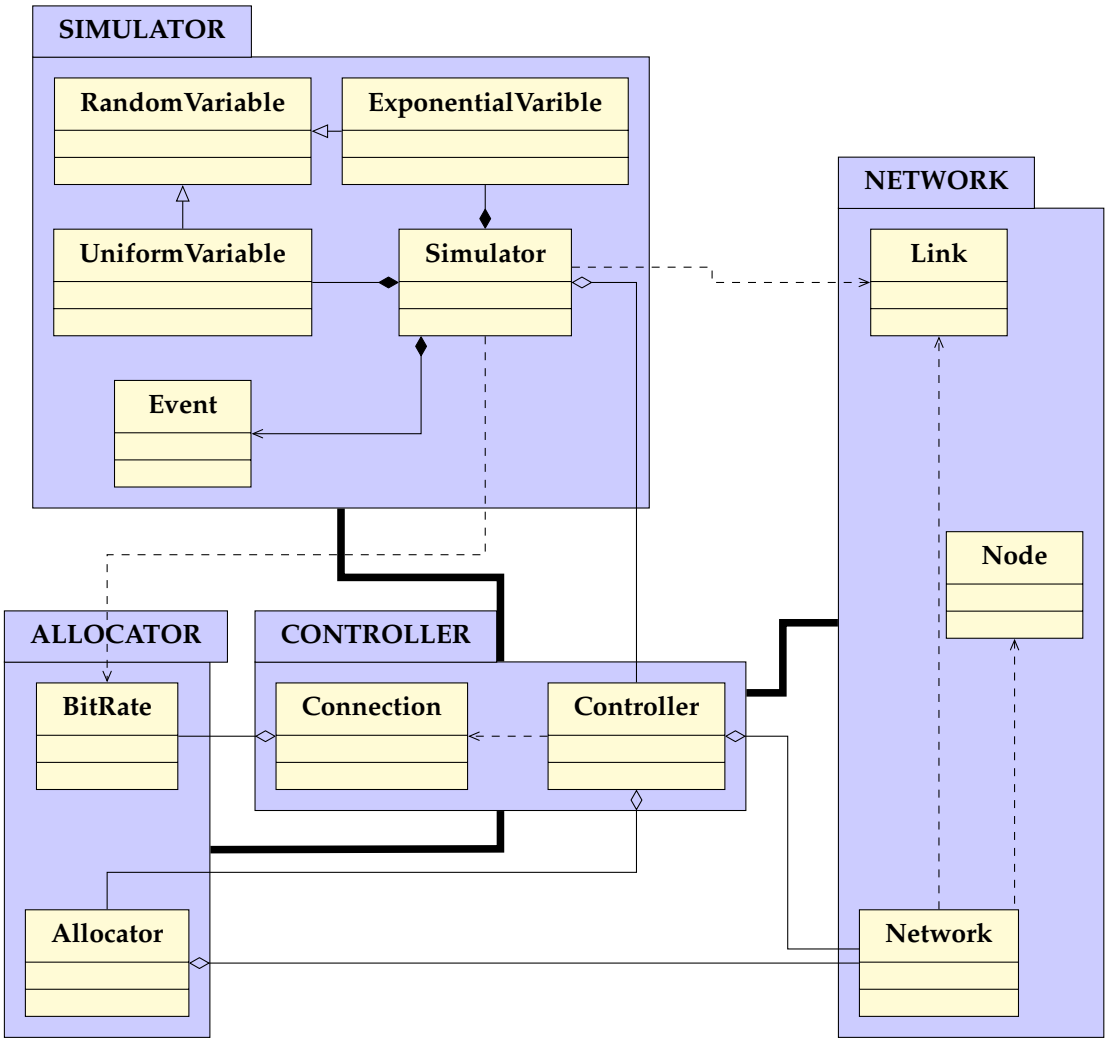


FIGURA 6.3: Clases de los componentes del simulador FNS.

Las clases del simulador que se modificaron para agregar la compatibilidad de multibanda fueron:

- *Link*: Se integró el soporte de *Links* con multiples bandas y distintos *Slots* por cada una.
- *Network*: Se agregó la capacidad de leer y almacenar redes en donde los enlaces tengan distintas capacidades de *Slots* dependiendo de las bandas que se quieran utilizar.
- *Controller*: Se habilitó la posibilidad de asignar y desasignar *Slots* en una red multibanda.
- *Bitrate*: Se incluyó el poder leer y almacenar un archivo de *Bit rate* donde se puede definir distintos alcances para cada banda dentro de una modulación.
- *Connection*: Se agregó la posibilidad de incluir conexiones de redes multibanda.
- *Simulator*: Se incluyó un constructor para soportar la creación de una simulación en un entorno multibanda.

6.3. Cambios a la clase *Link*

En la clase *Link* se agregaron atributos y métodos que se presentan en la Figura 6.4. El atributo *bands_and_slots* es del tipo *map* la cual es una estructura de datos llamada *Hash Map* que almacena pares clave-valor. En esta estructura, cada clave es única y se utiliza para acceder a su respectivo valor asociado. La implementación en C++ de esta estructura se hace mediante un árbol binario [59], por lo que la complejidad de búsqueda e inserción es de O(log(n)). En este caso, la clave es de tipo *char* debido a que las bandas están representadas por letras y el valor de cada clave es el vector de booleanos para los *slots*. Por otro lado, el atributo *number_of_bands* almacena el número de bandas que están en uso.

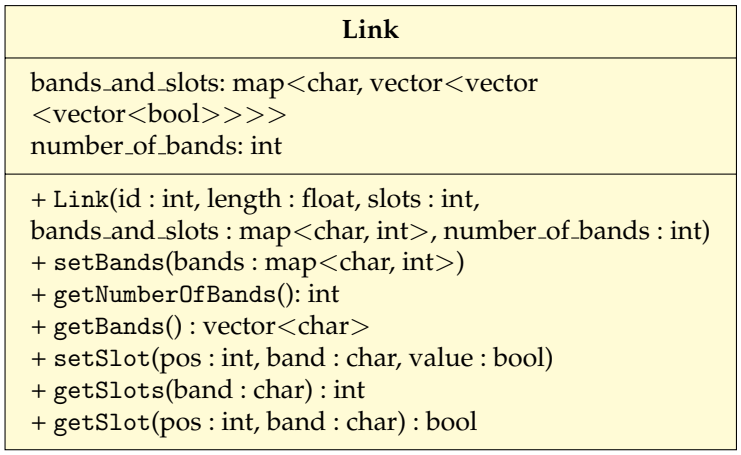


FIGURA 6.4: Diagrama UML de la clase *Link* modificada.

6.3.1. Métodos agregados

- `Link(id, length, slots, bands_and_slots, number_of_bands)`

  Constructor del objeto *Link* para un enlace multibanda.

  Parámetros:

  ***id*: Entero (*int*)**

  Identificador del objeto. Sirve para diferenciar el objeto creado de los demás.

  ***length*: Flotante (*float*)**

  La longitud de este nuevo objeto *Link*. La longitud indica la distancia entre dos nodos. La unidad de medida para la longitud es kilómetros.

  ***slots*: Entero (*int*)**

  El tamaño del vector de *Slots* de este objeto Link. El *Slot* es el canal por el que se transmite la información.

  ***bands_and_slots*: Mapa *hash* (*map<char, int>*)**

Un mapa *hash* que contiene la letra correspondiente a la banda como clave, representada por un *Char*, y el número de *Slots* que tiene como valor.

- `setBands(bands)`

Cambia el número de bandas y sus respectivos slots del enlace.

Parámetros:

***bands*: Mapa *hash* (*map<char, int>*)**

El mapa que asocia las claves *Char* de las bandas con los valores *Int* que representan los *Slots* respectivos.

- `getNumberOfBands() : Number_of_bands`

Obtiene el atributo *Number_of_bands* del objeto *Link*.

Devuelve:

***Number_of_bands*: Entero (*int*)**

Número de bandas de este objeto *Link*.

- `getBands() : bands`

Obtiene un vector de valores *Char* que representan las bandas del enlace.

Devuelve:

***bands*: Vector (*vector<char>*)**

Vector de caracteres que representan las bandas dentro de este objeto *Link*.

- `setSlot(pos, band, value)`

Establece el valor de un *Slot* específico dentro del vector de *Slots* de la banda especificada. La posición indicada debe estar dentro de los límites del vector, la banda y el valor del parámetro se asignará a el *Slot* correspondiente.

Parámetros:

***pos*: Entero (*int*)**

La posición del *Slot* dentro del vector de *Slots* mayor o igual que cero.

***band*: Carácter (*char*)**

El *char* de la banda dentro del *Link*.

***value*: Booleano (*bool*)**

El estado (activo o inactivo) que se asignará al *Slot* especificado.

- `getSlots(band) : slots`

  Obtiene el tamaño del vector de *Slots* de la banda especificada del enlace.

  Parámetros:

  ***band*: Carácter (*char*)**

  El *cChar* de la banda dentro del *Link*.

  Devuelve:

  ***slots*: Entero (*int*)**

  El tamaño del vector de *Slots*.

- `getSlot(pos, band) : status`

  Obtiene el estado de un *Slot* específico dentro del vector de *Slots* de la banda especificada.

  Parámetros:

  ***pos*: Entero (*int*)**

  La posición que desea consultar en el vector de *Slots*.

  ***band*: Carácter (*char*)**

  El *Char* de la banda dentro del *Link*.

  Devuelve:

  ***status*: Booleano (*bool*)**

  El estado del *Slot* especificado.

6.4. Cambios a la clase *Network*

Además de las modificaciones realizadas en la clase Link, también se llevaron a cabo cambios en la clase *Network* del simulador que se encuentran en la Figura 6.5. En este caso, no fue necesario agregar atributos o modificar los ya existentes. Lo anterior, es debido a que la bandas actúan a nivel de los enlaces y no de la red completa.

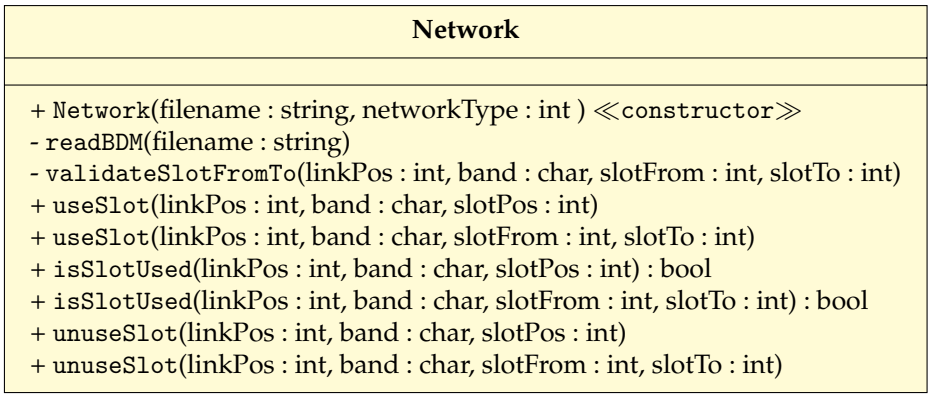


FIGURA 6.5: Diagrama UML de la clase *Network* modificada.

6.4.1. Métodos modificados

- `Network(filename, networkType)`

  Constructor del objeto *Network* para una red EON o Multibada.

  Parámetros:

  ***filename*: Cadena de caracteres (*string*)**

  Nombre del archivo *Json* que define la red.

  ***networkType*: Entero (*int*)**

  Tipo de red, p. ej. EON (igual a 1), SDM (igual a 2) y BDM (igual a 3).

6.4.2. Métodos agregados

- `readBDM(filename)`

  Lee un archivo *Json* de red multibanda.

  Parámetros:

  ***filename*: Cadena de caracteres (*string*)**

  Nombre del archivo *Json* que define la red.

- `validateSlotFromTo(linkPos, band, slotFrom, slotTo)`

  Verifica si las posiciones de enlace y *Slot* proporcionadas están dentro de los límites permitidos, y si la banda dada existe en el enlace especificado.

  Parámetros:

  ***linkPos*: Entero (*int*)**

  La posición del enlace dentro del vector de *Links*.

  ***band*: Carácter (*char*)**

  El *Char* de la banda dentro del *Link*.

  ***slotFrom*: Entero (*int*)**

  La posición inicial de los *Slots* que deben estar sin usar dentro del vector de *Slots*.

  ***slotTo*: Entero (*int*)**

  El límite antes de la posición final de los *Slots* que deben estar sin usar dentro del vector de *Slots* (desactiva hasta el *Slot* número $(\text{slotTo} - 1)$). Su valor debe ser mayor que *SlotFrom*.

- `useSlot(linkPos, band, slotPos)`

Ocupa un único *Slot* en una posición dada de una banda específica dentro de un *Link* en una posición dada dentro de la red.

Parámetros:

***linkPos*: Entero (*int*)**

La posición del enlace dentro del vector de *Links*.

***band*: Carácter (*char*)**

El *Char* de la banda dentro del *Link*.

***slotPos*: Entero (*int*)**

La posición del único *Slot* a ser usado dentro del vector de *Slots*.

- `useSlot(linkPos, band, slotFrom, slotTo)`

Activa un rango de *Slots* de una banda específica dentro de un *Link* en una posición dada dentro de la red.

Parámetros:

***linkPos*: Entero (*int*)**

La posición del enlace dentro del vector de *Links*.

***band*: Carácter (*char*)**

El *Char* de la banda dentro del *Link*.

***slotFrom*: Entero (*int*)**

La posición inicial de los *Slots* que deben estar sin usar dentro del vector de *Slots*.

***slotTo*: Entero (*int*)**

El límite antes de la posición final de los *Slots* que deben estar sin usar dentro del vector de *Slots* (desactiva hasta el *Slot* número $(\text{slotTo} - 1)$). Su valor debe ser mayor que *SlotFrom*.

- `isSlotUsed(linkPos, band, slotPos) : status`

Determina si el *Slot* en la banda especificada de un *Link* ya está siendo utilizado o no.

Parámetros:

***linkPos*: Entero (*int*)**

La posición del enlace dentro del vector de *Links*.

***band*: Carácter (*char*)**

El *Char* de la banda dentro del *Link*.

***slotPos*: Entero (*int*)**

La posición del *Slot* especificado para verificar dentro del vector de *Slots* (dentro del *Link* especificado).

Devuelve:

***status*: Booleano (*bool*)**

La condición del *Slot* especificado. Si está activo, devuelve verdadero; de lo contrario, devuelve falso.

- `isSlotUsed(linkPos, band, slotFrom, slotTo) : status`

Determina si un rango de *Slots* de la banda especificada de un *Link* ya están siendo utilizados o no.

Parámetros:

***linkPos*: Entero (*int*)**

La posición del enlace dentro del vector de *Links*.

***band*: Carácter (*char*)**

El *Char* de la banda dentro del *Link*.

***slotFrom*: Entero (*int*)**

La posición inicial de los *Slots* a ser verificados si están siendo utilizados dentro del vector de *Slots*.

***slotTo*: Entero (*int*)**

El límite antes de la posición final de los *Slots* a ser verificados si están siendo utilizados dentro del vector de *Slots* (verifica hasta el *Slot* número $(slotTo - 1)$). Su valor debe ser mayor que *SlotFrom*.

Devuelve:

***status*: Booleano (*bool*)**

La condición del rango especificado de *Slots*. Si encuentra al menos un *Slot* activado/utilizado, entonces se considera que todo el rango deseado de *Slots* está en uso y devuelve verdadero; de lo contrario, si todos están sin usar, devuelve falso.

- `unuseSlot(linkPos, band, slotPos)`

Libera una sola posición de *Slot* del vector de *Slots* de una banda dada dentro de un *Link* en una posición dada dentro de la red.

Parámetros:

***linkPos*: Entero (*int*)**

La posición del enlace dentro del vector de *Links*.

***band*: Carácter (*char*)**

El *Char* de la banda dentro del *Link*.

