Universidad de la República
Facultad de Ingeniería

Py5cheSim v2.0: Extensión de Funcionalidades de un Simulador de Redes 5G para Ensayo de Asignación de Recursos

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Diego Sánchez*, Mateo Trujillo•, Paula Varela•

en cumplimiento parcial de los requerimientos para la obtención del título de *Ingeniero en Sistemas de Comunicación/•Ingeniero Electricista.

Tutor

Claudina Rattaro ..... Universidad de la República
Pablo Belzarena..... Universidad de la República
Lucas Inglés ..... Universidad de la República

Tribunal

Federico La Rocca ..... Universidad de la República
Juan Pablo González..... Universidad de la República
Martín Randall ..... Universidad de la República

Montevideo
lunes 3 abril, 2023

Py5cheSim v2.0: Extensión de Funcionalidades de un Simulador de Redes 5G para Ensayo de Asignación de Recursos, Diego Sánchez, Mateo Trujillo, Paula Varela.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 101 páginas.
Compilada el lunes 3 abril, 2023.
http://iie.fing.edu.uy/

Si he logrado ver más lejos ha sido porque he subido a hombros de gigantes.

Isaac Newton

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

A los tutores Pablo, Claudina y Lucas por guiarnos a lo largo de este proyecto. A Gabriela Pereyra por crear la primera versión de Py5cheSim sobre la cual basamos nuestro trabajo. También a los creadores de DeepMIMO que nos brindaron su soporte. Por último a nuestras familias que nos apoyaron en todo momento.

Este trabajo se hizo en marco del Proyecto de Investigación "Herramientas de simulación de redes móviles de futura generación" FVF-2021-128– DICYT. Fondo Carlos Vaz Ferreira, Convocatoria 2021, Dirección Nacional de Innovación, Ciencia y Tecnología, Ministerio de Educación y Cultura, Uruguay.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

En la actualidad las redes móviles están evolucionando para contemplar diferentes casos de uso. Esta evolución viene dada por la estandarización y despliegue de la tecnología 5G, la cual entre otras cosas, hace uso de diferentes configuraciones simultáneas a nivel de radio para lograr satisfacer dichos casos de usos. Esta pluralidad de configuraciones y servicios hace que la asignación de recursos se convierta en un aspecto central de esta tecnología, aspecto que queda por fuera del estándar. Esta libertad de implementación crea la necesidad de tener herramientas de evaluación de desempeño de diferentes algoritmos de asignación de recursos, para que investigadores y fabricantes de radio bases puedan evaluarlos de forma primaria, sin la necesidad de desplegarlo sobre una red real.

En este panorama surge Py5cheSim, un simulador de tráfico de redes 5G el cual es de código abierto, fácil de usar y está implementado en *Python*. Ésto lo convierte en una gran alternativa para aquellos usuarios que están investigando la aplicación de técnicas de *machine learning* para la asignación de recursos (algo indispensable por la variabilidad y la complejidad de los escenarios en este tipo de redes).

El objetivo de este proyecto es mejorar el modelo de canal y el soporte para MIMO (*Multiple Input Multiple Output*) masivo de Py5cheSim, mediante la integración del *framework DeepMIMO*, el cual es otro *software* gratuito e implementado en *Python*. La contribución principal de este proyecto es liberar una segunda versión de Py5cheSim con dichos aspectos mejorados. Con la nueva versión del simulador se tienen escenarios más realistas (con posibilidad de usuarios móviles) y se cuenta con un nuevo algoritmo de asignación de recursos disponible (algoritmo basado en MIMO). Es importante destacar que el nuevo desarrollo tiene compatibilidad total con la versión anterior.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Capítulo 1

Introducción

En este capítulo se explican generalidades de las redes de quinta generación y de las dos herramientas clave en nuestro trabajo: Py5cheSim y DeepMIMO. Estas últimas serán desarrolladas en el siguiente capítulo. Una vez dado el contexto y la motivación se presenta el objetivo principal, objetivos específicos y el alcance del proyecto.

1.1. Motivación y contexto

Las comunicaciones móviles han tenido un gran avance en los últimos 40 años. Desde la primera generación en la década de los 80, que permitía realizar una llamada telefónica inalámbrica, hasta la actualidad donde la conexión a Internet sin cables es cuestión de todos los días. Las tecnologías móviles han evolucionado a la par del ser humano y sus necesidades. En la última década nace la 5ª Generación de Tecnlogías Móviles, o 5G, que busca cubrir las exigencias y los requisitos de las comunicaciones inalámbricas actuales.

Para responder al crecimiento de la red telefónica tradicional y los problemas y exigencias que esto conlleva se definen tres escenarios de uso que se consideran en las recomendaciones IMT-2020 de ITU-R WP5D [5], teniendo en cuenta opiniones de la industria móvil, diferentes organizaciones regionales y de operadores. 5G ofrece los servicios de banda súper ancha eMBB (*Enhaced Mobile Broadband*), servicios que requieren muy baja latencia y alta confiablidad URLLC (*Ultra-reliable and low-latency communications*) y los servicios mMCT (*Massive Machine Type Communications*), que pueden observarse en la figura 1.1

A continuación, se detallan cada uno de estos casos de uso:

- **Enhanced Mobile Broadband (eMBB)**: dado que los servicios de banda ancha móvil son los principales motores para el uso de sistemas móviles como 3G, 4G y LTE, este escenario apunta a ser el de uso más importante. Debido a que la demanda crece continuamente y surgen nuevas áreas de aplicación, se establecen nuevos requisitos para lo que ITU-R denomina banda ancha móvil mejorada (eMBB). Cubren una amplia gama de casos de uso, incluidos puntos de acceso y cobertura de áreas amplias. El primero permite altas

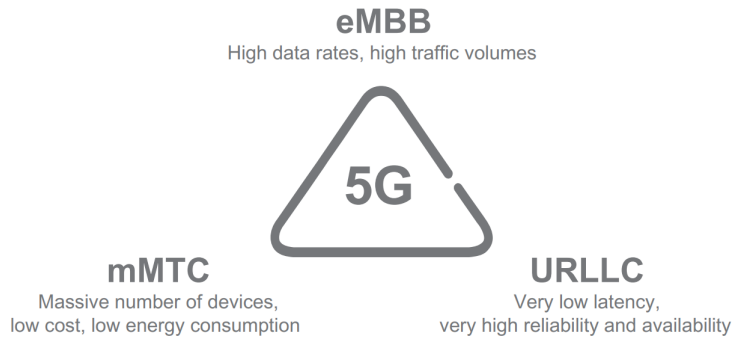

Figura 1.1: Clasificación de casos de uso de 5G. Figura extraída de [9]

velocidades de datos y alta densidad de usuarios, mientras que el segundo hace hincapié en la movilidad y la calidad de servicio mejorada, teniendo menores requisitos de velocidad de datos y densidad de usuarios.

- **Ultra-reliable and low-latency communications (URLLC)**: este escenario tiene como objetivo cubrir tanto la comunicación centrada en el humano como la comunicación centrada en la máquina, denominada critical machine-type communication (C-MTC). Se caracteriza por casos de uso con estrictos requisitos de baja latencia, alta confiabilidad y alta disponibilidad. Algunos ejemplos incluyen el control inalámbrico de equipos industriales, cirugías médicas remotas, comunicación vehículo a vehículo relacionada a la seguridad vial.

- **Massive machine-type communications (mMTC)**: este es un caso de uso puramente centrado en la máquina. Este tipo de comunicaciones se caracterizan por una gran cantidad de dispositivos conectados, que presentan pocas transmisiones y pequeños volúmenes de datos que no son sensibles a los retrasos. El desafío de la red se encuentra en soportar la cantidad total de dispositivos, pudiendo ser millones, conectados, cursando tráfico y consumiendo energía.

Para cumplir con estos requisitos 5G NR propone varias novedades, entre las cuales se destacan la operabilidad en bandas de ondas milimétricas, numerología variable, *network slicing* y MIMO (*Multiple Input Multiple Output*) masivo. La operabilidad en bandas de ondas milimétricas, frecuencias mayores a 24 GHz (rango de frecuencia 2 o FR2), se trae a consideración en 5G para satisfacer los requerimientos de tasa de datos extremas y la demanda en áreas con alto tráfico de datos [9]. Además, 5G NR soporta numerología variable de OFDM (*Orthogonal Frequency-Division Multiplexing*), con espacio variable entre subportadoras, esto es utilizar diferentes SCS (*sub-carrier spacing*) entre subportadoras OFDM con la finalidad de disminuir la duración de símbolo y en consecuencia también la duración de *slot*, el cual es la unidad mínima de asignación de recursos si no se aplica otra técnica para bajar la latencia. En la figura 1.2 se pueden ver las numerologías definidas en 5G junto con la duración de *slot* correspondiente. Por

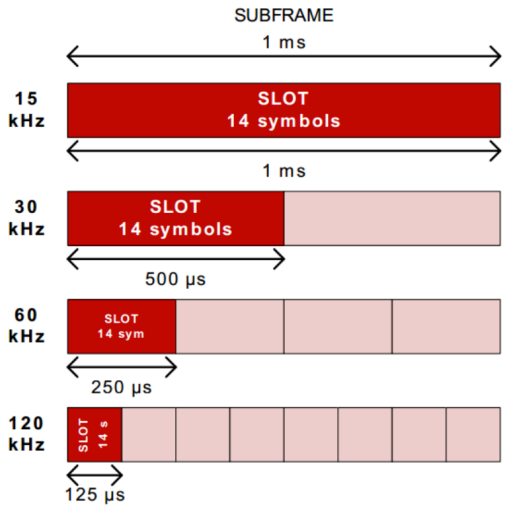

Figura 1.2: Numerologías definidas en 5G junto con la duración de *slot correspondiente*. Figura extraída de [14].

otra parte, una sola configuración de radio no es capaz de satisfacer los distintos casos de uso contemplados por la nueva tecnología. Por esta razón, en 5G surge el concepto de *network slicing*, el cual es implementado en varios niveles: por un lado, a nivel de núcleo de la red y red de transporte, en donde los flujos de paquetes son manejados de diferente manera dependiendo de la *slice* a la que pertenezcan, y por el otro, a nivel físico, donde a cada *slice* se le asigna una porción del espectro, y en esta, la comunicación es configurada según los requerimientos de la *slice*. A modo de referencia, en la tabla 1.1 se observa una comparación entre algunas de las las capacidades otorgadas por la tecnología 4G (LTE) y la tecnología 5G. Estas métricas serán posibles no sólo con las novedades mencionadas que trae 5G, sino que también con algoritmos inteligentes de asignación de recursos. Nuestro proyecto es una contribución en este sentido como se explicará más adelante.

Estos servicios y novedades introducidas por 5G llevan a explorar nuevos campos de trabajos. Entre ellos, la asignación de recursos realizada por las radiobases (*scheduling*), sobre todo, porque el estándar no define como debe ser la asignación de recursos, y la simulación orientada a esta asignación. La etapa de simulación es fundamental para todo usuario que desee probar el funcionamiento de un scheduler para 5G antes de desplegarlo en una red real, teniendo así un acercamiento al comportamiento que va a presentar. Así se puede conocer si el *scheduler* se comporta de manera esperada o detectar problemas en la etapa de diseño, sin necesidad de llegar a una etapa de producción.

Los simuladores de redes disponibles hasta el momento no soportan todas las funcionalidades y novedades mencionadas, ver un análisis detallado de las herramientas de simulación en [15], [16], por ello surgió en el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería, UdelaR, un proyecto de tesis de maestría enfocado en desarrollar un simulador para redes 5G: “Scheduling in 5G Networks: Developing a 5G Cell Capacity Simulator” [14] desarrollada por Gabriela Pereyra.

Tabla 1.1: Comparación IMT Advanced vs IMT 2020

	4G (LTE)	**5G**
Retardo (extremo a extremo)	30-50 ms (10 ms para RAN bidireccional (*pre-scheduled*); hasta 50 ms si otros factores son considerados (por ejemplo, transmisión, Internet, servidores *proxy*))	1ms
Velocidad	100 Mbps promedio, picos de 600 Mb/s	1Gbps
Movilidad	350km/h	500km/h
Cantidad de dispositivos conectados por km^2	10K	1000K

Esta tesis se enmarcó dentro del trabajo del grupo de investigación ARTES [2] y más específicamente dentro del proyecto de investigación FMV ANII Inteligencia Artificial aplicada a Redes 5G [13]. El simulador, Py5cheSim, se creó con el objetivo de tener una herramienta *open source* que soporte todas las funcionalidades de 5G. Otro de los cometidos de esta herramienta fue que permitiera ensayar el funcionamiento de un asignador de recursos en la etapa de desarrollo de forma sencilla. Esta última funcionalidad resulta de suma utilidad para operadores de redes celulares o investigadores que quieren comparar distintos algoritmos o probar nuevas propuestas en un entorno controlado y con características cercanas a una red 5G.

Py5cheSim v1.0 realiza simulaciones del comportamiento de una red 5G, tomando como entradas grupos de usuarios, su SINR (*Signal to Interference Noise Ratio*), su tipo de comunicación y otros parámetros. Como salida devuelve indicadores de performance. Este simulador tiene numerosas simplificaciones, que quitan realidad a las simulaciones realizadas. Una de ellas es el valor aleatorio del SINR de los usuarios del escenario sobre el que se va a simular. Este proyecto de grado elimina estas simplificaciones y agrega funcionalidades nuevas, para obtener una simulación más completa y realista.

El proyecto "DeepMIMO: A Generic Deep Learning Dataset for Millimeter Wave and Massive MIMO Systems" [3] desarrollado por la Universidad de Arizona, provee un *framework* para estudiar los canales entre usuarios y radiobases de distintos escenarios que incluye el trabajo. DeepMIMO provee un conjunto de escenarios con radiobases, equipos de usuarios (UEs) y otros parámetros configurables. La salida de DeepMIMO es un conjunto de condiciones del canal para cada par radiobase-UE, esto incluye la respuesta en frecuencia del canal, los ángulos de partida y llegada y otros datos de interés. En particular se utilizan las salidas de los escenarios de este *framework* para generar diferentes parámetros de entrada al

simulador desarrollado en este proyecto.

1.2. Objetivos

El proyecto tiene como objetivo trabajar sobre Py5cheSim 1.0 y extender sus funcionalidades. Como producto principal del proyecto se espera liberar una nueva versión del simulador. Con la nueva versión los escenarios a simular serán más realistas. Permitirán a los usuarios (investigadores, operadores de red) probar algoritmos que exploten otras características 5G, que en la versión 1.0 no son soportadas o están en una versión muy simplificada.

1.2.1. Objetivos específicos

- Estudiar el simulador Py5cheSim v1.0, lo que incluye el estudio de la documentación asociada, pruebas de ejemplos y casos de uso típicos.
- Estudiar las principales características de las redes 5G.
- Integrar el *framework* DeepMIMO para obtener métricas del canal en escenarios dinámicos, donde tanto agentes externos como el usuario tengan la posibilidad de moverse.
- Estudiar MIMO, extender las funcionalidades del simulador Py5cheSim para dar soporte a esta característica y verificar las mismas definiendo las métricas adecuadas.
- Estudiar asignación de recursos, implementar un *scheduler* que utilice las funcionalidades mejoradas.

1.2.2. Alcance

Este proyecto tiene como alcance comprender los principios de las redes 5G y entender el funcionamiento del simulador *Py5cheSim* en su versión 1.0. Además, trabajar en mejoras del simulador y obtener una nueva versión, en particular: el modelado del canal y el soporte para MIMO masivo. Por último, realizar un *scheduler* que utilice estas mejoras implementadas y verifique la funcionalidad del simulador.

1.3. Organización del documento

El resto del documento está organizado en los siguientes capítulos: en el capítulo 2 se introduce un marco teórico con definiciones y conceptos importantes para la lectura. Además, se profundiza en los antecedentes, explicando sus funcionamientos y sus características principales. En el capítulo 3 se presenta Py5cheSim v2.0 con las principales decisiones de diseño, mientras que en el capítulo 4 se muestran las diferentes verificaciones para comprobar el funcionamiento del simulador y del

scheduler implementado. Por último, en el capítulo 5 se presentan las conclusiones finales del proyecto y posible trabajo futuro.

Capítulo 2

Antecedentes y marco teórico

En este capítulo se profundiza en los proyectos que preceden a Py5cheSim v2.0 y se brinda un marco teórico en el que se basa el desarrollo de las mejoras del simulador. Primeramente se definen y explicitan algunos conceptos teóricos relevantes al proyecto. Si bien la teoría referente a redes 5G, canal inalámbrico y sistemas MIMO es amplia, en estas secciones se encontrarán fundamentos de la temática orientados a lo utilizado durante el diseño y desarrollo de Py5cheSim v2.0. Luego, se describe el funcionamiento del simulador Py5cheSim en su primera versión, se explica el procedimiento para realizar una simulación y se comentan las funciones más relevantes a este proyecto. Por último, se describe el *framework* DeepMIMO utilizado para simular el canal inalámbrico. Se detallan los parámetros de entrada para realizar una simulación, los parámetros de salida y se da una visión general de los escenarios disponibles.