***slotPos*: Entero (*int*)**

La posición del único *Slot* a ser sin liberado dentro del vector de *Slots*.

- `unuseSlot(linkPos, band, slotFrom, slotTo)`

Desactiva un rango de *Slots* en el vector de *Slots* de una banda específica dentro de un *Link* en una posición dada dentro de la red.

Parámetros:

***linkPos*: Entero (*int*)**

La posición del enlace dentro del vector de *Links*.

***band*: Carácter (*char*)**

El *Char* de la banda dentro del *Link*.

***slotFrom*: Entero (*int*)**

La posición inicial de los *Slots* a ser sin usar/desactivados dentro del vector de *Slots*.

***slotTo*: Entero (*int*)**

El límite antes de la posición final de los *Slots* a ser sin usar/desactivados dentro del vector de *Slots* (desactiva hasta el *Slot* número $(slotTo - 1)$). Su valor debe ser mayor que *SlotFrom*.

Junto con la modificación e inclusión de los métodos a la clase *Network* se cambió el archivo *Json* que define a la red. Originalmente y para redes EON sigue la estructura que se muestra en la Figura 6.6. Los componentes principales del archivo son el alias, que es el nombre de la red y los *Links*, que es una lista que contiene el nodo destino (*Dst*), el *Id*, el largo (*Length*), la cantidad de *Slots* y el nodo fuente *Src* del *Link*.

```
1   "alias": "NSFNet",
2   "links": [
3       {
4           "dst": 1,
5           "id": 0,
6           "length": 1130,
7           "slots": 100,
8           "src": 0
9       },
```

FIGURA 6.6: Ejemplo de archivo *Json* original para la red.

Con el fin de lograr el correcto funcionamiento del simulador en un entorno de red multibanda, se modificó la estructura del archivo *Json* de forma que se pueda leer mediante el método `readBDM`. El atributo *slots* se convierte en un diccionario donde la llave es la letra que representa a la banda en mayúscula y el valor es la cantidad de *Slots* de cada banda correspondiente. En la Figura 6.7 se puede observar a la

izquierda el archivo original que se utiliza para redes EON, mientras que a la derecha se encuentra la red adaptada para utilizarse con multibanda. En la linea 7 del archivo modificado se puede ver que los *Slots* ahora son un diccionario donde se almacenan los *Slots* por banda.

```
1    "alias": "NSFNet",
2    "links": [
3        {
4            "dst": 1,
5            "id": 0,
6            "length": 1130,
7            "slots": 100,
8            "src": 0
9        },
```

```
1    "alias": "NSFNet",
2    "links": [
3        {
4            "dst": 1,
5            "id": 0,
6            "length": 1130,
7            "slots": {"C":100, "L":70, "S":50, "E":40},
8            "src": 0
9        },
```

FIGURA 6.7: Comparación Archivo *Json* original con el modificado para la red.

Con estos cambios es posible asignar cantidades especificas de *Slots* a cada una de las diferentes bandas que se utilicen en la simulación, de forma que al momento de crear el algoritmo se pueda acceder de forma independiente a la cantidad *Slots* según la banda que se quiera utilizar.

6.5. Cambios a la clase *Controller*

La Figura 6.8 ilustra el diagrama UML de la clase *Controller* con los cambios realizados. Se agregaron los atributos *bands*, el cual almacena un vector de caracteres que representan las bandas utilizadas en la red, y el atributo *bandsSlots* del tipo *map* que asocia cada banda con una matriz de enteros, que a su vez representa los *Slots* en cada enlace y su estado, al igual que en la clase *Connection*.

Controller
- bands: vector<char> - bandsSlots: map<char,vector<vector<int>>>
+ setAssignBDM() + setUnassignBDM() - unassignConnectionBDM(idConnection : long long, time : double)

FIGURA 6.8: Diagrama UML de la clase *Controller* modificada.

6.5.1. Métodos agregados

- `setAssignBDM()`

  Se encarga de configurar el método de asignación de una conexión.

- `setUnassignBDM()`

  Se encarga de configurar el método de des-asignación de una conexión.

- `setUnassignBDM()`

  Es utilizado para des-asignar una conexión en una red multibanda.

6.6. Cambios a la clase *Bitrate*

En la Figura 6.9 se ven los cambios a la clase *Bitrate*, donde se agregaron los atributos *Bands*, *slotsPerBand* y *reachPerBand*. Estos atributos se tratan de vectores que almacenan otros vectores con los valores de la banda correspondiente, los cuales son la letra, los *slots* y el *reach* respectivamente.

Bitrate
- bands: vector<vector<char>> - slotsPerBand: vector<vector<int>> - reachPerBand: vector<vector<double>>
+ `BitRate`(&bitRate : const BitRate) ≪`constructor`≫ + `readBitRateFileBDM`(fileName : string) : vector<BitRate> + `selectBitrateMethod`(fileName : string, networkType : int) : vector<BitRate> + `addModulation`(modulation : string, slots : int, reach : double, band : vector<char>, slotsPerBand : vector<int>, reachPerBand : vector<double>) + `getBand`(modulation : int, pos : int) : char + `getNumberOfSlotsPerBand`(modulation : int, pos : int) : int + `getReachPerBand`(modulation : int, pos : int) : double + `getNumberOfBands`() : int

FIGURA 6.9: Diagrama UML de la clase *Bitrate* modificada.

6.6.1. Métodos modificados

- `BitRate(bitRate)`

  Constructor de copia que inicializa un nuevo objeto *BitRate* utilizando otro objeto de la misma clase.

  Parámetros:

  ***bitRate*: Objeto *Bitrate* constante (*const BitRate*)**

  Objeto a copiar.

6.6.2. Métodos agregados

- `readBitRateFileBDM(fileName) : BitrateVector`

  Lee un archivo *Json* y extrae su información para llenar un vector de objetos *BitRate*.

  Parámetros:

***filename*: Cadena de caracteres (*string*)**

Ubicación del archivo *Json*, como una ruta relativa. Por ejemplo: *"../bitrate/bitrate.json"*.

Devuelve:

***BitrateVector*: Vector (*vector<BitRate>*)**

Vector con los objetos *BitRate* del archivo *Json*.

- `selectBitrateMethod(fileName, networkType) : BitrateVector`

Lee un archivo *Json* y extrae su información para llenar un vector de objetos *BitRate*.

Parámetros:

***filename*: Cadena de caracteres (*string*)**

Ubicación del archivo *Json*, como una ruta relativa. Por ejemplo: *"../bitrate/bitrate.json"*.

***networkType*: Entero (*int*)**

Tipo de red, p. ej. EON (igual a 1), SDM (igual a 2) y BDM (igual a 3).

Devuelve:

***BitrateVector*: Vector (*vector<BitRate>*)**

Vector con los objetos *BitRate* del archivo *Json*.

- `addModulation(modulation, slots, reach, band, slotsPerBand, reachPerBand)`

Agrega una nueva modulación al objeto *BitRate* actual, proporcionando su nombre, cantidad de *slots* y distancia de alcance.

Parámetros:

***modulation*: Cadena de caracteres (*string*)**

El nombre de la modulación deseada.

***slots*: Entero (*int*)**

El número de *Slots* que se asignarán para la modulación.

***reach*: Número de doble precisión (*double*)**

La distancia máxima de la suma total de bandas que se alcanzará para esta modulación.

***band*: Vector (*vector<char>*)**

El vector de caracteres que representa las bandas dentro de la tasa de bits.

***slotsPerBand*: Vector (*vector<int>*)**

El vector de enteros que representa los *Slots* de las bandas dentro de la tasa de bits, por ejemplo: $std :: vector < int > slotsPerBand = 100, 200;$.

***reachPerBand*: Vector (*vector<double>*)**

El vector de *double* que representa el alcance de las bandas dentro de la tasa de bits, por ejemplo: $std :: vector < double > reachPerBand = 2758, 2762;$.

- `getBand(modulation, pos) : band`

Obtiene el carácter de la banda en una posición dada (argumento) dentro de la modulación.

Parámetros:

***modulation*: Entero (*int*)**

El índice de posición para la modulación deseada.

***pos*: Entero (*int*)**

El índice de posición para la banda deseada.

Devuelve:

band*: Carácter (*char)

El carácter de la banda de la modulación deseada en la posición deseada.

- `getNumberOfSlotsPerBand(modulation, pos) : numberSlots`

Obtiene el número de *Slots* en una posición dada (argumento) dentro de una banda del vector de *Slots* de las modulaciones.

Parámetros:

***modulation*: Entero (*int*)**

El índice de posición para la modulación deseada.

***pos*: Entero (*int*)**

El índice de posición para la banda deseada.

Devuelve:

numberSlots*: Entero (*int)

El número de *Slots* de la banda de la modulación deseada.

- `getReachPerBand(modulation, pos) : reach`

Obtiene la distancia máxima de alcance en una banda dada (argumento) dentro del vector de alcances de las modulaciones.

Parámetros:

***modulation*: Entero (*int*)**

El índice de posición para la modulación deseada.

***pos*: Entero (*int*)**

El índice de posición para la banda deseada.

Devuelve:

***reach*: Número de doble precisión (*double*)**

El *reach* de la banda de la modulación deseada.

- `getNumberOfBands() : numberOfBands`

Obtiene el número de bandas disponibles en el objeto *BitRate* actual.

Devuelve:

***numberOfBands*: Entero (*int*)**

El número de bandas en el objeto *BitRate* actual.

Al igual que con la clase *Network*, se utiliza un archivo *Json* para definir el *Bit rate* el cual se tiene que cambiar para poder soportar distintas bandas. Se tiene en cuenta que cada banda tiene un *Reach* distinto según el formato de modulación utilizado. Este archivo tiene la estructura que se muestra en la Figura 6.10. La llave principal se trata de la tasa de bits, que tiene como valor una lista donde se colocan las modulaciones. Por su parte, cada modulación tiene sus respectivos *Slots* y el *Reach*.

```
"10": [
    {
        "BPSK": {
            "slots": 1,
            "reach": 5520
        }
    }
],
"40": [
    {
        "BPSK": {
            "slots": 4,
            "reach": 5520
        }
```

FIGURA 6.10: Ejemplo de archivo *Json* original para el *Bit rate*.

Las modificaciones realizadas son que se ha agregado una capa adicional entre el formato de modulación y la cantidad de *Slots* y el *Reach*. Esta nueva capa consiste en una lista de diccionarios que contienen una llave, representada por una letra, que identifica a la banda correspondiente, mientras que su valor asociado representa la cantidad de *Slots* y el *Reach* que tiene la banda específicamente, recordando que dependiendo de la banda a utilizar se ve afectado el alcance de la modulación.

La Figura 6.11 presenta una comparación entre el archivo de *Bit rate* original, ubicado a la izquierda, y el archivo modificado para una red multibanda, representado a la derecha. En este contexto, se han empleado valores de ejemplo con el propósito de ilustrar un caso específico, mostrando que en el nuevo archivo de *Bit rates* la inclusión de las bandas por cada formato de modulación.

En este sentido, la inclusión de esta capa adicional en el archivo de *Bit rate* permite una mayor adaptabilidad al definir los *Bit rates* en una red multibanda, debido a la capacidad de definir por cada banda la cantidad de *Slots* y *Reach*.

```
"10": [
    {
        "BPSK": {
            "slots": 1,
            "reach": 5520
        }
    }
],
"40": [
    {
        "BPSK": {
            "slots": 4,
            "reach": 5520
        }
```

```
"10": [
    {
        "BPSK": [
            {
                "C": {
                    "slots": 1,
                    "reach": 13000
                }
            },
            {
                "L": {
                    "slots": 1,
                    "reach": 14400
                }
```

FIGURA 6.11: Comparación Archivo *Json* original con el modificado para el *Bit rate*.

6.7. Cambios a la clase *Connection*

En la Figura 6.12 se presenta el diagrama UML de la clase *Connection* modificada. Se agregó el atributo *bands* el cual almacena un vector de caracteres que representan las bandas utilizadas en la red, y el atributo *bandsSlots* que un *map* que asocia cada banda con una matriz de enteros, que a su vez representa los *slots* en cada enlace y su estado.

Connection
- bands: vector<char> - bandsSlots: map<char,vector<vector<int>>>
+ addLink(idLink : int, band : char, slots: vector<int>) + addLink(idLink : int, band : char, fromSlot : int, toSlot : int) + getBands() : vector<char>

FIGURA 6.12: Diagrama UML de la clase *Connection* modificada.

6.7.1. Métodos agregados

- `addLink(idLink, band, slots)`

  Agrega un nuevo enlace al objeto de conexión. El *Id* del enlace se agrega al vector de enlaces, y los *Slots* al vector de *Slots*.

  Parámetros:

***idLink*: Entero (*int*)**

El *Id* del nuevo enlace agregado al objeto de conexión.

***band*: Carácter (*char*)**

El *Char* de la banda dentro del *Link*.

***slots*: Vector (*vector<int>*)**

Vector que contiene la posición de los *Slots*.

- `addLink(idLink, band, fromSlot, toSlot)`

Agrega un nuevo enlace al objeto de conexión. El *Id* del enlace se agrega al vector de enlaces, y los *Slots* en el rango desde *FromSlot* hasta *ToSlot* se agregan al vector de *Slots*.

Parámetros:

***idLink*: Entero (*int*)**

El *Id* del nuevo enlace agregado al objeto de conexión.

***band*: Carácter (*char*)**

El *Char* de la banda dentro del *Link*.

***fromSlot*: Entero (*int*)**

La posición del primer *Slot* a ser utilizado en el modo.

***toSlot*: Entero (*int*)**

La posición del último *Slot* a ser utilizado en el modo.

- `getBands() : bands`

Obtiene el vector de caracteres de las bandas disponibles en el objeto actual.

Devuelve:

***bands*: Vector (*vector<char>*)**

Vector de caracteres de las bandas.

6.8. Cambios a la clase *Simulator*

La última clase que se modificó en el simulador fue la clase *Simulator* que se puede ver en la Figura 6.13. En donde se modificó un constructor para permitir la utilización de multibanda. El constructor modificado, acepta nombres de archivo para la red, la ruta y el *Bit rate*, junto con el tipo de red.

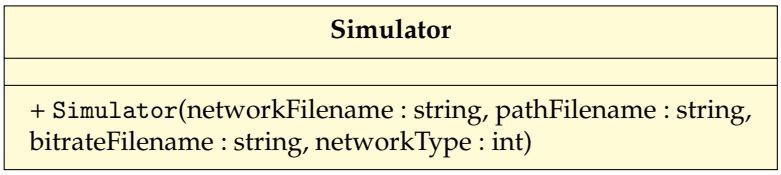


FIGURA 6.13: Diagrama UML de la clase *Simulator* modificada.

6.8.1. Métodos modificados

- `Simulator(networkFilename, pathFilename, bitrateFilename, networkType)`

  Construye el objeto *Simulator* a partir de dos archivos *Json*. Estos archivos contienen la configuración de la red y las rutas.

  Parámetros:

  ***networkFilename*: Cadena de caracteres (*string*)**

  Fuente del archivo de red. Este archivo es la configuración de la red e incluye información de los nodos (identificación, destino, origen, longitud, *Slots*).

  ***pathFilename*: Cadena de caracteres (*string*)**

  Este archivo contiene las rutas entre nodos.

  ***bitrateFilename*: Cadena de caracteres (*string*)**

  Este archivo es la configuración de la red e incluye información de los nodos (identificación, destino, origen, longitud, *Slots*).

  ***networkFilename*: Entero (*int*)**

  Define el tipo de red, p. ej. EON (igual a 1), SDM (igual a 2) y BDM (igual a 3).