2.1. Canal AWGN

En un canal AWGN (*Additive White Gaussian Noise*) la señal modulada $s(t)$ tiene un ruido $n(t)$ añadido antes de la recepción. Este ruido $n(t)$ se modela como un proceso aleatorio gaussiano con media cero y densidad espectral de potencia $N_0/2$. La señal recibida es entonces $r(t) = s(t) + n(t)$. [11]

Se define la relación señal a ruido (SNR, *Signal to Noise Ratio*) como el cociente entre la potencia de la señal recibida y la potencia del ruido, dentro del ancho de banda de la señal transmitida $s(t)$. La potencia recibida P_r depende de varios parámetros, entre ellos la potencia transmitida, el *path loss*, el desvanecimiento por multicaminos. La potencia de ruido está determinada por el ancho de banda de la señal transmitida y las propiedades espectrales de $n(t)$. Si el ancho de banda de la señal transmitida $s(t)$ es $2B$, dado que el ruido $n(t)$ tiene una densidad espectral de potencia uniforme $N_0/2$, la potencia de ruido total dentro del ancho de banda es $N = N_0/2 \times 2B = N_0 B$. Por lo tanto la ecuación de SNR se escribe como

$$SNR = \frac{P_r}{N_0 B}. \tag{2.1}$$

En las redes 5G se utiliza tecnología de acceso OFDM (*Orthogonal Frecuency-Division Multiplexing*), debido a la robustez ante la dispersión en el tiempo y la facilidad de la explotación tanto en el dominio del tiempo como en el dominio de la frecuencia cuando se definen las estructuras para los diferentes canales. La ventaja de la modulación en subportadoras es que el ancho de banda de cada subcanal es relativamente angosto [11]. En particular, si la potencia de transmisión en la subportadora i es P_i, el desvanecimiento en esa subportadora es α_i y B el ancho de banda de cada subcanal, entonces la SNR recibida en cada subportadora es

$$SNR_i = \frac{P_i \alpha_i^2}{N_0 B}. \tag{2.2}$$

2.2. Tecnologías de radio 5G

Como se comentó en el capítulo 1, las redes 5G introdujeron diferentes novedades para satisfacer los requisitos propuestos. A continuación se detallan dos relevantes para este proyecto: la numerología variable en conjunto con la estructura de la trama en frecuencia y los sistemas de MIMO masivo.

2.2.1. Estructura de la trama en frecuencia

Un RE, *resource element*, es el recurso físico más chico en NR, y consiste de una subportadora durante un símbolo OFDM, y 12 subportadoras consecutivas en el dominio de la frecuencia se denominan un *resource block*. En la figura 2.1 se observa una descripción gráfica de estos recursos. La diferencia con la definición de *resource block* en LTE es que en NR se define solamente en el dominio de la frecuencia, mientras que en LTE se define como 12 subportadoras en frecuencia durante un *time-slot*.

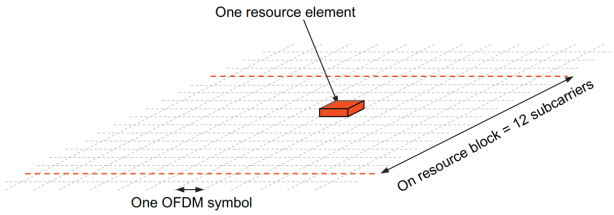

Figura 2.1: *Resource element* y *resource block* en NR. Figura extraída de [9].

Se mantiene la estructura de PRB (*Physical Resource Block*), pero como se comentó anteriormente, NR permite espaciados flexibles entre subportadoras, por lo que cambia la duración en tiempo y el largo en frecuencia de un PRB, como se observa en la figura 2.2. En la figura 2.2 se observan tres separaciones entre subportadoras diferentes, definidas como: $\Delta f = 2^{\mu} \times 15$ kHz, donde μ puede tomar el valor de 0, 1, 2, 3, 4 y determina la configuración del espaciado entre subportadoras.

Una portadora en NR puede tener hasta 275 *resource blocks*, lo que corresponde a $12 \times 275 = 3300$ subportadoras utilizadas. Como el ancho de banda por

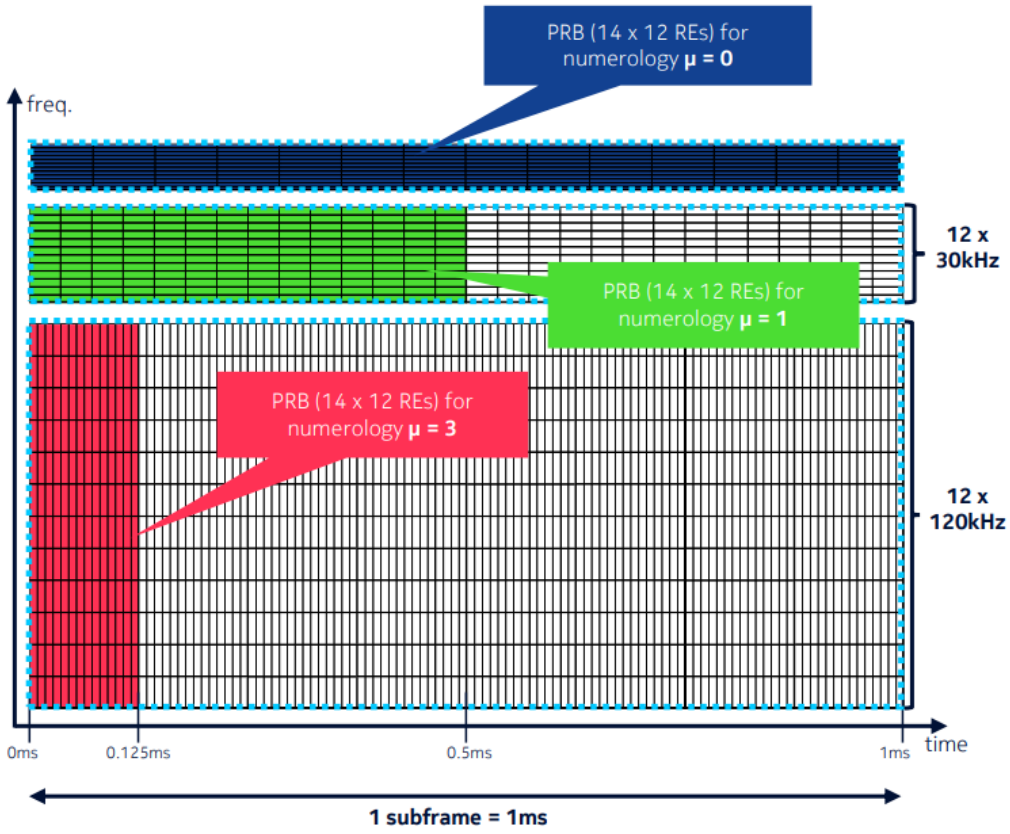

Figura 2.2: Estructura de un PRB para diferentes espaciados entre subportadoras. Figura extraída de [6].

portadora está limitado por 400 MHz, también están limitados los anchos de banda máximos para cada espacio entre portadora definidos, resultando en anchos de banda máximos de 50/100/200/400 MHz para $\Delta f = 15/30/60/120$ kHz. [9]

2.2.2. Sistemas MIMO Masivo

La transmisión/recepción multiantena es una herramienta fundamental de las redes 5G para permitir diversidad, directividad y multiplexación espacial, permitiendo altas velocidades de datos y una alta eficiencia del sistema. Esta es una característica heredada de las redes 4G, la novedad ahora es que al operar en bandas milimétricas se puede hablar de MIMO[1] masivo.

En estos sistemas donde los dispositivos se encuentran en diferentes direcciones relativas a la radiobase y los mismos moviéndose en diferentes direcciones, el uso de antenas dirigidas como se hacía hasta el momento en los sistemas de comunicación no es aplicable. Sin embargo, los sistemas de MIMO masivo logran un efecto similar utilizando paneles de antenas con un gran número de pequeñas antenas. En

[1]En el anexo A.1.1 se presentan algunos conceptos básicos sobre los sistemas MIMO.

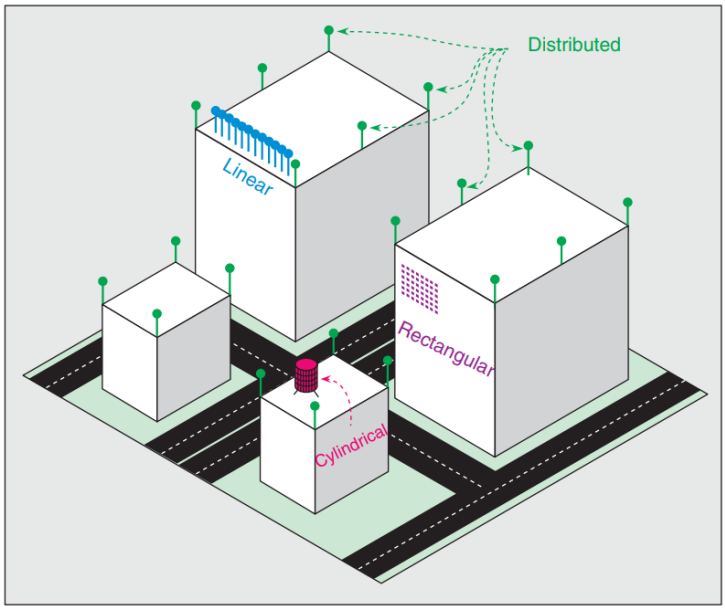

Figura 2.3: Algunas posibles configuraciones de paneles de antenas para radiobases que implementan MIMO masivo. Figura extraída de [8].

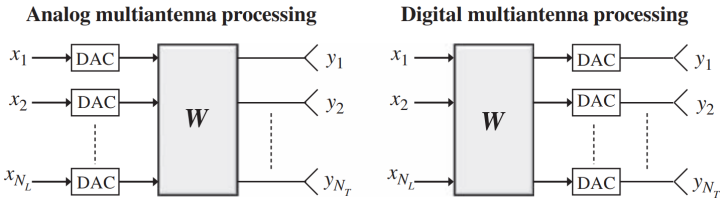

Figura 2.4: Procesamiento analógico vs digital de multi-antena. Figura extraída de [9].

la figura 2.3 se pueden observar diferentes configuraciones que pueden tomar los paneles de antenas en sistemas de MIMO masivo tales como: lineal, rectangular, cuadrada o cilíndrica.

En general, el esquema de un sistema de multi-antena de transmisión se puede modelar como el vector x multiplicado por una matriz W de tamaño $N_T \times N_L$. Según la implementación, existen diferentes grados de restricciones y capacidades de la transmisión. En particular, donde se aplica la matriz W de procesamiento depende de la implementación. En la figura 2.4 se pueden distinguir dos casos:

- El procesamiento se aplica antes de la conversión de digital a analógico, en la parte digital de la transmisión.
- El procesamiento se aplica después de la conversión de digital a analógico, en la parte analógica de la transmisión.

El procesamiento analógico implica que no es posible multiplexar en frecuencia las transmisiones mediante formación de haces, ya que no es posible transmitir a dispositivos en diferentes ubicaciones. Por otro lado, el procesamiento digital,

permite procesar diferentes señales en la misma portadora, lo que posibilita transmisiones simultaneas a diferentes dispositivos. A continuación se detalla más acerca del procesamiento digital y el procesamiento analógico.

Procesamiento digital

En el caso del procesamiento digital, la matriz W es muchas veces denominada como matriz de precodificación o de *precoding* y el procesamiento multi-antena *precoding*. La implementación digital de MIMO masivo soporta la recepción de múltiples *layers* en paralelo.

Uno de los aspectos importantes del *precoding* es que las señales de demodulación de referencia (DMRSs, *Demodulation Reference Signals*) son precodificadas de igual forma que los datos. Ésto hace que la precodificación pueda verse desde el lado del receptor como parte de un canal multidimensional. Es decir, en vez de ver la matriz H de canal "verdadera", el receptor verá una matriz de canal H' que es la concatenación de la matriz H con la matriz de *precoding* W. Por lo tanto el transmisor puede elegir una matriz de *precoding* arbitraria sin necesidad de avisarle al receptor sobre la misma.

Las especificaciones del *precoding* de *downlink* están determinadas entonces por las medidas y reportes del usuario a la radiobase. Estas medidas del UE (*User Equipment*) forman parte del CSI (*Channel State Information*) y pueden contener: indicador de rango (RI, *Rank Indicator*), indicador de matriz de precoding (PMI, *Precoder-Matrix Indicator*) e indicador de calidad de canal (CQI, *Channel-Quality Indicator*). El RI indica la tasa de transmisión que soporta, es decir la cantidad de capas N_L de transmisión para la comunicación de *downlink*. El PMI informa a la radiobase el indice de la matriz de precoding del codebook que prefiere utilizar, dado el rango seleccionado. Por último, el CQI le informa a la radiobase la calidad del canal dada la matriz de *precoding* seleccionada, por lo tanto la tasa y la modulación para transmitir que el UE cree satisfactoria.

Como se menciona, el PMI indica la matriz de *precoding* escogida por el UE. Cada valor de PMI corresponde a una única matriz de *precoding*. El conjunto de PMI posibles corresponde a un conjunto de matrices de *precoding* denominado *precoder codebook*. Como el UE selecciona el PMI teniendo en cuenta el número de antenas N_T y el rango seleccionado N_L, existe al menos un *codebook* para cada combinación válida $N_T \times N_L$. También es necesario aclarar que el *codebook* es utilizado en el contexto del reporte del PMI, pero no impone restricciones a la radiobase sobre la matriz que va a utilizar finalmente. Aunque parezca lógico que se utilice la matriz reportada por el PMI, en ocasiones la radiobase tiene otra información que termina influyendo en la selección de otra matriz de *precoding*. Por ejemplo, en MU-MIMO no solo se busca elegir la matriz de *precoding* que enfoque la energía hacia el UE, sino también la que limite la interferencia con otros *scheduled* UEs en el mismo instante. Por lo tanto la radiobase no solo debe tener en cuenta el PMI reportado por el UE, sino también tener en cuenta el PMI reportado por todos los UEs que necesitan recursos en ese instante de tiempo.

NR define dos tipos de CSI, *Type I CSI* y *Type II CSI*, que difieren en la estructura y el tamaño del *codebook*. A continuación se detallan generalidades de

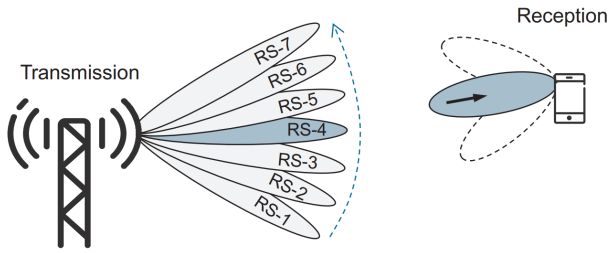

Figura 2.5: Ajuste de *beam* de transmisión. Figura extraída de [9].

ambos tipos, se puede leer más acerca de ellos en las secciones 11.2.1 y 11.2.2 de [9].

- ***Type I CSI***: utilizado principalmente para escenarios *Single User* MIMO (SU-MIMO), donde a un usuario se le asigna un recurso en tiempo/frecuencia, con la transmisión de varias capas en paralelo (multiplexación espacial alta).
- ***Type II CSI***: utilizado en escenarios *Multiple User* MIMO (MU-MIMO), con múltiples UEs a los que se le asignan recursos simultaneamente en el mismo tiempo y frecuencia, pero con una limitada cantidad de capas de transmisión, máximo dos, por UE *scheduled*.

Procesamiento analógico

En el caso de operaciones a altas frecuencias con un gran número de antenas como se trata en MIMO masivo, el procesamiento es más común que se haga de manera analógica con el foco en formación de haces (*beam-forming*). Esto implica que las transmisiones pueden realizarse en una dirección a la vez y además, las transmisiones de *downlink* a diferentes UEs ubicados en zonas diferentes relativas a la radiobase deben realizarse separadas en el tiempo. Analogamente, el receptor solo puede enfocarse en una dirección a la vez. Por lo tanto, lo que se busca es establecer un *beam pair* que provea buena conectividad durante la comunicación.

Una vez establecido el *beam* inicial de comunicación se busca refinarlo. Para ello, el UE puede medir un conjunto de RS correspondientes a diferentes *beams* de *downlink*, como se observa en la figura 2.5 y reportarlos a la radiobase, para que decida ajustar el haz utilizado.