6.9. *Testing* del módulo

Para comprobar el correcto funcionamiento de la implementación del módulo se crearon pruebas unitarias, las cuales son una práctica fundamental en el desarrollo de software, ya que permiten identificar y corregir posibles errores o fallos. Se crearon pruebas unitarias para cada clase intervenida: *Bitrate*, *Link*, *Network*, *Controller*, *Connection* y *Simulator*, las cuales constaron de poner a prueba los métodos implementados junto a comprobar que el resto de métodos no se vieran influenciados. Estas pruebas fueron diseñadas para asegurarse de que funcionaran correctamente en diferentes escenarios y casos de uso, utilizando diferentes parámetros y provocando fallos intencionados.

En este contexto, se utilizó *Catch2* para las pruebas unitarias y CMake como una herramienta para ejecutarlas. *Catch2* es un marco de pruebas para C++ que se distribuye principalmente como un solo archivo de cabecera, aunque algunas extensiones pueden requerir archivos de cabecera adicionales. *CMake* es una herramienta de

código abierto que facilita la construcción, prueba y empaquetado de software en diversos entornos. Su utilidad radica en su capacidad para generar automáticamente archivos de configuración y construcción, lo que simplifica el proceso de compilación y asegura una mayor portabilidad del software.

Al emplear *CMake* y *Catch2* para la ejecución de las pruebas, se logró una automatización del proceso, lo que agilizó la validación y verificación de las nuevas funcionalidades implementadas. Esto se debe a que *CMake* y *Catch2* permiten definir los pasos necesarios para compilar, enlazar y ejecutar las pruebas unitarias, así como gestionar las dependencias del proyecto de manera flexible.

De forma sintetizada, la implementación del diseño se basó en el código oficial de la biblioteca, al cual se le realizó un *Fork* y se creó una nueva rama para el desarrollo donde se agregaron y modificaron los métodos necesarios. Se diseñaron y ejecutaron pruebas unitarias utilizando *Catch2* y *CMake* como herramienta de automatización. Este enfoque de desarrollo riguroso y sistemático aseguró la integridad y la calidad del código modificado, y permitió la incorporación efectiva de las nuevas funcionalidades en la biblioteca. Además, se utilizaron *pipelines* de integración continua y entrega continua (*CI/CD*) para automatizar el proceso de compilación, prueba y despliegue del código modificado.

Para validar que el módulo implementado simule correctamente una red multibanda y poder ver las diferencias con otros métodos de simulación, se programaron 6 distintos códigos en total de algoritmos de asignación de recursos para ambientes multibanda utilizando el simulador con el módulo implementado y sin el módulo implementado. La comparación se encuentra en la Tabla 6.1 y sigue las siguientes definiciones:

- V1: Variante 1 del algoritmo propuesto en [60] con el simulador *Flex Net Sim* (FNS) con y sin el módulo multibanda incorporado.
- V2: Variante 2 del algoritmo propuesto en [60] con el simulador FNS con y sin el módulo multibanda incorporado.
- V3: Variante 3 del algoritmo propuesto en [60] con el simulador FNS con y sin el módulo multibanda incorporado.
- MB: Resultado del código con el módulo multibanda incorporado.
- No MB: Resultado del código sin el módulo multibanda incorporado.
- Líneas código: Cantidad de líneas del código sin considerar espacios, variables globales ni la sección correspondiente al *main*.
- Líneas *Bit rate*: Cantidad de líneas del archivo json que contiene los *Bit rates*.
- Palabras: Cantidad de palabras del código.
- Complejidad: Nivel de complejidad del código, que se divide en tres niveles:
    - Bajo: Implica que es fácil modificar el código para agregar funciones, cambiar parámetros o probar otras implementaciones, y se pueden acceder a una gran cantidad de parámetros a través de macros.
    - Medio: Significa que se puede modificar el código para agregar funciones, cambiar parámetros o probar otras implementaciones sin necesidad de realizar cambios significativos en el código o en el simulador.

- Alto: Indica que es difícil acceder a parámetros de la simulación y requiere un alto conocimiento del simulador para obtener los datos deseados.

	Líneas código		Líneas *Bit rate*		Palabras		Complejidad	
	MB	No MB	MB	No MB	MB	No MB	MB	No MB
V1	95	262	572	417	372	960	Baja	Alta
V2	102	355	572	417	402	1373	Baja	Alta
V3	95	270	572	417	376	976	Baja	Alta

TABLA 6.1: Características de los códigos de simulación multibanda.

Según la Tabla 6.1, es notable que las variantes del algoritmo con el módulo multibanda tienen significativamente menos líneas de código y palabras que las variantes sin el módulo. Esto sugiere que el módulo multibanda permite una implementación más eficiente del algoritmo, lo que puede resultar en un tiempo de desarrollo más corto y una mayor facilidad de mantenimiento en caso de que el código requiera modificaciones. En cuanto a las líneas de *Bit rate* se puede ver que el formato de *Json* propuesto utiliza más líneas en comparación del formato original. Sin embargo, hay que tener en cuenta que el nuevo formato permite una definición más precisa de *Slots* y *Reach* así como un mejor acceso a estos valores dentro de la simulación.

Además, la complejidad de los códigos con el módulo es baja, lo que implica que es fácil modificar el código para agregar funciones, cambiar parámetros o probar otras implementaciones.

Por otro lado, los códigos sin el módulo tienen una complejidad alta. Esto indica que es difícil acceder a parámetros de la simulación y requiere un alto conocimiento del simulador para obtener los datos deseados. Esta barrera puede ralentizar el proceso de desarrollo y limitar la capacidad para adaptarse a diferentes escenarios de simulación.

De lo anterior, se puede decir que el simulador con el módulo multibanda proporciona una implementación más eficiente y manejable del algoritmo propuesto en comparación con el simulador sin el módulo. Esto facilita la experimentación y la adaptabilidad a diferentes escenarios de simulación. Por lo tanto, se puede concluir que el simulador con el módulo multibanda es una herramienta que entrega más facilidades para la simulación de redes ópticas en ambientes multibanda.

Para realizar la validación de resultados, se hizo una comparación entre la biblioteca con y sin el módulo multibanda implementado. Se obtuvieron los resultados de las variante 1, variante 2 y variante 3 de los algoritmos presentados en [60]. Para esto, se obtuvo el *Bandwidth Blocking Probability* o BBP, que es la relación entre el ancho de banda bloqueado y el ancho de banda total solicitado [61]. Esta relación se puede ver en la Ecuación 6.1 donde nb es el número de *Bit rates* utilizados, MW_b es el peso medio asignado a la tasa de *Bits* b, BT_b es la cantidad total de tráfico en la tasa de bits b y BC_b es la cantidad de tráfico bloqueado en la tasa de bits b. El BBP se obtuvo para diferentes *Offered Traffic Load* que es la demanda de conexiones que se generan por las fuentes de tráfico y que se expresa en *Erlangs* los que equivalen al número promedio de conexiones simultáneas que se requieren en un intervalo de tiempo dado. Se puede ver en la Ecuación 6.2 la forma en la que se calcula la carga de tráfico. Donde ρ es la carga de tráfico en *Erlangs*, λ es el número de conexiones solicitadas por unidad de tiempo y μ es la duración media de una conexión.

$$BBP = \frac{\sum_0^{nb} MW_b * (\frac{BC_b}{BT_b}))}{\sum_0^{nb} MW_b} \tag{6.1}$$

$$\rho = \frac{\lambda}{\mu} \tag{6.2}$$

En la Figura 6.14 se puede ver los resultados que se obtuvieron de las variantes programadas en el simulador sin el módulo implementado. El eje vertical representa el BBP obtenido en la simulación mientras que el eje horizontal es el *Offered Traffic Load* utilizado.

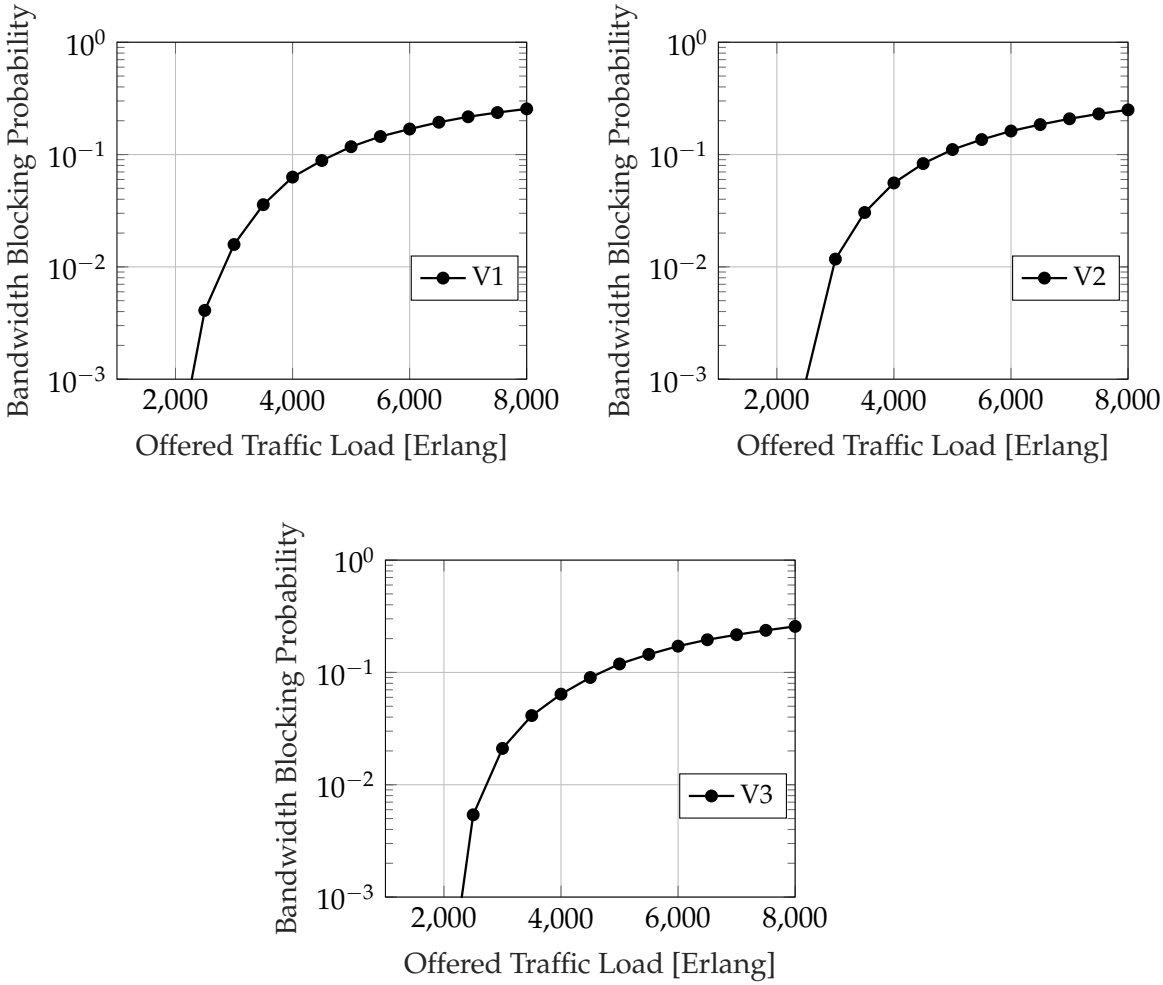


FIGURA 6.14: Resultados de V1, V2 y V3 sin módulo multibanda utilizando las bandas C+L+S+E.

Por otro lado, en la Figura 6.15 se puede ver los resultados que se obtuvieron de las variantes programadas en el simulador sin el módulo implementado. En las figuras podemos ver una tendencia a subir el BBP según aumenta el tráfico, formando una curva que cambia levemente en cada variante.

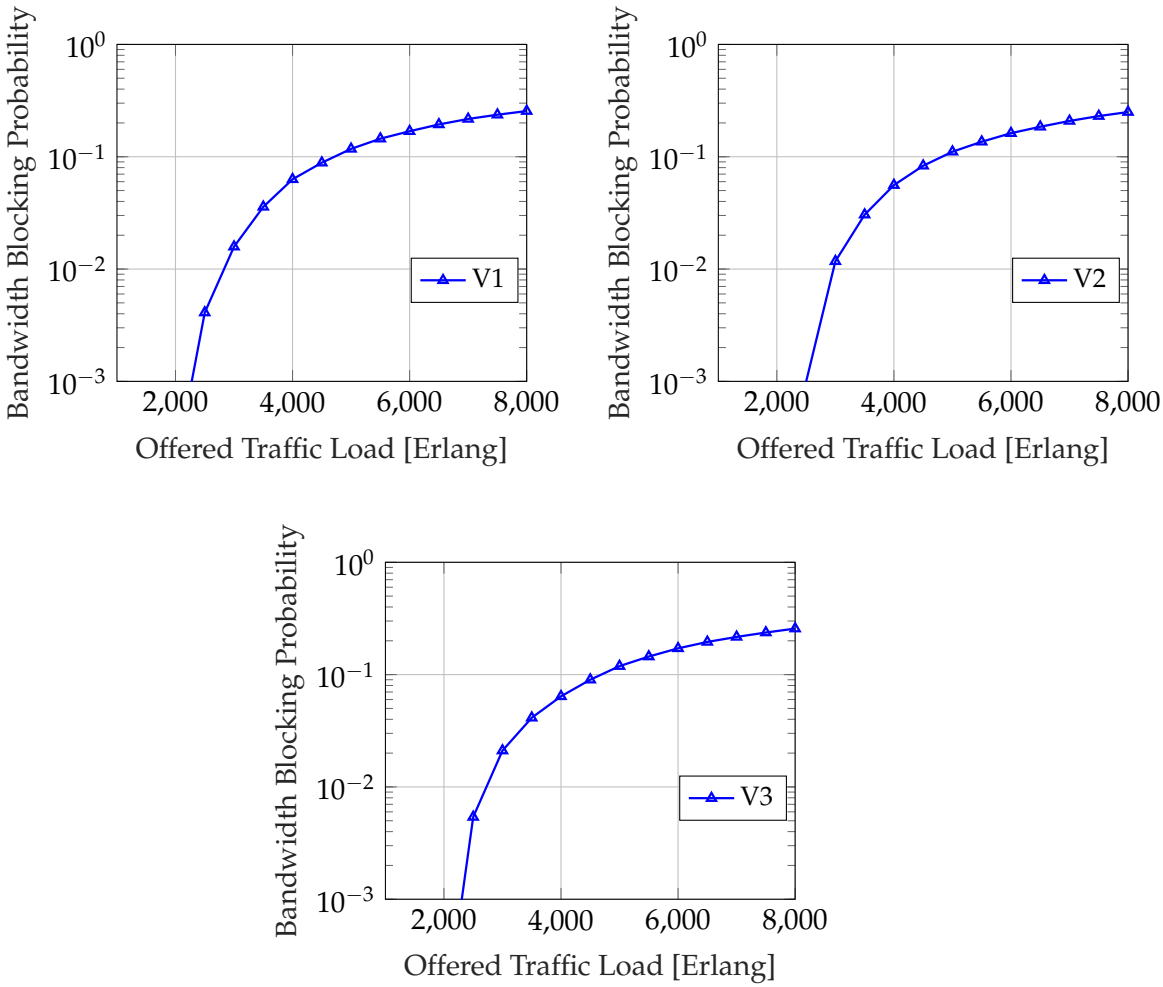


FIGURA 6.15: Resultados de V1, V2 y V3 con módulo multibanda utilizando las bandas C+L+S+E.

Para poder realizar un análisis de estos resultados se decidió sobreponer los gráficos de cada variante respectiva, esto se realiza en la Figura 6.16.

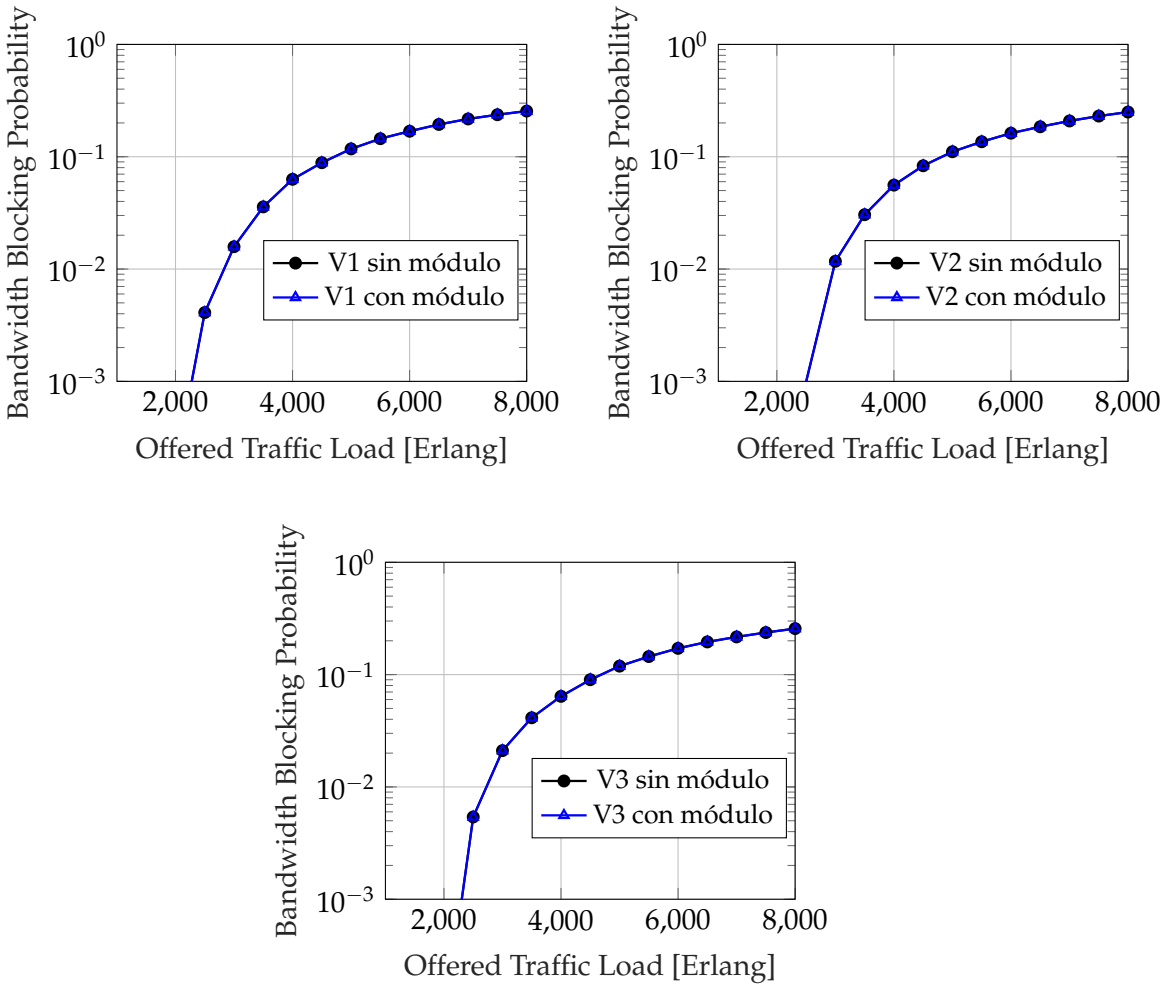


FIGURA 6.16: Comparación entre variantes con y sin módulo implementado.

Como se puede observar en la Figura 6.16, ambas líneas coinciden perfectamente en sus resultados para todas las variantes. Esta concordancia se debe al hecho de que se utilizó la misma semilla en ambas simulaciones, lo que se refiere a que los número aleatorios generados en la simulación, como las conexiones que llegan o salen, son exactamente las mismas en ambos casos. Este resultado confirma la precisión del módulo implementado en la realización de las simulaciones, ya que las condiciones iniciales y el proceso de simulación son consistentes en las variantes de los algoritmos, lo que valida la reproducibilidad de los resultados.

El módulo implementado para el simulador se encuentra disponible en el repositorio oficial de *GitLab* de la biblioteca del simulador *Flex Net Sim*[1]. En este repositorio, además, se proporciona un ejemplo de cómo utilizar las nuevas características, junto con su respectiva documentación.

[1] `https://gitlab.com/DaniloBorquez/flex-net-sim/-/tree/master`

Capítulo 7

Conclusiones

Este trabajo ha abordado el problema de la simulación para la asignación de recursos en redes ópticas elásticas multibanda. Este problema implica la identificación de las rutas óptimas, la asignación del espectro y la determinación de los niveles de modulación adecuados para satisfacer las demandas de ancho de banda en una red que opera con diferentes bandas de longitud de onda.

Se ha realizado una revisión del estado del arte en simuladores de redes ópticas. Específicamente, se analizó en profundidad sus características distintivas, ventajas y limitaciones de los simuladores *FlexGridSim*, *CEONS*, *OMNeT++*, *ONS*, *ElasticO++* y *Flex Net Sim*. A través de este análisis, se ha identificado una carencia de herramientas disponibles para simular problemas en entornos de red multibanda sin modificar herramientas ya existentes. Por otro lado, se encontró una falta de flexibilidad y facilidad para codificar algoritmos de asignación de recursos en estos ambientes.

En respuesta a estas limitaciones, se desarrolló una nueva herramienta de simulación en C++ que facilita la codificación de algoritmos de asignación de recursos en redes ópticas elásticas multibanda, reduciendo significativamente la cantidad de líneas de código necesarias para codificar un algoritmo, además de permitir una implementación más eficiente gracias al uso de Macros para redes multibanda. Se describió detalladamente la metodología que se siguió para el diseño, implementación y evaluación de esta herramienta. Además, se definieron los resultados esperados y los criterios para considerarlos un éxito.

Mirando hacia el futuro, se propone llevar a cabo las siguientes actividades:

- Implementar la compatibilidad de multibanda con otro tipo de redes, como las redes Multinúcleo. Esto implica desarrollar y adaptar métodos y clases para garantizar que la herramienta pueda interactuar eficientemente con estas redes.
- Mantener la biblioteca actualizada para satisfacer las nuevas necesidades que puedan surgir a medida que se investigan nuevos algoritmos.Incluyendo la adición de macros para acceder a nuevos parámetros del simulador y el desarrollo de nuevos métodos que puedan requerir estos algoritmos.
- Evaluar el impacto y la utilidad de la herramienta en la comunidad científica investigando trabajos donde se emplee esta herramienta. Realizar métricas y hacer ajustes según sea necesario. Para evaluar el impacto, se realizará un seguimiento de las citas de los trabajos que utilizan esta herramienta. Las métricas de uso, como el número de descargas, usuarios activos y frecuencia de uso, también se recogerán para evaluar la utilidad de la herramienta. Los

ajustes se realizarán en función de los comentarios de los usuarios y las métricas recogidas para mejorar continuamente la herramienta y su utilidad para la comunidad científica.

- Trabajar en la comparación de tiempos de simulación con futuras herramientas que puedan existir para simular redes ópticas multibanda. Esto permitirá evaluar la eficiencia y rendimiento de la herramienta en comparación con las soluciones que puedan surgir en el futuro.

Se presenta un avance hacia la resolución del problema de las herramientas disponibles para simular la asignación de recursos en redes ópticas elásticas multibanda. El objetivo principal de este esfuerzo es mejorar las herramientas disponibles para los investigadores que trabajan en este campo. Al proporcionar una herramienta más robusta para la simulación asignación de recursos, se espera facilitar la tarea de los investigadores para abordar eficazmente el problema de crear algoritmos de asignación de recursos.

Bibliografía

[1] J. W. Carey, "Technology and ideology: The case of the telegraph," Prospects, vol. 8, p. 303–325, 1983.

[2] M. A. Matin, Cellular Telephony System. Cham: Springer International Publishing, 2018, pp. 89–106. [Online]. Available: https://doi.org/10.1007/978-3-319-70129-5_6

[3] G. O'Regan, A Brief History of Computing, 1st ed. Springer Publishing Company, Incorporated, 2008, ch. 17.

[4] S. Charp, "Trends in using computers in education," Proceedings of the 5th Jerusalem Conference on Information Technology, 1990. 'Next Decade in Information Technology', pp. 644–646, 1990. [Online]. Available: https://api.semanticscholar.org/CorpusID:10078591

[5] A. Yourtchenko and F. Gont, "On the implementation of tcp urgent data," in Computer Science, Business, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:202778311

[6] G. O'Regan, A Brief History of Computing, 1st ed. Springer Publishing Company, Incorporated, 2008, ch. 18.

[7] O. Spaniol, "High speed local area networks — what, why, when and how: Planning, installation and first experiences of a hslan in an heterogeneous environment," in High-Capacity Local and Metropolitan Area Networks, G. Pujolle, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 73–82.

[8] M. Yang, Y. Sun, X. Pan, H. Wan, and X. Lu, "Development and prospect of twisted pair cables," IOP Conference Series: Earth and Environmental Science, vol. 170, no. 4, p. 042126, jul 2018. [Online]. Available: https://dx.doi.org/10.1088/1755-1315/170/4/042126

[9] M. Backmann, D. C. Pfeiler, and A. Wassmuth, "Crosstalk model for pair-shielded data cables," in Physics, 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:201081210

[10] E. L. Davis, "Fast ethernet: 100basetx and 100baset4 network interface adaptor architectures," in Emerging High-Speed Local-Area Networks and Wide-Area Networks, vol. 2608. SPIE, 1995, pp. 37–41.

[11] R. Pax, "Repeater between home systems coaxial cable and twisted pair media," in Proceedings of International Conference on Consumer Electronics, 1995, pp. 404–.

[12] M. J. O'Mahony, C. Politi, D. Klonidis, R. Nejabati, and D. Simeonidou, "Future optical networks," Journal of Lightwave Technology, vol. 24, no. 12, pp. 4684–4696, 2006.

[13] J. Engebretson, "Bandwidth demand forecast: 300 mbps will be enough for most households to 2031," Strategy Analytics, May 2021. [Online]. Available: https://phys.org/news/2023-07-korean-team-room-temperature-ambient-pressure-superconductor.html

[14] Cisco, "Cisco Annual Internet Report (2018–2023) White Paper," https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html, Cisco, Tech. Rep., 2020.

[15] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of iot: Applications, challenges, and opportunities with china perspective," IEEE Internet of Things Journal, vol. 1, no. 4, pp. 349–359, 2014.

[16] K. Bilal and A. Erbad, "Edge computing for interactive media and video streaming," in 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), 2017, pp. 68–73.

[17] Cisco, "Cisco Visual Networking Index -(VNI) (2017-2022," http://ciscovnilatam.com/america-latina/.

[18] G. Charlet and S. Bigo, "Upgrading wdm submarine systems to 40-gbit/s channel bitrate," Proceedings of the IEEE, vol. 94, pp. 935 – 951, 06 2006.

[19] A. Tanenbaum, Redes de computadoras. Editorial Alhambra S. A. (SP), 2003, ch. 2. [Online]. Available: https://books.google.cl/books?id=WWD-4oF9hjEC

[20] J. D. Downie, J. Hurley, D. Pikula, S. Ten, and C. Towery, "Study of edfa and raman system transmission reach with 256 gb/s pm-16qam signals over three optical fibers with 100 km spans," Opt. Express, vol. 21, no. 14, pp. 17 372–17 378, Jul 2013. [Online]. Available: http://opg.optica.org/oe/abstract.cfm?URI=oe-21-14-17372

[21] J. K. Perin, J. M. Kahn, J. D. Downie, J. Hurley, and K. Bennett, "Importance of amplifier physics in maximizing the capacity of submarine links," Journal of Lightwave Technology, vol. 37, no. 9, pp. 2076–2085, 2019.

[22] T. Horiguchi and M. Tateda, "Optical-fiber-attenuation investigation using stimulated brillouin scattering between a pulse and a continuous wave," Opt. Lett., vol. 14, no. 8, pp. 408–410, 04 1989. [Online]. Available: http://opg.optica.org/ol/abstract.cfm?URI=ol-14-8-408

[23] G. Shen and R. S. Tucker, "Translucent optical networks: the way forward [topics in optical communications]," IEEE Communications Magazine, vol. 45, no. 2, pp. 48–54, 2007.

[24] M. J. Adams and I. D. Henning, Propagation in Monomode Fibres. Boston, MA: Springer US, 1990, pp. 21–33. [Online]. Available: https://doi.org/10.1007/978-1-4899-3710-0_3

[25] J. M. Simmons, Optical Network Design and Planning, 1st ed. Springer Publishing Company, Incorporated, 2008, ch. 5.

[26] International Telecommunication Union, "Spectral grids for wdm applications: Dwdm frequency grid," ITU-T, Recommendation ITU-T G.694.1, October 2020. [Online]. Available: https://www.itu.int/rec/T-REC-G.694.1-202010-I/es

[27] N. Sambo, A. Ferrari, A. Napoli, N. Costa, J. Pedro, B. Sommerkorn-Krombholz, P. Castoldi, and V. Curri, "Provisioning in multi-band optical networks," Journal of Lightwave Technology, vol. 38, no. 9, pp. 2598–2605, 2020.

[28] A. Ferrari, A. Napoli, J. K. Fischer, N. Costa, A. D'Amico, J. Pedro, W. Forysiak, E. Pincemin, A. Lord, A. Stavdas, J. P. F.-P. Gimenez, G. Roelkens, N. Calabretta, S. Abrate, B. Sommerkorn-Krombholz, and V. Curri, "Assessment on the achievable throughput of multi-band itu-t g.652.d fiber transmission systems," Journal of Lightwave Technology, vol. 38, no. 16, pp. 4279–4291, 2020.

[29] D. Rafique and A. D. Ellis, "Nonlinear penalties in long-haul optical networks employing dynamic transponders," Opt. Express, vol. 19, no. 10, pp. 9044–9049, May 2011. [Online]. Available: https://opg.optica.org/oe/abstract.cfm?URI=oe-19-10-9044

[30] A. Mitra, D. Semrau, N. Gahlawat, A. Srivastava, P. Bayvel, and A. Lord, "Effect of channel launch power on fill margin in c+l band elastic optical networks," Journal of Lightwave Technology, vol. 38, no. 5, pp. 1032–1040, 2020.

[31] C. Zhang, X. Liu, J. Li, A. Zhang, H. Liu, L. Feng, K. Lv, S. Liao, Z. Chang, and J. Zhang, "Optical layer impairments and their mitigation in c+l+s+e+o multi-band optical networks with g.652 and loss-minimized g.654 fibers," Journal of Lightwave Technology, vol. 40, no. 11, pp. 3415–3424, 2022.

[32] C. V. Saradhi and S. Subramaniam, "Physical layer impairment aware routing (pliar) in wdm optical networks: issues and challenges," IEEE Communications Surveys & Tutorials, vol. 11, no. 4, pp. 109 –130, 2009.

[33] M. Klinkowski, P. Lechowicz, and K. Walkowiak, "Survey of resource allocation schemes and algorithms in spectrally-spatially flexible optical networking," Optical Switching and Networking, vol. 27, pp. 58–78, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S157342771730053X

[34] B. C. Chatterjee, N. Sarma, and E. Oki, "Routing and spectrum allocation in elastic optical networks: A tutorial," IEEE Communications Surveys & Tutorials, vol. 17, no. 3, pp. 1776–1800, 2015.

[35] Q. A. Nguyen and B. Jaumard, "Optimal provisioning of elastic optical networks," in 2020 22nd International Conference on Transparent Optical Networks (ICTON), 2020, pp. 1–5.

[36] S. Talebi, F. Alam, I. Katib, M. Khamis, R. Salama, and G. N. Rouskas, "Spectrum management techniques for elastic optical networks: A survey," Optical Switching and Networking, vol. 13, pp. 34–48, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1573427714000253

[37] Q. Rahman, Y. Aneja, S. Bandyopadhyay, and A. Jaekel, "Optimal regenerator placement in survivable translucent networks," in 2014 10th International Conference on the Design of Reliable Communication Networks (DRCN), 2014, pp. 1–7.