El conjunto de RSs correspondientes al conjunto de *beams* deben incluirse en el recurso de reportes NZP (*Non-Zero-Power*) CSI-RS. El dispositivo puede reportar hasta cuatro señales de referencias (CSI-RS) en una instancia. Cada reporte incluye:

- Indicador de qué RS, es decir qué *beam* se está reportando.
- La medida de LI-RSRP (*Layer Indicator-Reference Signal Received Power*) para el *beam* más fuerte.
- Para el resto de los tres *beams* la diferencia entre los LI-RSRP medidos y el LI-RSRP del *beam* más fuerte.

Para leer más sobre *beam managment* se puede acceder al capítulo 12 de [9].

2.3. Py5cheSim

Py5cheSim es un simulador para redes 5G desarrollado en Python, flexible y de código abierto. El mismo está basado en Simpy, el cual es un *framework* para la simulación de eventos discretos y es utilizado para simular el envío y recepción de paquetes. Su objetivo es generar métricas de rendimiento a nivel de celda sobre *schedulers* implementados por el usuario en una configuración establecida, evitando así la necesidad de llevarlo a un ambiente real. El mismo está enfocado en la asignación de recursos entre diferentes usuarios y servicios, con diferentes requerimientos.

Este simulador surge principalmente por dos razones: primero, y como se explicó en el capítulo anterior, 5G define una variedad de casos de uso que no pueden ser satisfechas por una única configuración de red, lo que genera que el papel del *scheduler* en la asignación de recursos ocupe un lugar relevante en el diseño de una red 5G. Por otro lado está la falta de herramientas gratuitas y sencillas que cumplan el mismo objetivo.

2.3.1. Características principales

A continuación se explican las características principales de Py5cheSim, las cuales también son relevantes para este proyecto.

Network Slicing

En este simulador, RAN (*Radio Acces Network*) Slicing es implementada mediante el uso de dos tipos de *schedulers* diferentes: uno *inter slice scheduler* y varios *intra slice schedulers*. La función del *inter slice scheduler* es asignar recursos a las diferentes *slices*. Por otro lado, la función del *intra slice schedulers* es asignar recursos a los diferentes usuarios pertenecientes a una *slice*.

Las *slices* son creadas con la finalidad de configurar los parámetros de la comunicación según los requerimientos del grupo de usuarios a servir. Estos requerimientos son: retardo, banda de operación, cantidad de UEs a servir, perfil de tráfico de paquetes y capacidades de los UEs. Por otro lado, los parámetros de la comunicación que se pueden configurar son los siguientes: numerología, SCS (*Sub-Carrier Spacing*), TTI (*Transmission Time Interval*), modo de duplexación y el algoritmo de asignación de recursos.

Soporte de múltiples numerologías

Como se comentó en el capítulo 1, uno de los mecanismos contemplados en 5G para soportar comunicaciones de baja latencia es el uso de diferentes numerologías. El simulador soporta todas las numerologías definidas en 5G y la misma es elegida al momento de crear la *slice* según el requerimiento de retardo que tenga.

FDD/TDD Frame

Py5cheSim soporta ambos métodos de duplexación: FDD (*Frequency Division Duplex*) y TDD (*Time Divison Duplex*). Para el caso de ondas milimétricas, es decir, cuando la frecuencia central es mayor a 6 GHz, también llamado $FR2$ [14], solamente se puede utilizar TDD. Luego, para el caso de $FR1$ se debe especificar el modo de duplexación a utilizar, y el simulador no verifica la relación entre este y la banda utilizada. Esto se hace con el objetivo de permitir más escenarios que los definidos por el estándar.

Soporte para Uplink/Downlink

El simulador soporta ambas direcciones para el flujo de paquetes entre un UE y la radiobase, pero tiene la limitante que para cada UE definido solo se puede tener un tipo de flujo.

SU/MU-MIMO

Py5cheSim soporta ambos modos para MIMO con algunas simplificaciones. Para el caso de SU-MIMO, la cantidad de RB asignados a un UE es multiplicada por la cantidad de *layers* especificadas al momento de crear el grupo de usuarios. Por otro lado, para MU-MIMO se mantiene la cantidad de RB asignados a un UE, pero con la diferencia de que hay más cantidad de RB disponibles para asignar.

Esto es una simplificación grande debido a que en realidad, la cantidad de *layers* que se puede asignar a un UE, o el uso de MU-MIMO, dependen de las condiciones del canal entre los UEs y la radiobase. Este proyecto de grado trabaja en este aspecto.

Perfiles de tráfico

El simulador soporta la definición de distintos perfiles de tráfico por grupo de usuarios. Por defecto, tanto el tamaño de los paquetes, como el tiempo entre los mismos, siguen la distribución de Pareto, aunque es posible definir otras distribuciones modificando los métodos que se encargan de obtener el tamaño de los paquetes y el tiempo entre los mismos.

Generación de SINR

Py5cheSim no tuvo como objetivo generar un modelo de canal realista para las simulaciones, es por esta razón que los UEs se incializan con un valor de SINR entre 0 dB y 40 dB, el cual puede ser especificado o no en el archivo de configuración, y luego durante la simulación sufre pequeñas variaciones. En caso de no ser especificado se sortea un valor aleatorio entre 0 dB y 40 dB.

Por otro lado, no se hace una distinción a nivel de PRB sobre la calidad del canal, sino que se utiliza un único valor de SINR para toda la banda. Además, los valores de SINR no varían según la carga en la celda, es decir, se supone que la

comunicación con otros UEs no interfieren. Como se comentó anteriormente, este proyecto de grado busca darle mayor realidad a este aspecto.

Implementaciones de schedulers

El simulador cuenta con algunas implementaciones de *inter slice* e *intra slice schedulers*. Para ambos tipos de *schedulers* se encuentran implementados los algoritmos de Round Robin y Proportional Fair, aunque la característica principal en este punto es que se pueden crear nuevos *inter slice* e *intra slice schedulers* heredando de las clases ya definidas y sobrescribiendo los métodos de interés.

2.3.2. Pasos para realizar una simulación

Todo lo necesario para ejecutar una simulación debe ser especificado en el archivo *simulation.py*. A grandes rasgos, en este archivo se deben instanciar dos clases diferentes: *Cell* y *UEgroup*. La primera es el modelo que representa la celda y para instanciarla se deben especificar los siguientes parámetros: ancho de banda, rango de frecuencia, tamaño del *buffer* de la *bearer*, modo de duplexación, tiempo entre asignaciones de recursos a nivel de *inter slice scheduler* y por último se debe especificar el *inter slice scheduler* que se va a utilizar.

Por otro lado, para instanciar *UEgroups* se debe indicar la cantidad de UEs de *downlink* y *uplink*, el perfil de tráfico para UEs en ambas direcciones, la latencia máxima soportada, el *intra slice scheduler* a utilizar para la slice que se genera para ese grupo, el modo de MIMO y por último, la celda a la cual pertenece el grupo de UEs.

Luego de modificar el archivo *simulation.py* con los datos definidos, solamente se debe ejecutar el *script*. Al finalizar la simulación, se generan gráficas mostrando la performance de los *schedulers* en las diferentes *slices*.

2.3.3. Operación básica

En la figura 2.6 se esquematiza el manejo de los paquetes que realiza el simulador Py5cheSim. Primero, con los datos de perfil de tráfico brindados al instanciar el *UEgroup*, es que el método *queueAppPckt* la clase *PacketFlow* es capaz de generar paquetes que sigan dicho perfil para cada UE del grupo. Estos paquetes se almacenan en la cola *appBuffer* del UE. Una vez que el UE se conecta a la celda, el método *receivePckt* de la clase *UE* toma los paquetes del *buffer* de aplicación y los almacena en la cola *BearerBuffer* perteneciente al *UE*.

Luego de la asignación de recursos realizada en cada TTI por los *intra slice schedulers*, el método *queueUpdate* del *intra slice scheduler* toma paquetes desde el *BearerBuffer* de los UEs y los transforma en *TransportBlocks* asignando un MCS apropiado, obteniendo de esta manera una cantidad determinada de bits para dicho UE en ese TTI.

Por último, en cada TTI, el método *queuesOut* del *intra slice scheduler* toma todos los *TransportBlocks* del *buffer* del *scheduler* y los envía por el canal físico.

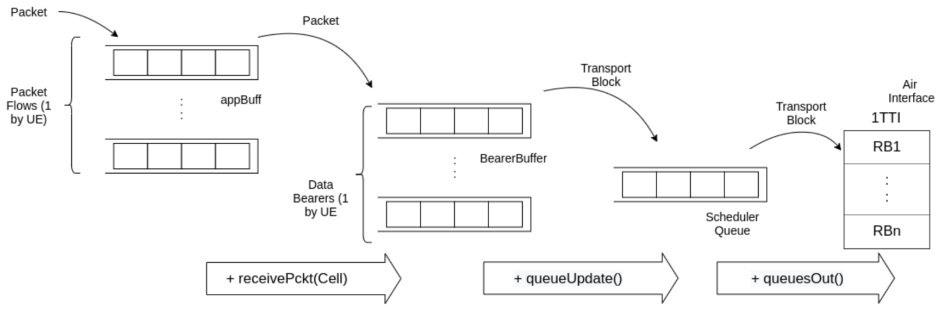

Figura 2.6: Esquema de manejo de paquetes que realiza Py5cheSim. Figura extraída de [14].

En este punto el envío puede fallar con probabilidad BLER, y en ese caso se vuelve a encolar el *TransportBlock*.

2.3.4. Aspectos para mejorar

En la documentación del simulador Py5cheSim [14] se explicitan varios aspectos del simulador que pueden ser extendidos con el objetivo de lograr una simulación más realista. Entre ellos se encuentran el modelado del canal inalámbrico y el soporte para MIMO, los cuales toman un lugar protagónico en la segunda versión del mismo. A continuación se explican cada uno de ellos en más detalle.

Modelado del canal inalámbrico

Py5cheSim utiliza un único valor de SINR para modelar el canal entre un UE y la radiobase, ignorando de esta manera que el canal puede tener diferentes calidades en los distintos PRBs de la banda. Por otro lado, dicho valor de SINR es establecido en el archivo de configuración o elegido aleatoriamente entre 0 dB y 40 dB para cada UE en el grupo. Esta característica quita realismo a la simulación, debido a que dicho valores provienen de UEs inmóviles, y por lo tanto no describen un escenario real.

Soporte para MIMO

El simulador permite definir en el archivo de configuración si el grupo utiliza MU-MIMO o SU-MIMO, junto con la cantidad de *layers* en el caso de SU-MIMO. Luego, durante la simulación, la cantidad de RB asignados al UE es multiplicada por la cantidad *layers* en caso que sea SU-MIMO, o se aumenta la cantidad de RB a asignar en caso que sea MU-MIMO. Esta implementación es solamente una aproximación y deja lugar a mejoras. En una situación real, el uso de MIMO junto con la cantidad de *layers* asignadas y los RB disponibles depende de las condiciones del canal.

2.4. DeepMIMO

Para mejorar la funcionalidad de canal inalámbrico en Py5cheSim, se buscó integrarlo con un *framework* que simulara el canal inalámbrico y brindara una SINR más realista. Para ello se evaluaron varias opciones que se explicaran a continuación.

El *software* de predicción inalámbrica Wireless Insite® [1] fue el primero estudiado. Este *software* proporciona predicciones de las características del canal de comunicación y modelos de propagación empírica en entornos urbanos, rurales, internos y de caminos mixtos. Este *software* fue descartado rápidamente ya que uno de los objetivos planteados fue que el simulador sea *opensource* y Wireless Insite®es un programa pago.

El segundo *framework* estudiado fue 'ViWi: A Deep Learning Dataset Framework for Vision-Aided Wireless Communications' [4]. Este *framework* genera un conjunto de datos de canal inalámbrico utilizando modelado 3D y *software* de *ray-tracing*. Si bien ViWi otorga los datos requeridos, se descartó rápidamente ya que es necesario utilizar el programa MatLab, lo que nuevamente incumple el objetivo de que el simulador sea *opensource*. Además, ViWi tiene un alto costo computacional ya que hace uso de una gran cantidad de imágenes y datos visuales, no necesarios para el desarrollo de este proyecto.

A través de ViWi, se llegó al *framework* 'DeepMIMO: A Generic Deep Learning Dataset for Millimeter Wave and Massive MIMO Applications' [3] de la ASU (Arizona State University). Este es un generador de datos para canales de banda milimétrica con soporte para MIMO masivo. A diferencia de ViWi, se puede ejecutar tanto en MatLab como en Python. Los canales de DeepMIMO están construídos a través de datos de *ray-tracing*, obtenidos del simulador Wireless InSite desarrollado por la empresa Remcom. Estos escenarios de *ray-tracing* están disponibles de manera gratuita en la página de DeepMIMO. Los datos que otorga el generador DeepMIMO quedan determinados dado el conjunto de parámetros a setear y el escenario de *ray-tracing*. Éstos puntos llevan a que DeepMIMO permita reproducir diferentes conjuntos de datos para poder realizar comparaciones y evaluaciones de diferentes desarrollos académicos. Se concluyó que el *framework* DeepMIMO resultaba beneficioso para el proyecto y fue el elegido.

Los escenarios disponibles son seis, tres exteriores y tres interiores. Estos escenarios son brindados por DeepMIMO, ya que como se explicitó anteriormente, Wireless Insite es un programa pago. En la tabla 2.1 se observan las principales características de los escenarios disponibles en DeepMIMO.

Uno de los escenarios más utilizados para probar las implementaciones realizadas fue el 'I2 (Indoor 2) Blockage Scenario'. Éste se trata de una habitación de 7 m $\times$ 13 m $\times$ 3 m con un bloqueo de la línea de vista entre la radiobase y los usuarios, que pueden llegar a ser ciento cuarenta mil. La frecuencia a la que opera este escenario es de 28 GHz. En la figura 2.7 se muestra una visión de planta del escenario y en la figura 2.8 una vista 3D del mismo.

Los parámetros de entrada al generador DeepMIMO para seleccionar, antes de correr el código, son varios. Existe la posibilidad de cargarlos de manera predeter-

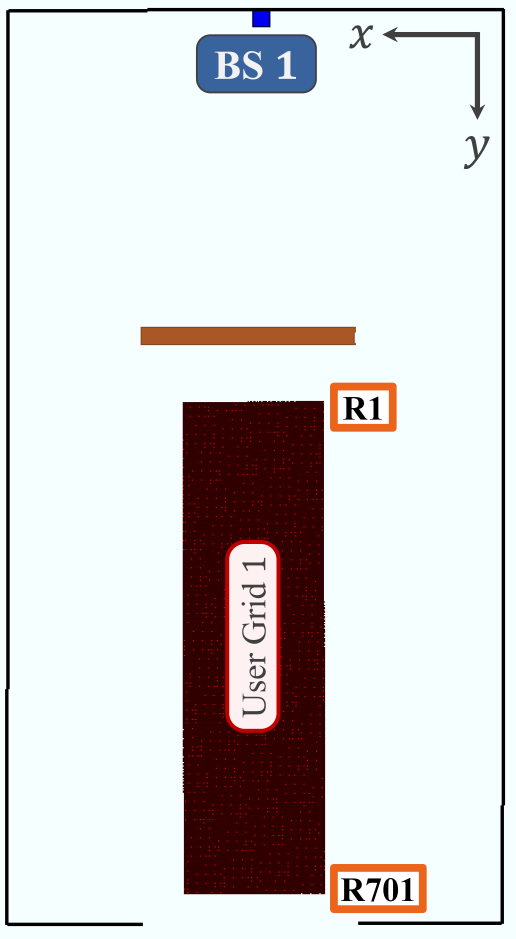

Figura 2.7: Visión superior del escenario 'I2'. Figura extraída de [3]

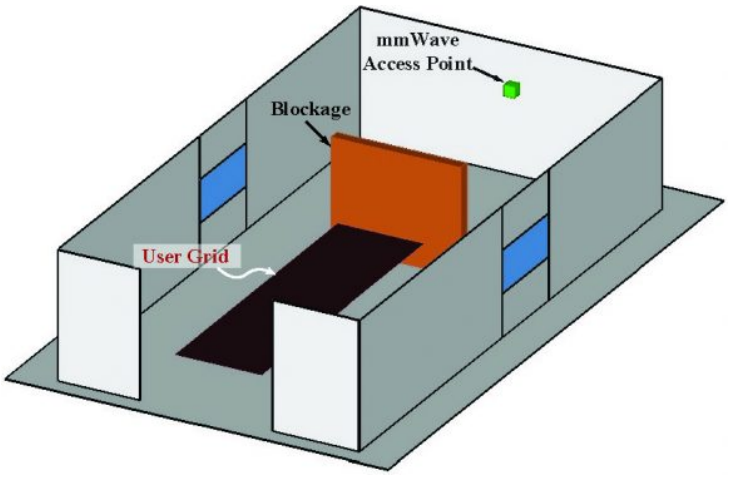

Figura 2.8: Visión 3D del escenario 'I2'. Figura extraída de [3]

Tabla 2.1: Principales características de los escenarios disponibles.