[38] E. Lach and W. Idler, "Modulation formats for 100g and beyond," Optical Fiber Technology, vol. 17, no. 5, pp. 377–386, 2011, 100G and Beyond. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1068520011001027

[39] S. S. Dash, F. Pythoud, D. Hillerkuss, B. Baeuerle, A. Josten, P. Leuchtmann, and J. Leuthold, "Constellation modulation – an approach to increase spectral efficiency," Opt. Express, vol. 25, no. 14, pp. 16 310–16 331, Jul 2017. [Online]. Available: https://opg.optica.org/oe/abstract.cfm?URI=oe-25-14-16310

[40] Y. Lee and B. Mukherjee, "Traffic engineering in next-generation optical networks," IEEE Communications Surveys & Tutorials, vol. 6, no. 3, pp. 16–33, 2004.

[41] S. Zhang, C. Martel, and B. Mukherjee, "Dynamic traffic grooming in elastic optical networks," IEEE Journal on Selected Areas in Communications, vol. 31, no. 1, pp. 4–12, 2013.

[42] A. Fontinele, I. Santos, J. N. Neto, D. R. Campelo, and A. Soares, "An efficient ia-rmlsa algorithm for transparent elastic optical networks," Computer Networks, vol. 118, pp. 1–14, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128617300634

[43] A. M. Law, Simulation Modeling & Analysis, 5th ed. New York, NY, USA: McGraw-Hill, 2015, ch. 1.

[44] A. C. Drummond, "Wdmsim: Wdm optical network simulator," http://www.lrc.ic.unicamp.br/wdmsim/.

[45] P. M. Moura and A. C. Drummond, "FlexGridSim: Flexible Grid Optical Network Simulator," http://www.lrc.ic.unicamp.br/FlexGridSim/.

[46] M. Aibin and M. Blazejewski, "Complex elastic optical network simulator (ceons)," in 2015 17th International Conference on Transparent Optical Networks (ICTON), 2015, pp. 1–4.

[47] A. Varga, OMNeT++. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 35–59. [Online]. Available: https://doi.org/10.1007/978-3-642-12331-3_3

[48] L. R. Costa and A. C. Drummond, "ONS – optical network simulator," //ons-simulator.com, 2020.

[49] R. S. Tessinari, B. Puype, D. Colle, and A. S. Garcia, "Elastico++: An elastic optical network simulation framework for omnet++," Optical Switching and Networking, vol. 22, pp. 95–104, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1573427716300571

[50] F. Falcón, G. España, and D. Bórquez-Paredes, "Flex net sim: A lightly manual," 2021.

[51] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of json and xml data interchange formats: A case study," in 22nd International Conference on Computer Applications in Industry and Engineering 2009, CAINE 2009, 01 2009, pp. 157–162.

[52] B. Choudhary and S. K. Rakesh, "An approach using agile method for software development," in 2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH), 2016, pp. 155–158.

[53] S. Sharma and N. Hasteer, "A comprehensive study on state of scrum development," in 2016 International Conference on Computing, Communication and Automation (ICCCA), 2016, pp. 867–872.

[54] M. Coram and S. Bohner, "The impact of agile methods on software project management," in 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05), 2005, pp. 363–370.

[55] A. F. Chowdhury and M. N. Huda, "Comparison between adaptive software development and feature driven development," in Proceedings of 2011 International Conference on Computer Science and Network Technology, vol. 1, 2011, pp. 363–367.

[56] M. O. Ahmad, J. Markkula, and M. Oivo, "Kanban in software development: A systematic literature review," in 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, 2013, pp. 9–16.

[57] D. M. D. R. Vijay Anand, "Popular agile methods in software development: review and analysis," International Journal of Scientific and Technical Advancements, vol. 2, no. 4, pp. 147–150, 2016.

[58] D. Spinellis, "Git," IEEE Software, vol. 29, no. 3, pp. 100–101, 2012.

[59] "std::map," [Online; accessed 24-September-2023]. [Online]. Available: https://cplusplus.com/reference/map/map/

[60] F. Calderón, A. Lozada, P. Morales, D. Bórquez-Paredes, N. Jara, R. Olivares, G. Saavedra, A. Beghelli, and A. Leiva, "Heuristic approaches for dynamic provisioning in multi-band elastic optical networks," IEEE Communications Letters, vol. 26, no. 2, pp. 379–383, 2022.

[61] W. Shi, Z. Zhu, M. Zhang, and N. Ansari, "On the effect of bandwidth fragmentation on blocking probability in elastic optical networks," IEEE Transactions on Communications, vol. 61, no. 7, pp. 2970–2978, 2013.

Apéndice A

Repositorio de Gitlab de la biblioteca Flex Net Sim

https://gitlab.com/DaniloBorquez/flex-net-sim

Apéndice B

Código variable 1 con módulo Multibanda

```
double computeMedian(std::vector<double>& datos) {
    std::sort(datos.begin(), datos.end());
    size_t n = datos.size();
    if (n % 2 == 0) {
        size_t mitad = n / 2;
        return (datos[mitad - 1] + datos[mitad]) / 2.0;
    } else {
        return datos[n / 2];
    }
}
void getMedian(std::vector<std::vector<std::vector<std::vector<Link *>>>> *paths){
  int minLength;
  int sri=0;
  std::vector<double> totalLength;
  for (int i = 0; i < nodes; ++i) {
    for (int j = 0; j < nodes; ++j) {
      if(i==j) continue;
      for (int r = 0; r < paths->at(i)[j].size(); ++r) {
        double length = 0;
        for (int l = 0; l < paths->at(i)[j][r].size(); l++) {
          length+=paths->at(i)[j][r][l]->getLength();
        }
        if (r == 0) {
          minLength = length;
          sri = r;
        }
        if (length < minLength) {
          minLength = length;
          sri = r;
        }
      }
      std::vector<int> sriPlusLength={sri,minLength};
      totalLength.push_back(minLength);
      shorterIndex[std::make_pair(i, j)] = sriPlusLength;
    }
  }
  median = computeMedian(totalLength);
}
BEGIN_ALLOC_FUNCTION(V1) {
  int currentNumberSlots;
  int currentSlotIndex;
  int numberOfSlots;
  int r=shorterIndex[std::make_pair(SRC, DST)][0]; // Shorter route index
  int minLength = shorterIndex[std::make_pair(SRC, DST)][1];
  if(minLength<median) orderOfBands="set_1";
```

```
46  else orderOfBands="set_2";
47  std::vector<char> bandsOrder = bandsPreference[orderOfBands];
48  std::vector<char> bands;
49  char band;
50  std::map<char, std::vector<bool>> totalSlots;
51  int bitrateInt = bitRates_map[REQ_BITRATE];
52  bitrateCountTotal[bitrateInt] += 1;
53  bands = VECTOR_OF_BANDS(r, 0); // The vector of bands (chars) for the first Link
54  for (int bn = 0; bn < NUMBER_OF_BANDS(r, 0); bn++) {
55    band = bands[bn];
56    totalSlots[band] = std::vector<bool>(LINK_IN_ROUTE(r, 0)->getSlots(band), false);
57  }
58  for (int l = 0; l < NUMBER_OF_LINKS(r); l++) {
59    bands = LINK_IN_ROUTE(r, l)->getBands();
60    for (int bn = 0; bn < NUMBER_OF_BANDS(r, l); bn++) {
61      band = bands[bn];
62      for (int s = 0; s < LINK_IN_ROUTE(r, l)->getSlots(band); s++) {
63        totalSlots[band][s] = totalSlots[band][s] | LINK_IN_ROUTE(r, l)->getSlot(s,
    band);
64      }
65    }
66  }
67  for (int bo = 0; bo < bandsOrder.size(); bo++) {
68    for (int m = 0; m < NUMBER_OF_MODULATIONS; m++) {
69      std::map<char, int> bandPos = REQ_POS_BANDS(m);
70      char band = bandsOrder[bo];
71      int bandIndex = bandPos[band];
72      numberOfSlots = REQ_SLOTS_BDM(m, bandIndex);
73      if (minLength > REQ_REACH_BDM(m, bandIndex)) continue;
74      currentNumberSlots = 0;
75      currentSlotIndex = 0;
76      for (int s = 0; s < totalSlots[band].size(); s++) {
77        if (totalSlots[band][s] == false) {
78          currentNumberSlots++;
79        } else {
80          currentNumberSlots = 0;
81          currentSlotIndex = s + 1;
82        }
83        if (currentNumberSlots == numberOfSlots) {
84          for (int l = 0; l < NUMBER_OF_LINKS(r); l++) {
85            ALLOC_SLOTS_BDM(LINK_IN_ROUTE_ID(r, l), band, currentSlotIndex,
    numberOfSlots)
86          }
87          return ALLOCATED;
88        }
89      }
90    }
91  }
92  bitrateCountBlocked[bitrateInt] += 1;
93  return NOT_ALLOCATED;
94 }
95 END_ALLOC_FUNCTION
```

Apéndice C

Código variable 1 sin módulo Multibanda

```
class AuxRuta {
public:
    AuxRuta();
    AuxRuta(const std::vector<Link *> &enlaces, int ranuras, std::string modulacion,
    int modulacion_int, double alcance, int banda_int, std::string banda, double
    velocidad_bits, int longitud_total_enlaces, int id_ruta);
    ~AuxRuta();
    int obtenerRanuras();
    double obtenerAlcance();
    double obtenerVelocidadBits();
    std::vector<Link *> obtenerEnlaces();
    std::string obtenerModulacion();
    int obtenerModulacionInt();
    int obtenerRecursos();
    int obtenerBandaInt();
    std::string obtenerBanda();
    int obtenerLongitudTotal();
    int obtenerIdRuta();
private:
    int recursos;
    std::vector<Link *> enlaces;
    std::string modulacion;
    int modulacion_int;
    int ranuras;
    double alcance;
    double velocidad_bits;
    int banda_int;
    int id_ruta;
    int longitud_total_enlaces;
    std::string banda;
};
void parametrosCorteBanda(int tamano_enlaces, std::string banda, int **arr) {
    int banda = bandas[banda];
    *arr = new int[3];
    switch(banda) {
        case 0:
            (*arr)[0] = 0;
            (*arr)[1] = 343;
            (*arr)[2] = 344;
            break;
        case 1:
            (*arr)[0] = 344;
            (*arr)[1] = 823;
            (*arr)[2] = 480;
            break;
```

```
        case 2:
            (*arr)[0] = 824;
            (*arr)[1] = 1583;
            (*arr)[2] = 760;
            break;
        case 3:
            (*arr)[0] = 1584;
            (*arr)[1] = 2719;
            (*arr)[2] = 1136;
            break;
    }
    return;
}
AuxRuta::AuxRuta(const std::vector<Link *> &enlaces, int ranuras, std::string
    modulacion, int modulacion_int, double alcance, int banda_int, std::string banda,
     double velocidad_bits, int longitud_total_enlaces, int id_ruta){
    this->enlaces = enlaces;
    this->ranuras = ranuras;
    this->modulacion = modulacion;
    this->modulacion_int = modulacion_int;
    this->alcance = alcance;
    this->recursos = this->ranuras * enlaces.size();
    this->velocidad_bits = velocidad_bits;
    this->longitud_total_enlaces = longitud_total_enlaces;
    this->id_ruta = id_ruta;
    this->banda = banda;
    this->banda_int = banda_int;
};
AuxRuta::AuxRuta(){
    this->enlaces = std::vector<Link *>();
    this->ranuras = 0;
    this->modulacion = std::string();
    this->alcance = 0;
    this->recursos = 0;
    this->velocidad_bits = 0;
}
AuxRuta::~AuxRuta() {}
int AuxRuta::obtenerRanuras(){
    return this->ranuras;
}
double AuxRuta::obtenerAlcance(){
    return this->alcance;
}
double AuxRuta::obtenerVelocidadBits(){
    return this->velocidad_bits;
}
std::vector<Link *> AuxRuta::obtenerEnlaces(){
    return this->enlaces;
}
std::string AuxRuta::obtenerModulacion(){
    return this->modulacion;
}
int AuxRuta::obtenerModulacionInt(){
    return this->modulacion_int;
}
int AuxRuta::obtenerRecursos(){
    return this->recursos;
}
std::string AuxRuta::obtenerBanda(){
    return this->banda;
}
int AuxRuta::obtenerBandaInt(){
    return this->banda_int;
```

```
105 }
106 int AuxRuta::obtenerLongitudTotal(){
107     return this->longitud_total_enlaces;
108 }
109 int AuxRuta::obtenerIdRuta(){
110     return this->id_ruta;
111 }
112 bool RBM_V(AuxRuta *v1, AuxRuta *v2){
113     if (v1->obtenerLongitudTotal() != v2->obtenerLongitudTotal()) return v1->
    obtenerLongitudTotal() < v2->obtenerLongitudTotal();
114     if (v1->obtenerBandaInt() != v2->obtenerBandaInt()) return v1->obtenerBandaInt()
    < v2->obtenerBandaInt();
115     return v1->obtenerModulacionInt() < v2->obtenerModulacionInt();
116 }
117 void enfoqueOfflineLiberar(std::vector<std::vector<std::vector<std::vector<AuxRuta
    *>>>> rutasOffline){
118     for (int b = 0; b < rutasOffline.size(); b++)
119         for (int s = 0; s < rutasOffline[b].size(); s++)
120             for (int d = 0; d < rutasOffline[b][s].size(); d++)
121                 for (int r = 0; r < rutasOffline[b][s][d].size(); r++)
122                     delete(rutasOffline[b][s][d][r]);
123 }
124 double calcularMediana(std::vector<double> longitudes)
125 {
126   size_t tamano = longitudes.size();
127   if (tamano == 0)
128   {
129     return 0;  // Indefinido.
130   }
131   else
132   {
133     sort(longitudes.begin(), longitudes.end());
134     if (tamano % 2 == 0)
135     {
136       return (longitudes[tamano / 2 - 1] + longitudes[tamano / 2]) / 2;
137     }
138     else
139     {
140       return longitudes[tamano / 2];
141     }
142   }
143 }
144 std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>
    enfoqueOfflineOrden_V1_Conjunto1(std::vector<TasaBits> tasasBits,
145 {
146     std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>> rutasOffline = std
    ::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>(tasasBits.size());
147     for (int b = 0; b < tasasBits.size(); b++){ // Para tasaBits
148         rutasOffline[b] = std::vector<std::vector<std::vector<AuxRuta *>>>((*rutasRed
    ).size());
149         for (int s = 0; s < rutasRed->size();
150             s++){ // Para SRC s
151             rutasOffline[b][s] = std::vector<std::vector<AuxRuta *>>((*rutasRed)[s].
    size());
152             for (int d = 0; d < (*rutasRed)[s].size();
153                 d++){ // Para DST d
154                 if (s == d) continue; // Si SRC = DST, saltar
155                 rutasOffline[b][s][d] = std::vector<AuxRuta *>();
156                 for (int r = 0; r < (*rutasRed)[s][d].size();
157                     r++){ // Para cada RUTA r entre s y d
158                     double LongitudTotal = 0;
159                     for (int l = 0; l < (*rutasRed)[s][d][r].size(); l++)
```

```
                            LongitudTotal += (*rutasRed)[s][d][r][l]->obtenerLongitud
    ();
                    for (int m = 0; m < tasasBits[b].obtenerNumeroModulaciones();
                        m++){
                        if (tasasBits[b].obtenerAlcance(m) >= LongitudTotal){
                            std::string modulacion_y_banda = tasasBits[b].
    obtenerModulacion(m);
                            std::string modulacion = modulacion_y_banda.substr(0,
    modulacion_y_banda.find('-'));
                            modulacion_y_banda.erase(0, modulacion_y_banda.find('-')
    + 1);
                            std::string banda = modulacion_y_banda.substr(0,
    modulacion_y_banda.find('-'));
                            rutasOffline[b][s][d].push_back(new AuxRuta((*rutasRed)[s
    ][d][r],tasasBits[b].obtenerNumeroRanuras(m),modulacion,modulaciones[modulacion],
                        }
                    }
                }
                // Hecho con las rutas entre [s][d] en la tasa de bits b, ordenamos
    las rutas:
                std::sort(rutasOffline[b][s][d].begin(), rutasOffline[b][s][d].end(),
     RBM_V);
            }
        }
    }
    return rutasOffline;
}
std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>
    enfoqueOfflineOrden_V1_Conjunto2(std::vector<TasaBits> tasasBits,
{
    std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>> rutasOffline = std
    ::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>(tasasBits.size());
    for (int b = 0; b < tasasBits.size(); b++){ // Para tasaBits
        rutasOffline[b] = std::vector<std::vector<std::vector<AuxRuta *>>>((*rutasRed
    ).size());
        for (int s = 0; s < rutasRed->size();
            s++){ // Para SRC s
            rutasOffline[b][s] = std::vector<std::vector<AuxRuta *>>((*rutasRed)[s].
    size());
            for (int d = 0; d < (*rutasRed)[s].size(); // Para DST d
                if (s == d) continue; // Si SRC = DST, saltar
                rutasOffline[b][s][d] = std::vector<AuxRuta *>();
                for (int r = 0; r < (*rutasRed)[s][d].size();
                    r++){ // Para cada RUTA r entre s y d
                    double LongitudTotal = 0;
                    for (int l = 0; l < (*rutasRed)[s][d][r].size(); l++)
                            LongitudTotal += (*rutasRed)[s][d][r][l]->obtenerLongitud
    ();
                    for (int m = 0; m < tasasBits[b].obtenerNumeroModulaciones();
                        m++){
                        if (tasasBits[b].obtenerAlcance(m) >= LongitudTotal){
                            std::string modulacion_y_banda = tasasBits[b].
    obtenerModulacion(m);
                            std::string modulacion = modulacion_y_banda.substr(0,
    modulacion_y_banda.find('-'));
                            modulacion_y_banda.erase(0, modulacion_y_banda.find('-')
    + 1);
                            std::string banda = modulacion_y_banda.substr(0,
    modulacion_y_banda.find('-'));
                            rutasOffline[b][s][d].push_back(new AuxRuta((*rutasRed)[s
    ][d][r],
                        }
                    }
```

```
205                 }
206                 std::sort(rutasOffline[b][s][d].begin(), rutasOffline[b][s][d].end(),
        RBM_V);
207               }
208           }
209       }
210       return rutasOffline;
211   }
212   BEGIN_ALLOC_FUNCTION(Variante_1) {
213     int ranurasActuales;
214     int indiceRanuraActual;
215     int *indices_ranuras_banda = NULL;
216     int tasaBitsInt = mapa_tasasBits[TASA_BITS_SOLICITADA];
217     int numeroRanuras;
218     std::vector<AuxRuta *> *rutasRBMSA_tasaBits_conjunto;
219     std::vector<bool> ranurasTotales;
220     conteo_tasasBits_total[tasaBitsInt] += 1;
221     if (rutasOffline[tasaBitsInt][SRC][DST][0]->obtenerLongitudTotal() < mediana_v1){
222       rutasRBMSA_tasaBits_conjunto = &rutasOffline_conjunto1[tasaBitsInt][SRC][DST];
223     } else rutasRBMSA_tasaBits_conjunto = &rutasOffline_conjunto2[tasaBitsInt][SRC][DST
        ];
224     for (int r = 0; r < (*rutasRBMSA_tasaBits_conjunto).size();
225         r++){ // <- Para ruta r entre SRC y DST
226       numeroRanuras = (*rutasRBMSA_tasaBits_conjunto)[r]->obtenerRanuras();
227       ranurasTotales = std::vector<bool>((*rutasRBMSA_tasaBits_conjunto)[r]->
        obtenerEnlaces()[0]->obtenerRanuras(), false);
228       if (indices_ranuras_banda != NULL) delete(indices_ranuras_banda);
229       parametrosCorteBanda((*rutasRBMSA_tasaBits_conjunto)[r]->obtenerEnlaces()[0]->
        obtenerRanuras(),
230                            (*rutasRBMSA_tasaBits_conjunto)[r]->obtenerBanda(),
231                            &indices_ranuras_banda);
232       for (int l = 0; l < (*rutasRBMSA_tasaBits_conjunto)[r]->obtenerEnlaces().size();
233           l++){ // <- Para enlace l en ruta r
234         for (int s = indices_ranuras_banda[0]; s <= indices_ranuras_banda[1];
235             s++){ // <- Para cada ranura s en la banda actual, del enlace l
236           ranurasTotales[s] = ranurasTotales[s] | (*rutasRBMSA_tasaBits_conjunto)[r]->
        obtenerEnlaces()[l]->obtenerRanura(s);
237         }
238       }
239       ranurasActuales = 0;
240       indiceRanuraActual = indices_ranuras_banda[0];
241       for (int s = indices_ranuras_banda[0]; s <= indices_ranuras_banda[1];
242           s++){
243         if (ranurasTotales[s] == false) ranurasActuales++;
244         else {
245           ranurasActuales = 0;
246           indiceRanuraActual = s + 1;
247       }
248         if (ranurasActuales == numeroRanuras) {
249           for (int l = 0; l < (*rutasRBMSA_tasaBits_conjunto)[r]->obtenerEnlaces().size
        ();
250                l++){
251             ASIGNAR_RANURAS((*rutasRBMSA_tasaBits_conjunto)[r]->obtenerEnlaces()[l]->
        obtenerId(), indiceRanuraActual, numeroRanuras)
252           }
253           if (indices_ranuras_banda != NULL) delete(indices_ranuras_banda);
254           return ALLOCATED;
255         }
256       }
257     }
258     conteo_tasasBits_bloqueado[tasaBitsInt] += 1;
259     if (indices_ranuras_banda != NULL) delete(indices_ranuras_banda);
260     return NOT_ALLOCATED;
```

```
261 }
262 END_ALLOC_FUNCTION
```