Escenario	Frecuencias de operación	Usuarios	Radiobases
O1 (Outdoor 1)	3.4 GHz - 3.5 GHz - 28 GHz - 60 GHz	1 millón	18
O1 Blockage (Outdoor 1 Blockage)	3.5 GHz - 28 GHz	490 mil	1
O1_drone (Outdoor 1 Drones and Flying RIS)	200 GHz	270 mil drones	1 - 1 RIS
O2 (Outdoor 2)	3.4 GHz - 3.5 GHz	115 mil	2
I1 (Indoor 1)	2.4 GHz - 2.5 GHz	150 mil	64 antenas
I2 (Indoor 2) Blockage	28 GHz	140 mil	1
I3 (Indoor 3)	2.4 GHz - 60 GHz	118 mil	2

minada. Los valores que toman se pueden observar en la figura 2.9. Los parámetros de interés para este proyecto son el escenario elegido, el *bandwidth* del canal, las radiobases activas, la cantidad de antenas por radiobase, la cantidad de usuarios del escenario y las subportadoras de OFDM.

Los parámetros de salida de DeepMIMO corresponden a variables del canal entre la radiobase i y el usuario j. A continuación se detallan:

- Channel Matrix: `dataset[i]['user']['channel'][j]`.

  Se trata de la matriz de estado del canal que otorga para cada radiobase y usuario, el valor de la transferencia del canal.

- Ray-tracing Path Parameters: `dataset[i]['user']['paths'][j]`

  Este parámetro es un diccionario, que para cada camino contiene información de: ángulo azimutal y cenital de llegada y salida, tiempo de llegada, fase, potencia, *delay spread*, número de caminos y caminos activos.

- Line-of-Sight Status: `dataset[i]['user']['LoS'][j]`

  Es una variable que puede tomar valores de {-1, 0, 1} e indica la existencia de línea de vista.

- TX-RX Distance: `dataset[i]['user']['distance'][j]`

  La distancia euclidea entre las ubicaciones de RX y TX en metros.

- Path loss: `dataset[i]['user']['pathloss'][j]`

  El path-loss combinado del canal entre RX y TX, en dBm.

```
{parameters} = {
    'dataset_folder': './Raytracing_scenarios',
    'scenario': 'O1_60',
    'dynamic_settings': {
        'first_scene': 1,
        'last_scene': 5
    },
    'num_paths': 10,
    'active_BS': np.array([1, 2]),
    'user_row_first': 1,
    'user_row_last': 2,
    'row_subsampling': 1,
    'user_subsampling': 1,
    'enable_BS2BS': 1,
    'OFDM_channels': 1,
    'OFDM': {
        'subcarriers': 512,
        'subcarriers_limit': 32,
        'subcarriers_sampling': 1,
        'cyclic_prefix_ratio': 0.1,
        'bandwidth': 0.05,
        'pulse_shaping': 0,
        'rolloff_factor': 0.5,
        'upsampling_factor': 50.0
    },
    'low_pass_filter_ideal': 1,
    'bs_antenna': {
        'shape': np.array([ 1, 32,  1]),
        'spacing': 0
    },
    'ue_antenna': {
        'shape': np.array([1, 1, 1]),
        'spacing': 0.5
    }
}
```

Figura 2.9: Parámetros por defecto de DeepMIMO.

- User location: `dataset[i]['user']['location'][j]`

  La ubicación euclídea del usuario de la forma [x, y, z]

- Basestation location: `dataset[i]['location']` La ubicación euclídea de la radiobase de la forma [x, y, z]

2.5. Resumen del capítulo

En este capítulo se expuso un breve resumen de los principales conceptos teóricos de telecomunicaciones y 5G NR aplicados a este proyecto de grado. Además, se introdujo el simulador Py5cheSim en su primera versión y el *framework* DeepMIMO que se integra al simulador en esta nueva versión.

A continuación se desarrollará sobre el diseño y la implementación de la solución del simulador Py5cheSim en su versión 2.0 con las mejoras de canal y soporte para MIMO masivo mencionadas anteriormente.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Diseño y desarrollo de la solución

En este capítulo se explican las mejoras que se le realizaron al simulador Py5cheSim. Estas mejoras se pueden dividir en tres bloques:

- **Adquisición de datos de un escenario:** En la primer versión del simulador, los datos del estado del canal de los distintos usuarios consistía únicamente en el SNR de estos últimos. Además este valor de SNR era sorteado utilizando una distribución Gaussiana [14]. En la nueva versión se consideran más datos para el estado del canal de los usuarios, para ello se utilizan escenarios configurables donde se pueden extraer datos relevantes para la simulación.
- **Simulación de comunicaciones MIMO:** En la nueva versión del simulador se puede simular el comportamiento de una red cuando los usuarios utilizan MIMO.
- **Asignación de recursos:** En la nueva versión del simulador se puede hacer uso de un nuevo scheduler que asigna recursos a los usuarios teniendo en cuenta que los usuarios pueden estar utilizando MIMO.

3.1. Arquitectura general del sistema

El sistema construido tiene un funcionamiento que se puede resumir en la figura 3.1. En primer lugar se selecciona el escenario perteneciente a DeepMIMO con el que se desea trabajar. Una vez realizado lo anterior se ejecuta el *script* `channel_generator.py`, en esta etapa se seleccionan las características del escenario, los usuarios que lo componen y de la simulación que se va a realizar. Del escenario, se selecciona la frecuencia central y el ancho de banda disponible. En cuanto los usuarios se selecciona su cantidad de antenas y las características de movimiento a lo largo de la simulación. Por último se selecciona la duración de la simulación y la cantidad de escenas con la que cuenta la simulación. Al ejecutar el *script* con los distintos parámetros seleccionados, este devuelve como salida una carpeta con las características de las condiciones físicas de los usuarios. El simulador hace uso de esta carpeta y de un asignador de recursos a elección del usuario

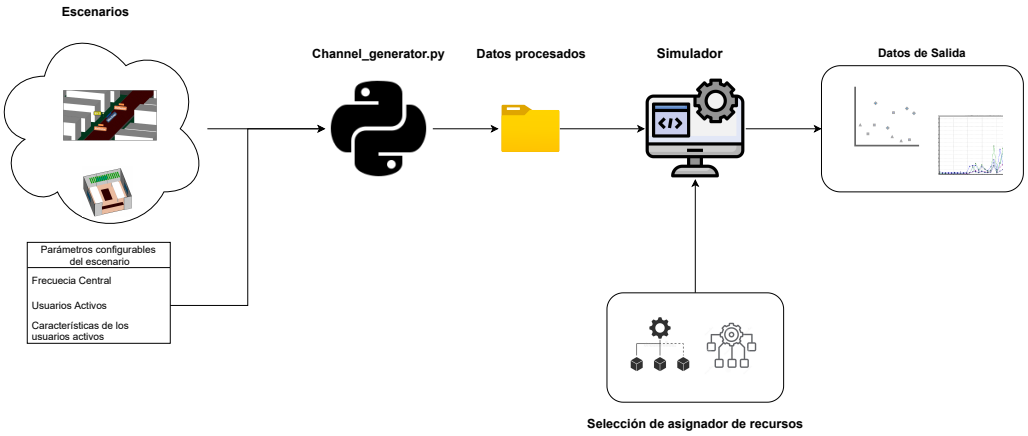

Figura 3.1: Diagrama del funcionamiento general del sistema.

para realizar la simulación. Como salida el simulador devuelve distintos resultados en forma de gráficas en función del tiempo, entre ellos el SNR, *Packet Loss*, *Throughput* y otros datos relacionados a las condiciones de conexión de los usuarios. Además devuelve archivos donde también se registran datos de la conexión de los usuarios a lo largo de la simulación.

3.2. Integración de DeepMIMO a Py5cheSim

En esta sección se explica la integración del *framework* DeepMIMO al simulador Py5cheSim.