Apéndice D

Código variable 2 con módulo Multibanda

```
double computeMedian(std::vector<double>& datos) {
    std::sort(datos.begin(), datos.end());
    size_t n = datos.size();
    if (n % 2 == 0) {
        size_t mitad = n / 2;
        return (datos[mitad - 1] + datos[mitad]) / 2.0;
    } else {
        return datos[n / 2];
    }
}
void shortestRoute(std::vector<std::vector<std::vector<std::vector<Link *>>>> *paths)
    {
  double tLR=0;
  int minLength=0;
  int sri=0;
  std::vector<double> totalLength;
  for (int i = 0; i < nodes; ++i) {
    for (int j = 0; j < nodes; ++j) {
      if(i==j) continue;
      for (int r = 0; r < paths->at(i)[j].size(); ++r) {
        double length = 0;
        for (int l = 0; l < paths->at(i)[j][r].size(); l++) {
          length+=paths->at(i)[j][r][l]->getLength();
        }
        if (r == 0) {
          minLength = length;
          sri = r;
        }
        if (length < minLength) {
          minLength = length;
          sri = r;
        }
      }
      if (minLength > tLR) {
          tLR = minLength;
      }
      std::vector<int> sriPlusLength={sri,minLength};
      totalLength.push_back(minLength);
      shorterIndex[std::make_pair(i, j)] = sriPlusLength;
    }
  }
  LR=computeMedian(totalLength);;
}
BEGIN_ALLOC_FUNCTION(V2) {
  int currentNumberSlots;
```

```
  int currentSlotIndex;
  int numberOfSlots;
  int r=shorterIndex[std::make_pair(SRC, DST)][0]; // Shorter route index
  double minLength = shorterIndex[std::make_pair(SRC, DST)][1];
  if(minLength <= LR/4) orderOfBands="set_1";
  else if(minLength <= LR/2) orderOfBands="set_2";
  else if(minLength <= (3*LR)/4) orderOfBands="set_3";
  else orderOfBands="set_4";
  std::vector<char> bandsOrder = bandsPreference[orderOfBands];
  std::vector<char> bands;
  char band;
  std::map<char, std::vector<bool>> totalSlots;
  int bitrateInt = bitRates_map[REQ_BITRATE];
  bitrateCountTotal[bitrateInt] += 1;
  bands = VECTOR_OF_BANDS(r, 0); // The vector of bands (chars) for the first Link
  for (int bn = 0; bn < NUMBER_OF_BANDS(r, 0); bn++) {
    band = bands[bn];
    totalSlots[band] = std::vector<bool>(LINK_IN_ROUTE(r, 0)->getSlots(band), false);
  }
  for (int l = 0; l < NUMBER_OF_LINKS(r); l++) {
    bands = LINK_IN_ROUTE(r, l)->getBands();
    for (int bn = 0; bn < NUMBER_OF_BANDS(r, l); bn++) {
      band = bands[bn];
      for (int s = 0; s < LINK_IN_ROUTE(r, l)->getSlots(band); s++) {
        totalSlots[band][s] = totalSlots[band][s] | LINK_IN_ROUTE(r, l)->getSlot(s,
    band);
      }
    }
  }
  for (int bo = 0; bo < bandsOrder.size(); bo++) {
    for (int m = 0; m < NUMBER_OF_MODULATIONS; m++) {
      std::map<char, int> bandPos = REQ_POS_BANDS(m);
      char band = bandsOrder[bo];
      int bandIndex = bandPos[band];
      numberOfSlots = REQ_SLOTS_BDM(m, bandIndex);
      if (minLength > REQ_REACH_BDM(m, bandIndex)) continue;
      currentNumberSlots = 0;
      currentSlotIndex = 0;
      for (int s = 0; s < totalSlots[band].size(); s++) {
        if (totalSlots[band][s] == false) {
          currentNumberSlots++;
        } else {
          currentNumberSlots = 0;
          currentSlotIndex = s + 1;
        }
        if (currentNumberSlots == numberOfSlots) {
          // Allocate slots for the selected band and modulation
          for (int l = 0; l < NUMBER_OF_LINKS(r); l++) {
            ALLOC_SLOTS_BDM(LINK_IN_ROUTE_ID(r, l), band, currentSlotIndex,
    numberOfSlots)
          }
          return ALLOCATED;
        }
      }
    }
  }
  bitrateCountBlocked[bitrateInt] += 1;
  return NOT_ALLOCATED;
}
END_ALLOC_FUNCTION
```

Apéndice E

Código variable 2 sin módulo Multibanda

```
class AuxRuta {
public:
    AuxRuta();
    AuxRuta(const std::vector<Link *> &enlaces, int ranuras, std::string modulacion,
    int modulacion_int, double alcance, int banda_int, std::string banda, double
    velocidad_bits, int longitud_total_enlaces, int id_ruta);
    ~AuxRuta();
    int obtenerRanuras();
    double obtenerAlcance();
    double obtenerVelocidadBits();
    std::vector<Link *> obtenerEnlaces();
    std::string obtenerModulacion();
    int obtenerModulacionInt();
    int obtenerRecursos();
    int obtenerBandaInt();
    std::string obtenerBanda();
    int obtenerLongitudTotal();
    int obtenerIdRuta();
private:
    int recursos;
    std::vector<Link *> enlaces;
    std::string modulacion;
    int modulacion_int;
    int ranuras;
    double alcance;
    double velocidad_bits;
    int banda_int;
    int id_ruta;
    int longitud_total_enlaces;
    std::string banda;
};
void parametrosCorteBanda(int tamano_enlaces, std::string banda, int **arr) {
    int banda = bandas[banda];
    *arr = new int[3];
    switch(banda) {
        case 0:
            (*arr)[0] = 0;
            (*arr)[1] = 343;
            (*arr)[2] = 344;
            break;
        case 1:
            (*arr)[0] = 344;
            (*arr)[1] = 823;
            (*arr)[2] = 480;
            break;
```

```
        case 2:
            (*arr)[0] = 824;
            (*arr)[1] = 1583;
            (*arr)[2] = 760;
            break;
        case 3:
            (*arr)[0] = 1584;
            (*arr)[1] = 2719;
            (*arr)[2] = 1136;
            break;
    }
    return;
}
AuxRuta::AuxRuta(const std::vector<Link *> &enlaces, int ranuras, std::string
    modulacion, int modulacion_int, double alcance, int banda_int, std::string banda,
     double velocidad_bits, int longitud_total_enlaces, int id_ruta){
    this->enlaces = enlaces;
    this->ranuras = ranuras;
    this->modulacion = modulacion;
    this->modulacion_int = modulacion_int;
    this->alcance = alcance;
    this->recursos = this->ranuras * enlaces.size();
    this->velocidad_bits = velocidad_bits;
    this->longitud_total_enlaces = longitud_total_enlaces;
    this->id_ruta = id_ruta;
    this->banda = banda;
    this->banda_int = banda_int;
};
AuxRuta::AuxRuta(){
    this->enlaces = std::vector<Link *>();
    this->ranuras = 0;
    this->modulacion = std::string();
    this->alcance = 0;
    this->recursos = 0;
    this->velocidad_bits = 0;
}
AuxRuta::~AuxRuta() {}
int AuxRuta::obtenerRanuras(){
    return this->ranuras;
}
double AuxRuta::obtenerAlcance(){
    return this->alcance;
}
double AuxRuta::obtenerVelocidadBits(){
    return this->velocidad_bits;
}
std::vector<Link *> AuxRuta::obtenerEnlaces(){
    return this->enlaces;
}
std::string AuxRuta::obtenerModulacion(){
    return this->modulacion;
}
int AuxRuta::obtenerModulacionInt(){
    return this->modulacion_int;
}
int AuxRuta::obtenerRecursos(){
    return this->recursos;
}
std::string AuxRuta::obtenerBanda(){
    return this->banda;
}
int AuxRuta::obtenerBandaInt(){
    return this->banda_int;
```

```
105 }
106 int AuxRuta::obtenerLongitudTotal(){
107     return this->longitud_total_enlaces;
108 }
109 int AuxRuta::obtenerIdRuta(){
110     return this->id_ruta;
111 }
112 bool RBM_V(AuxRuta *v1, AuxRuta *v2){
113     if (v1->obtenerLongitudTotal() != v2->obtenerLongitudTotal()) return v1->
    obtenerLongitudTotal() < v2->obtenerLongitudTotal();
114     if (v1->obtenerBandaInt() != v2->obtenerBandaInt()) return v1->obtenerBandaInt()
    < v2->obtenerBandaInt();
115     return v1->obtenerModulacionInt() < v2->obtenerModulacionInt();
116 }
117 void enfoqueOfflineLiberar(std::vector<std::vector<std::vector<std::vector<AuxRuta
    *>>>> rutasOffline){
118     for (int b = 0; b < rutasOffline.size(); b++)
119         for (int s = 0; s < rutasOffline[b].size(); s++)
120             for (int d = 0; d < rutasOffline[b][s].size(); d++)
121                 for (int r = 0; r < rutasOffline[b][s][d].size(); r++)
122                     delete(rutasOffline[b][s][d][r]);
123 }
124 double calcularMediana(std::vector<double> longitudes)
125 {
126   size_t tamano = longitudes.size();
127   if (tamano == 0)
128   {
129     return 0;  // Indefinido.
130   }
131   else
132   {
133     sort(longitudes.begin(), longitudes.end());
134     if (tamano % 2 == 0)
135     {
136       return (longitudes[tamano / 2 - 1] + longitudes[tamano / 2]) / 2;
137     }
138     else
139     {
140       return longitudes[tamano / 2];
141     }
142   }
143 }
144 std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>
    enfoqueOfflineOrden_V2_Conjunto1(std::vector<TasaBits> tasasBits,
145                                                                 std::vector<std::
    vector<std::vector<std::vector<Enlace *>>>> *redRutas)
146 {
147     std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>> rutasOffline = std
    ::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>(tasasBits.size());
148     for (int b = 0; b < tasasBits.size(); b++){ // Para tasasBits
149         rutasOffline[b] = std::vector<std::vector<std::vector<AuxRuta *>>>((*redRutas
    ).size());
150         for (int s = 0; s < redRutas->size(); s++){ // Para SRC s
151             rutasOffline[b][s] = std::vector<std::vector<AuxRuta *> ((*redRutas)[s].
    size());
152             for (int d = 0; d < (*redRutas)[s].size(); d++){ // Para DST d
153                 if (s == d) continue; // SI SRC = DST, omitir
154                 rutasOffline[b][s][d] = std::vector<AuxRuta *>();
155                 for (int r = 0; r < (*redRutas)[s][d].size(); r++){ // Para cada RUTA
     r entre s y d
156                     double LongitudTotal = 0;
157                     for (int l = 0; l < (*redRutas)[s][d][r].size(); l++)
```

```
158                             LongitudTotal += (*redRutas)[s][d][r][l]->obtenerLongitud
    ();
159                     for (int m = 0; m < tasasBits[b].obtenerNumeroModulaciones(); m
    ++){
160                         if (tasasBits[b].obtenerAlcance(m) >= LongitudTotal){
161                             std::string modulacionYBanda = tasasBits[b].
    obtenerModulacion(m);
162                             std::string modulacion = modulacionYBanda.substr(0,
    modulacionYBanda.find('-'));
163                             modulacionYBanda.erase(0, modulacionYBanda.find('-') + 1)
    ;
164                             std::string banda = modulacionYBanda.substr(0,
    modulacionYBanda.find('-'));
165                             rutasOffline[b][s][d].push_back(new AuxRuta((*redRutas)[s
    ][d][r],
166                                                             tasasBits[b].obtenerNumeroRanuras
    (m),
167                                                             modulacion,
168                                                             modulaciones[modulacion],
169                                                             tasasBits[b].obtenerAlcance(m),
170                                                             bandas_v2_conjunto1[banda],
171                                                             banda,
172                                                             tasasBits[b].obtenerTasaBits(),
173                                                             LongitudTotal,
174                                                             r));
175                         }
176                     }
177                 }
178                 std::sort(rutasOffline[b][s][d].begin(), rutasOffline[b][s][d].end(),
     RBM_V);
179             }
180         }
181     }
182     return rutasOffline;
183 }
184 std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>
    enfoqueOfflineOrden_V2_Conjunto2(std::vector<TasaBits> tasasBits,
185                                                             std::vector<std::
    vector<std::vector<std::vector<Enlace *>>>> *redRutas)
186 {
187     std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>> rutasOffline = std
    ::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>(tasasBits.size());
188     for (int b = 0; b < tasasBits.size(); b++){ // Para tasasBits
189         rutasOffline[b] = std::vector<std::vector<std::vector<AuxRuta *>>>((*redRutas
    ).size());
190         for (int s = 0; s < redRutas->size(); s++){ // Para SRC s
191             rutasOffline[b][s] = std::vector<std::vector<AuxRuta *> ((*redRutas)[s].
    size());
192             for (int d = 0; d < (*redRutas)[s].size(); d++){ // Para DST d
193                 if (s == d) continue; // SI SRC = DST, omitir
194                 rutasOffline[b][s][d] = std::vector<AuxRuta *>();
195                 for (int r = 0; r < (*redRutas)[s][d].size(); r++){ // Para cada RUTA
     r entre s y d
196                     double LongitudTotal = 0;
197                     for (int l = 0; l < (*redRutas)[s][d][r].size(); l++)
198                             LongitudTotal += (*redRutas)[s][d][r][l]->obtenerLongitud
    ();
199                     for (int m = 0; m < tasasBits[b].obtenerNumeroModulaciones(); m
    ++){
200                         if (tasasBits[b].obtenerAlcance(m) >= LongitudTotal){
201                             std::string modulacionYBanda = tasasBits[b].
    obtenerModulacion(m);
```

```
202                             std::string modulacion = modulacionYBanda.substr(0,
    modulacionYBanda.find('-'));
203                             modulacionYBanda.erase(0, modulacionYBanda.find('-') + 1)
    ;
204                             std::string banda = modulacionYBanda.substr(0,
    modulacionYBanda.find('-'));
205                             rutasOffline[b][s][d].push_back(new AuxRuta((*redRutas)[s
    ][d][r],
206                                                             tasasBits[b].obtenerNumeroRanuras
    (m),
207                                                             modulacion,
208                                                             modulaciones[modulacion],
209                                                             tasasBits[b].obtenerAlcance(m),
210                                                             bandas_v2_conjunto2[banda],
211                                                             banda,
212                                                             tasasBits[b].obtenerTasaBits(),
213                                                             LongitudTotal,
214                                                             r));
215                         }
216                     }
217                 }
218                 std::sort(rutasOffline[b][s][d].begin(), rutasOffline[b][s][d].end(),
     RBM_V);
219             }
220         }
221     }
222     return rutasOffline;
223 }
224 std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>
    enfoqueOfflineOrden_V2_Conjunto3(std::vector<TasaBits> tasasBits,
225                                                                 std::vector<std::
    vector<std::vector<std::vector<Enlace *>>>> *redRutas)
226 {
227     std::vector<std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>
    rutasOffline = std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>(
    tasasBits.size());
228     for (int b = 0; b < tasasBits.size(); b++){ // Para tasasBits
229         rutasOffline[b] = std::vector<std::vector<std::vector<AuxRuta *>>>((*redRutas
    ).size());
230         for (int s = 0; s < redRutas->size(); s++){ // Para SRC s
231             rutasOffline[b][s] = std::vector<std::vector<AuxRuta *> ((*redRutas)[s].
    size());
232             for (int d = 0; d < (*redRutas)[s].size(); d++){ // Para DST d
233                 if (s == d) continue; // SI SRC = DST, omitir
234                 rutasOffline[b][s][d] = std::vector<AuxRuta *>();
235                 for (int r = 0; r < (*redRutas)[s][d].size(); r++){ // Para cada RUTA
     r entre s y d
236                     double LongitudTotal = 0;
237                     for (int l = 0; l < (*redRutas)[s][d][r].size(); l++)
238                             LongitudTotal += (*redRutas)[s][d][r][l]->obtenerLongitud
    ();
239                     for (int m = 0; m < tasasBits[b].obtenerNumeroModulaciones(); m
    ++){
240                         if (tasasBits[b].obtenerAlcance(m) >= LongitudTotal){
241                             std::string modulacionYBanda = tasasBits[b].
    obtenerModulacion(m);
242                             std::string modulacion = modulacionYBanda.substr(0,
    modulacionYBanda.find('-'));
243                             modulacionYBanda.erase(0, modulacionYBanda.find('-') + 1)
    ;
244                             std::string banda = modulacionYBanda.substr(0,
    modulacionYBanda.find('-'));
```

```
245                             rutasOffline[b][s][d].push_back(new AuxRuta((*redRutas)[s
    ][d][r],
246                                                             tasasBits[b].obtenerNumeroRanuras
    (m),
247                                                             modulacion,
248                                                             modulaciones[modulacion],
249                                                             tasasBits[b].obtenerAlcance(m),
250                                                             bandas_v2_conjunto3[banda],
251                                                             banda,
252                                                             tasasBits[b].obtenerTasaBits(),
253                                                             LongitudTotal,
254                                                             r));
255                         }
256                     }
257                 }
258                 std::sort(rutasOffline[b][s][d].begin(), rutasOffline[b][s][d].end(),
     RBM_V);
259             }
260         }
261     }
262     return rutasOffline;
263 }
264
265 std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>
    enfoqueOfflineOrden_V2_Conjunto4(std::vector<TasaBits> tasasBits,
266                                                                 std::vector<std::
    vector<std::vector<std::vector<Enlace *>>>> *redRutas)
267 {
268     std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>> rutasOffline = std
    ::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>(tasasBits.size());
269     for (int b = 0; b < tasasBits.size(); b++){ // Para tasasBits
270         rutasOffline[b] = std::vector<std::vector<std::vector<AuxRuta *>>>((*redRutas
    ).size());
271         for (int s = 0; s < redRutas->size(); s++){ // Para SRC s
272             rutasOffline[b][s] = std::vector<std::vector<AuxRuta *> ((*redRutas)[s].
    size());
273             for (int d = 0; d < (*redRutas)[s].size(); d++){ // Para DST d
274                 if (s == d) continue; // SI SRC = DST, omitir
275                 rutasOffline[b][s][d] = std::vector<AuxRuta *>();
276                 for (int r = 0; r < (*redRutas)[s][d].size(); r++){ // Para cada RUTA
     r entre s y d
277                     double LongitudTotal = 0;
278                     for (int l = 0; l < (*redRutas)[s][d][r].size(); l++)
279                             LongitudTotal += (*redRutas)[s][d][r][l]->obtenerLongitud
    ();
280                     for (int m = 0; m < tasasBits[b].obtenerNumeroModulaciones(); m
    ++){
281                         if (tasasBits[b].obtenerAlcance(m) >= LongitudTotal){
282                             std::string modulacionYBanda = tasasBits[b].
    obtenerModulacion(m);
283                             std::string modulacion = modulacionYBanda.substr(0,
    modulacionYBanda.find('-'));
284                             modulacionYBanda.erase(0, modulacionYBanda.find('-') + 1)
    ;
285                             std::string banda = modulacionYBanda.substr(0,
    modulacionYBanda.find('-'));
286                             rutasOffline[b][s][d].push_back(new AuxRuta((*redRutas)[s
    ][d][r],
287                                                             tasasBits[b].obtenerNumeroRanuras
    (m),
288                                                             modulacion,
289                                                             modulaciones[modulacion],
290                                                             tasasBits[b].obtenerAlcance(m),
```

```
                                                    bandas_v2_conjunto4[banda],
                                                    banda,
                                                    tasasBits[b].obtenerTasaBits(),
                                                    LongitudTotal,
                                                    r));
                        }
                    }
                }
                std::sort(rutasOffline[b][s][d].begin(), rutasOffline[b][s][d].end(),
     RBM_V);
            }
        }
    }
    return rutasOffline;
}
BEGIN_ALLOC_FUNCTION(Variant_2) {
  int numeroRanurasActuales;
  int indiceRanuraActual;
  int *indicesRanuraBanda = NULL;
  int tasaBitsInt = tasasBits_mapa[TASA_BITS_SOLICITADA];
  int numeroRanuras;
  std::vector<AuxRuta *> *rutasRBMSA_tasaBits_set;
  std::vector<bool> ranurasTotales;
  conteoTasasBitsTotales[tasaBitsInt] += 1;
  if (rutasOffline[tasaBitsInt][SRC][DST][0]->obtenerLongitudTotal() <=
     masLarga_menosCorta/4){
    rutasRBMSA_tasaBits_set = &rutasOffline_set1[tasaBitsInt][SRC][DST];
  } else if (rutasOffline[tasaBitsInt][SRC][DST][0]->obtenerLongitudTotal() <=
     masLarga_menosCorta/2){
    rutasRBMSA_tasaBits_set = &rutasOffline_set2[tasaBitsInt][SRC][DST];
  } else if (rutasOffline[tasaBitsInt][SRC][DST][0]->obtenerLongitudTotal() <= (3*
     masLarga_menosCorta)/4){
    rutasRBMSA_tasaBits_set = &rutasOffline_set3[tasaBitsInt][SRC][DST];
  } else rutasRBMSA_tasaBits_set = &rutasOffline_set4[tasaBitsInt][SRC][DST];
  for (int r = 0; r < (*rutasRBMSA_tasaBits_set).size(); r++){ // <- Para ruta r
     entre SRC y DST
    numeroRanuras = (*rutasRBMSA_tasaBits_set)[r]->obtenerRanuras();
    ranurasTotales = std::vector<bool>((*rutasRBMSA_tasaBits_set)[r]->obtenerEnlaces
    ()[0]->obtenerRanuras(), false);
    if (indicesRanuraBanda != NULL) delete(indicesRanuraBanda);
    parametrosRanuraBanda((*rutasRBMSA_tasaBits_set)[r]->obtenerEnlaces()[0]->
    obtenerRanuras(),
                          (*rutasRBMSA_tasaBits_set)[r]->obtenerBanda(),
                          &indicesRanuraBanda);

    for (int l = 0; l < (*rutasRBMSA_tasaBits_set)[r]->obtenerEnlaces().size(); l++){
     // <- Para enlace l en ruta r
      for (int s = indicesRanuraBanda[0]; s <= indicesRanuraBanda[1]; s++){ // <-
    Para cada ranura s en la banda actual, del enlace l
        ranurasTotales[s] = ranurasTotales[s] | (*rutasRBMSA_tasaBits_set)[r]->
    obtenerEnlaces()[l]->obtenerRanura(s);
      }
    }
    numeroRanurasActuales = 0;
    indiceRanuraActual = indicesRanuraBanda[0];
    for (int s = indicesRanuraBanda[0]; s <= indicesRanuraBanda[1]; s++){
      if (ranurasTotales[s] == false) numeroRanurasActuales++;
      else {
        numeroRanurasActuales = 0;
        indiceRanuraActual = s + 1;
    }
      if (numeroRanurasActuales == numeroRanuras) {
```

```
343         for (int l = 0; l < (*rutasRBMSA_tasaBits_set)[r]->obtenerEnlaces().size(); l
    ++){
344           ASIGNAR_RANURAS((*rutasRBMSA_tasaBits_set)[r]->obtenerEnlaces()[l]->
    obtenerId(), indiceRanuraActual, numeroRanuras)
345         }
346         if (indicesRanuraBanda != NULL) delete(indicesRanuraBanda);
347         return ALLOCATED;
348       }
349     }
350   }
351   conteoTasasBitsBloqueadas[tasaBitsInt] += 1;
352   if (indicesRanuraBanda != NULL) delete(indicesRanuraBanda);
353   return NOT_ALLOCATED;
354 }
355 END_ALLOC_FUNCTION
```

Apéndice F

Código variable 3 con módulo Multibanda

```
double computeMedian(std::vector<double>& datos) {
    std::sort(datos.begin(), datos.end());
    size_t n = datos.size();
    if (n % 2 == 0) {
        size_t mitad = n / 2;
        return (datos[mitad - 1] + datos[mitad]) / 2.0;
    } else {
        return datos[n / 2];
    }
}
void shortestRoute(std::vector<std::vector<std::vector<std::vector<Link *>>>> *paths,
     std::vector<double> bitrates){
  int minLength=0;
  int sri=0;
  for (int i = 0; i < nodes; ++i) {
    for (int j = 0; j < nodes; ++j) {
      if(i==j) continue;
      for (int r = 0; r < paths->at(i)[j].size(); ++r) {
        double length = 0;
        for (int l = 0; l < paths->at(i)[j][r].size(); l++) {
          length+=paths->at(i)[j][r][l]->getLength();
        }
        if (r == 0) {
          minLength = length;
          sri = r;
        }
        if (length < minLength) {
          minLength = length;
          sri = r;
        }
      }
        std::vector<int> sriPlusLength={sri,minLength};
        shorterIndex[std::make_pair(i, j)] = sriPlusLength;
    }
  }
  medianBitrate=computeMedian(bitrates);
}
BEGIN_ALLOC_FUNCTION(V3) {
  int currentNumberSlots;
  int currentSlotIndex;
  int numberOfSlots;
  int r=shorterIndex[std::make_pair(SRC, DST)][0]; // Shorter route index
  int minLength = shorterIndex[std::make_pair(SRC, DST)][1];
  if(REQ_BITRATE < medianBitrate) orderOfBands="set_1";
  else orderOfBands="set_2";
```

```
  std::vector<char> bandsOrder = bandsPreference[orderOfBands];
  std::vector<char> bands;
  int numberOfBands;
  char band;
  std::map<char, std::vector<bool>> totalSlots;
  int bitrateInt = bitRates_map[REQ_BITRATE];
  bitrateCountTotal[bitrateInt] += 1;
  bands = VECTOR_OF_BANDS(r, 0); // The vector of bands (chars) for the first Link
  for (int bn = 0; bn < NUMBER_OF_BANDS(r, 0); bn++) {
    band = bands[bn];
    totalSlots[band] = std::vector<bool>(LINK_IN_ROUTE(r, 0)->getSlots(band), false);
  }
  for (int l = 0; l < NUMBER_OF_LINKS(r); l++) {
    bands = LINK_IN_ROUTE(r, l)->getBands();
    numberOfBands = LINK_IN_ROUTE(r, l)->getNumberOfBands();
    for (int bn = 0; bn < numberOfBands; bn++) {
      band = bands[bn];
      for (int s = 0; s < LINK_IN_ROUTE(r, l)->getSlots(band); s++) {
        totalSlots[band][s] = totalSlots[band][s] | LINK_IN_ROUTE(r, l)->getSlot(s,
    band);
      }
    }
  }
  for (int bo = 0; bo < bandsOrder.size(); bo++) {
    for (int m = 0; m < NUMBER_OF_MODULATIONS; m++) {
      std::map<char, int> bandPos = REQ_POS_BANDS(m);
      char band = bandsOrder[bo];
      int bandIndex = bandPos[band];
      numberOfSlots = REQ_SLOTS_BDM(m, bandIndex);
      if (minLength > REQ_REACH_BDM(m, bandIndex)) continue;
      currentNumberSlots = 0;
      currentSlotIndex = 0;
      for (int s = 0; s < totalSlots[band].size(); s++) {
        if (totalSlots[band][s] == false) {
          currentNumberSlots++;
        } else {
          currentNumberSlots = 0;
          currentSlotIndex = s + 1;
        }
        if (currentNumberSlots == numberOfSlots) {
          for (int l = 0; l < NUMBER_OF_LINKS(r); l++) {
            ALLOC_SLOTS_BDM(LINK_IN_ROUTE_ID(r, l), band, currentSlotIndex,
    numberOfSlots)
          }
          return ALLOCATED;
        }
      }
    }
  }
  bitrateCountBlocked[bitrateInt] += 1;
  return NOT_ALLOCATED;
}
END_ALLOC_FUNCTION
```

Apéndice G

Código variable 3 sin módulo Multibanda

```
class AuxRuta {
public:
    AuxRuta();
    AuxRuta(const std::vector<Link *> &enlaces, int ranuras, std::string modulacion,
    int modulacion_int, double alcance, int banda_int, std::string banda, double
    velocidad_bits, int longitud_total_enlaces, int id_ruta);
    ~AuxRuta();
    int obtenerRanuras();
    double obtenerAlcance();
    double obtenerVelocidadBits();
    std::vector<Link *> obtenerEnlaces();
    std::string obtenerModulacion();
    int obtenerModulacionInt();
    int obtenerRecursos();
    int obtenerBandaInt();
    std::string obtenerBanda();
    int obtenerLongitudTotal();
    int obtenerIdRuta();
private:
    int recursos;
    std::vector<Link *> enlaces;
    std::string modulacion;
    int modulacion_int;
    int ranuras;
    double alcance;
    double velocidad_bits;
    int banda_int;
    int id_ruta;
    int longitud_total_enlaces;
    std::string banda;
};
void parametrosCorteBanda(int tamano_enlaces, std::string banda, int **arr) {
    int banda = bandas[banda];
    *arr = new int[3];
    switch(banda) {
        case 0:
            (*arr)[0] = 0;
            (*arr)[1] = 343;
            (*arr)[2] = 344;
            break;
        case 1:
            (*arr)[0] = 344;
            (*arr)[1] = 823;
            (*arr)[2] = 480;
            break;
```

```
        case 2:
            (*arr)[0] = 824;
            (*arr)[1] = 1583;
            (*arr)[2] = 760;
            break;
        case 3:
            (*arr)[0] = 1584;
            (*arr)[1] = 2719;
            (*arr)[2] = 1136;
            break;
    }
    return;
}
AuxRuta::AuxRuta(const std::vector<Link *> &enlaces, int ranuras, std::string
    modulacion, int modulacion_int, double alcance, int banda_int, std::string banda,
     double velocidad_bits, int longitud_total_enlaces, int id_ruta){
    this->enlaces = enlaces;
    this->ranuras = ranuras;
    this->modulacion = modulacion;
    this->modulacion_int = modulacion_int;
    this->alcance = alcance;
    this->recursos = this->ranuras * enlaces.size();
    this->velocidad_bits = velocidad_bits;
    this->longitud_total_enlaces = longitud_total_enlaces;
    this->id_ruta = id_ruta;
    this->banda = banda;
    this->banda_int = banda_int;
};
AuxRuta::AuxRuta(){
    this->enlaces = std::vector<Link *>();
    this->ranuras = 0;
    this->modulacion = std::string();
    this->alcance = 0;
    this->recursos = 0;
    this->velocidad_bits = 0;
}
AuxRuta::~AuxRuta() {}
int AuxRuta::obtenerRanuras(){
    return this->ranuras;
}
double AuxRuta::obtenerAlcance(){
    return this->alcance;
}
double AuxRuta::obtenerVelocidadBits(){
    return this->velocidad_bits;
}
std::vector<Link *> AuxRuta::obtenerEnlaces(){
    return this->enlaces;
}
std::string AuxRuta::obtenerModulacion(){
    return this->modulacion;
}
int AuxRuta::obtenerModulacionInt(){
    return this->modulacion_int;
}
int AuxRuta::obtenerRecursos(){
    return this->recursos;
}
std::string AuxRuta::obtenerBanda(){
    return this->banda;
}
int AuxRuta::obtenerBandaInt(){
    return this->banda_int;
```

```
105 }
106 int AuxRuta::obtenerLongitudTotal(){
107     return this->longitud_total_enlaces;
108 }
109 int AuxRuta::obtenerIdRuta(){
110     return this->id_ruta;
111 }
112 bool RBM_V(AuxRuta *v1, AuxRuta *v2){
113     if (v1->obtenerLongitudTotal() != v2->obtenerLongitudTotal()) return v1->
    obtenerLongitudTotal() < v2->obtenerLongitudTotal();
114     if (v1->obtenerBandaInt() != v2->obtenerBandaInt()) return v1->obtenerBandaInt()
    < v2->obtenerBandaInt();
115     return v1->obtenerModulacionInt() < v2->obtenerModulacionInt();
116 }
117 void enfoqueOfflineLiberar(std::vector<std::vector<std::vector<std::vector<AuxRuta
    *>>>> rutasOffline){
118     for (int b = 0; b < rutasOffline.size(); b++)
119         for (int s = 0; s < rutasOffline[b].size(); s++)
120             for (int d = 0; d < rutasOffline[b][s].size(); d++)
121                 for (int r = 0; r < rutasOffline[b][s][d].size(); r++)
122                     delete(rutasOffline[b][s][d][r]);
123 }
124 double calcularMediana(std::vector<double> longitudes)
125 {
126   size_t tamano = longitudes.size();
127   if (tamano == 0)
128   {
129     return 0;  // Indefinido.
130   }
131   else
132   {
133     sort(longitudes.begin(), longitudes.end());
134     if (tamano % 2 == 0)
135     {
136       return (longitudes[tamano / 2 - 1] + longitudes[tamano / 2]) / 2;
137     }
138     else
139     {
140       return longitudes[tamano / 2];
141     }
142   }
143 }
144 std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>
    enfoqueOffline_V3_Conjunto1(std::vector<TasaBits> tasasBits,
145                                                         std::vector<std::vector<std::
    vector<std::vector<Enlace *>>>> *redRutas)
146 {
147     std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>> rutasOffline = std
    ::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>(tasasBits.size());
148     for (int t = 0; t < tasasBits.size(); t++){ // Para tasaBits
149         rutasOffline[t] = std::vector<std::vector<std::vector<AuxRuta *>>>((*redRutas
    ).size());
150         for (int s = 0; s < redRutas->size(); s++){ // Para SRC s
151             rutasOffline[t][s] = std::vector<std::vector<AuxRuta *>>((*redRutas)[s].
    size());
152             for (int d = 0; d < (*redRutas)[s].size(); d++){ // Para DST d
153                 if (s == d) continue; // Si SRC = DST, saltar
154                 rutasOffline[t][s][d] = std::vector<AuxRuta *>();
155                 for (int r = 0; r < (*redRutas)[s][d].size(); r++){ // Para cada RUTA
     r entre s y d
156                     double LongitudTotal = 0;
157                     for (int l = 0; l < (*redRutas)[s][d][r].size(); l++)
158                         LongitudTotal += (*redRutas)[s][d][r][l]->getLongitud();
```

```
159                     for (int m = 0; m < tasasBits[t].getNumeroModulaciones(); m++){
160                         if (tasasBits[t].getAlcance(m) >= LongitudTotal){
161                             std::string modulacion_y_banda = tasasBits[t].
    getModulacion(m);
162                             std::string modulacion = modulacion_y_banda.substr(0,
    modulacion_y_banda.find('-'));
163                             modulacion_y_banda.erase(0, modulacion_y_banda.find('-')
    + 1);
164                             std::string banda = modulacion_y_banda.substr(0,
    modulacion_y_banda.find('-'));
165                             rutasOffline[t][s][d].push_back(new AuxRuta((*redRutas)[s
    ][d][r],
166                                                 tasasBits[t].getNumeroRanuras(m),
167                                                 modulacion,
168                                                 modulaciones[modulacion],
169                                                 tasasBits[t].getAlcance(m),
170                                                 bandas_v3_set1[banda],
171                                                 banda,
172                                                 tasasBits[t].getTasaBits(),
173                                                 LongitudTotal,
174                                                 r));
175                         }
176                     }
177                 }
178                 std::sort(rutasOffline[t][s][d].begin(), rutasOffline[t][s][d].end(),
     RBM_V);
179             }
180         }
181     }
182     return rutasOffline;
183 }
184 std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>
    enfoqueOffline_V3_Conjunto2(std::vector<TasaBits> tasasBits,
185                                                 std::vector<std::vector<std::
    vector<std::vector<Enlace *>>>> *redRutas)
186 {
187     std::vector<std::vector<std::vector<std::vector<AuxRuta *>>>> rutasOffline = std
    ::vector<std::vector<std::vector<std::vector<AuxRuta *>>>>(tasasBits.size());
188     for (int t = 0; t < tasasBits.size(); t++){ // Para tasaBits
189         rutasOffline[t] = std::vector<std::vector<std::vector<AuxRuta *>>>((*redRutas
    ).size());
190         for (int s = 0; s < redRutas->size(); s++){ // Para SRC s
191             rutasOffline[t][s] = std::vector<std::vector<AuxRuta *>>((*redRutas)[s].
    size());
192             for (int d = 0; d < (*redRutas)[s].size(); d++){ // Para DST d
193                 if (s == d) continue; // Si SRC = DST, saltar
194                 rutasOffline[t][s][d] = std::vector<AuxRuta *>();
195                 for (int r = 0; r < (*redRutas)[s][d].size(); r++){ // Para cada RUTA
     r entre s y d
196                     double LongitudTotal = 0;
197                     for (int l = 0; l < (*redRutas)[s][d][r].size(); l++)
198                         LongitudTotal += (*redRutas)[s][d][r][l]->getLongitud();
199                     for (int m = 0; m < tasasBits[t].getNumeroModulaciones(); m++){
200                         if (tasasBits[t].getAlcance(m) >= LongitudTotal){
201                             std::string modulacion_y_banda = tasasBits[t].
    getModulacion(m);
202                             std::string modulacion = modulacion_y_banda.substr(0,
    modulacion_y_banda.find('-'));
203                             modulacion_y_banda.erase(0, modulacion_y_banda.find('-')
    + 1);
204                             std::string banda = modulacion_y_banda.substr(0,
    modulacion_y_banda.find('-'));
```

```
                            rutasOffline[t][s][d].push_back(new AuxRuta((*redRutas)[s
    ][d][r],
                                                    tasasBits[t].getNumeroRanuras(m),
                                                    modulacion,
                                                    modulaciones[modulacion],
                                                    tasasBits[t].getAlcance(m),
                                                    bandas_v3_set2[banda],
                                                    banda,
                                                    tasasBits[t].getTasaBits(),
                                                    LongitudTotal,
                                                    r));
                        }
                    }
                }
                std::sort(rutasOffline[t][s][d].begin(), rutasOffline[t][s][d].end(),
     RBM_V);
            }
        }
    }
    return rutasOffline;
}
BEGIN_ALLOC_FUNCTION(Variant_3) {
  int numeroRanurasActuales;
  int indiceRanuraActual;
  int *indicesBandaRanura = NULL;
  int tasaBitsInt = tasasBits_map[TASA_BITS_SOLICITADA];
  int numeroRanuras;
  std::vector<AuxRuta *> *rutasRBMSA_tasaBits_set;
  std::vector<AuxRuta *> *rutasRBMSA_tasaBits = &rutasOffline[tasaBitsInt][SRC][DST];
  std::vector<bool> ranurasTotales;
  contadorTasaBitsTotal[tasaBitsInt] += 1;
  if (TASA_BITS_SOLICITADA < mediana_v3){
    rutasRBMSA_tasaBits_set = &rutasOffline_set1[tasaBitsInt][SRC][DST];
  } else rutasRBMSA_tasaBits_set = &rutasOffline_set2[tasaBitsInt][SRC][DST];
  for (int r = 0; r < (*rutasRBMSA_tasaBits_set).size(); r++){ // <- Para ruta r
    entre SRC y DST
    numeroRanuras = (*rutasRBMSA_tasaBits_set)[r]->getRanuras();
    ranurasTotales = std::vector<bool>((*rutasRBMSA_tasaBits_set)[r]->getEnlaces()
    [0]->getRanuras(), false);
    if (indicesBandaRanura != NULL) delete(indicesBandaRanura);
    parametrosBandaRanura((*rutasRBMSA_tasaBits_set)[r]->getEnlaces()[0]->getRanuras
    (),
                          (*rutasRBMSA_tasaBits_set)[r]->getBanda(),
                          &indicesBandaRanura);
    for (int l = 0; l < (*rutasRBMSA_tasaBits_set)[r]->getEnlaces().size(); l++){ //
    <- Para enlace l en ruta r
      for (int s = indicesBandaRanura[0]; s <= indicesBandaRanura[1]; s++){ // <-
    Para cada ranura s en la banda actual, del enlace l
        ranurasTotales[s] = ranurasTotales[s] | (*rutasRBMSA_tasaBits_set)[r]->
    getEnlaces()[l]->getRanura(s);
      }
    }
    numeroRanurasActuales = 0;
    indiceRanuraActual = indicesBandaRanura[0];
    for (int s = indicesBandaRanura[0]; s <= indicesBandaRanura[1]; s++){
      if (ranurasTotales[s] == false) numeroRanurasActuales++;
      else {
        numeroRanurasActuales = 0;
        indiceRanuraActual = s + 1;
      }
      if (numeroRanurasActuales == numeroRanuras) {
        for (int l = 0; l < (*rutasRBMSA_tasaBits_set)[r]->getEnlaces().size(); l++){
```

```
259           ASIGNAR_RANURAS((*rutasRBMSA_tasaBits_set)[r]->getEnlaces()[l]->getId(),
    indiceRanuraActual, numeroRanuras)
260         }
261         if (indicesBandaRanura != NULL) delete(indicesBandaRanura);
262         return ALLOCATED;
263       }
264     }
265   }
266   contadorTasaBitsBloqueado[tasaBitsInt] += 1;
267   if (indicesBandaRanura != NULL) delete(indicesBandaRanura);
268   return NOT_ALLOCATED;
269 }
270 END_ALLOC_FUNCTION
```