En primer lugar se selecciona uno de los escenarios otorgados por DeepMIMO [3] con el que se desea trabajar (ver tabla 2.1 de posibles escenarios) y los siguientes parámetros:

- **Ancho de banda del escenario**: El espectro disponible para utilizar por los usuarios y la radiobase.
- **Grupos de usuarios**: Los grupos de usuarios que estarán presentes en la simulación.
- **Características de los usuarios**: Incluye la velocidad y sentido de movimiento y su cantidad de antenas para recibir datos.
- **Potencia de transmisión**: Incluye la de los usuarios y las radiobases
- **subportadoras OFDM**: Todas las subportadoras disponibles para utilizar por los usuarios y la radiobase.
- **Cantidad de caminos**: Los usuarios de los escenarios reciben señal mediante varios caminos. Es posible acotar la cantidad de caminos que recibe un usuario. Para este proyecto se fijo la cantidad de caminos en 1, esto es

una aproximación que permite tomar decisiones de diseño que se explican más adelante.

- **Potencia de transmisión**: Es la potencia de transmisión asignada a las subportadoras OFDM.
- **Piso de ruido**: Es el valor que toma la densidad espectral de potencia del ruido en los usuarios. Teniendo en cuenta que se utiliza un modelo de ruido blanco, la densidad espectral de potencia es constante, esa constante es el piso de ruido.
- **Duración de la simulación**: La cantidad en milisegundos de la simulación a realizar.
- **Tasa de actualización**: El tiempo en que se considera que las condiciones del canal son estáticas. Los datos del escenario en este tiempo componen una escena.
- **Directorio de salida**: Donde se almacenan los archivos que posteriormente utilizará el simulador.

Estas entradas se ingresan a un script de nombre `Channel_Generator.py` que hace uso de ellas. Valiéndose de otros scripts con funciones auxiliares, este devuelve dos tipos de salidas. Una de ellas es un archivo `.json` con las características del escenario y la simulación, la otra salida es una carpeta que contiene archivos con las condiciones físicas de los usuarios para todos los grupos de usuarios ingresados.

El primer archivo, que nos indica las características del escenario y la simulación tiene la siguiente forma:

- Frecuencia central del escenario
- Ancho de banda del escenario
- Cantidad de PRBs del escenario
- Cantidad de portadoras del escenario
- Indicador de si el escenario es dinámico o no
- Tasa de actualización
- Duración de la simulación
- Grupos de usuarios

La segunda salida, consiste en una carpeta que se guarda en un directorio con el nombre del escenario. La carpeta contiene un directorio por cada grupo de usuarios presentes. Estas últimas contienen archivos con las características físicas de los usuarios para cada escena. Las características consisten en el SNR y rango

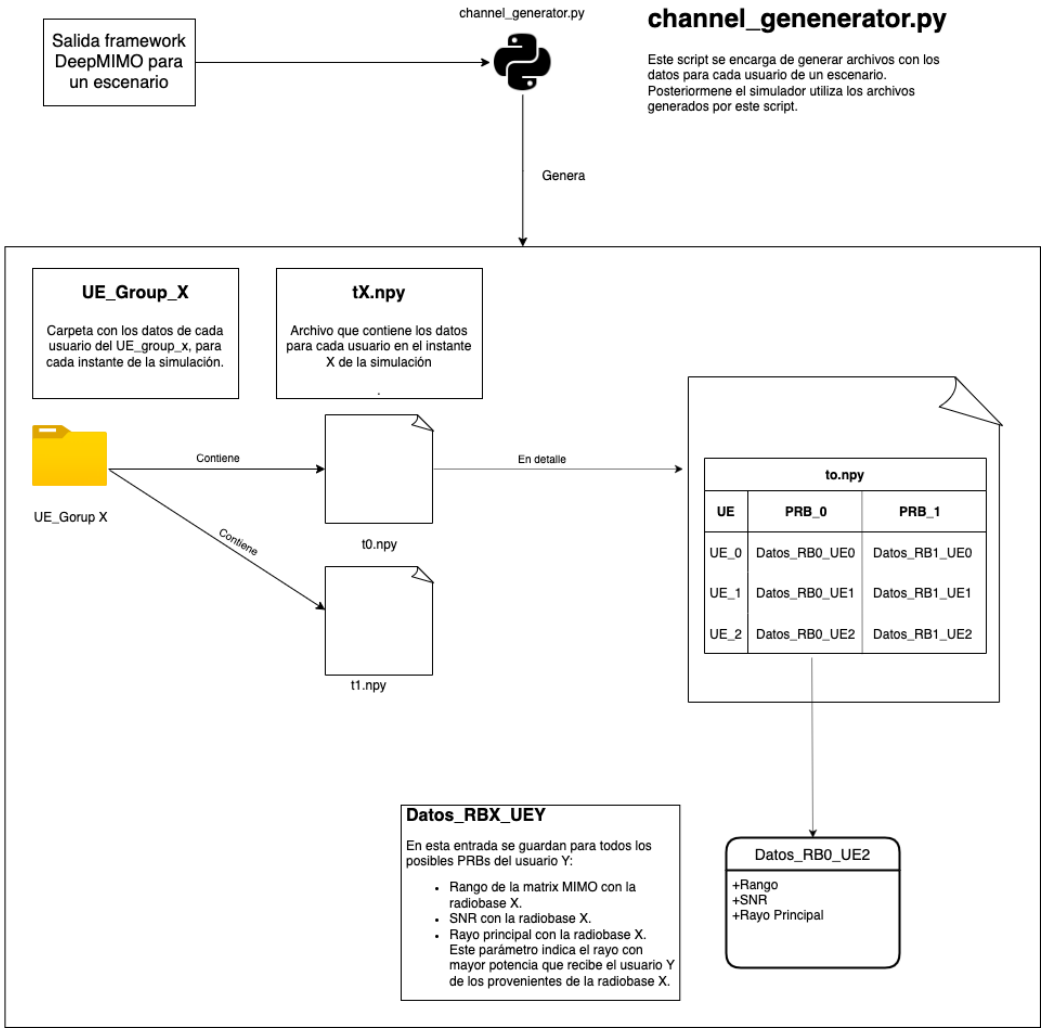

Figura 3.2: Esquema de integración del framework DeepMIMO al simulador.

en cada uno de los PRBs del escenario y el rayo principal de cada usuario. Más adelante se explican estos valores de salida y su cálculo.

La figura 3.2 muestra esquemáticamente cómo se extraen datos del escenario del *framework* y se los pasa al simulador como entrada.

Finalmente el simulador toma como entrada los archivos generados en la etapa anterior y un scheduler a elección del usuario y realiza la simulación.

3.2.1. Escenarios dinámicos

Originalmente el *framework* DeepMIMO no contiene escenarios dinámicos, por lo tanto fue necesario encontrar un mecanismo para dinamizarlos. Para realizar esta tarea se implementó una idea simple, tomar repetidas muestras de los datos del canal en el mismo escenario, los datos correspondientes a una misma repeti-

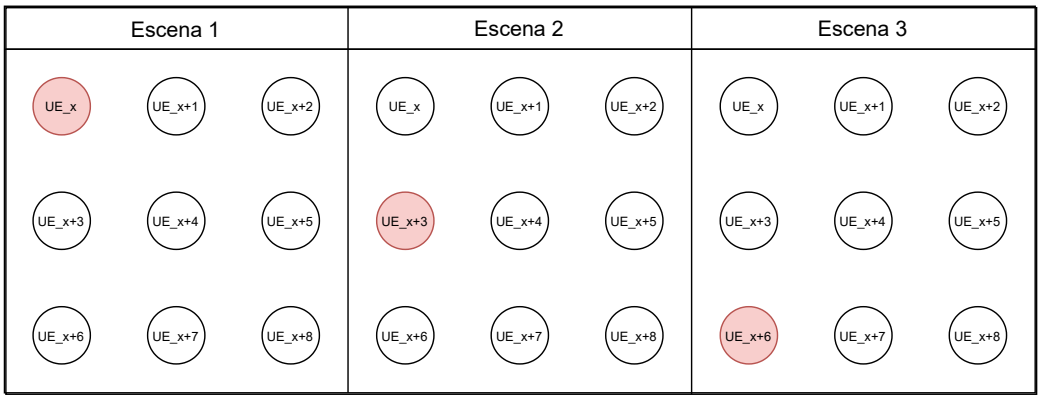

Figura 3.3: Esquema de toma de datos para un usuario dinámico que se mueve verticalmente, en una grilla de n usuarios.

ción componen una escena. Para simular el movimiento de un usuario, se toman muestras de los datos del canal de usuarios diferentes en las distintas escenas que componen la simulación. Particularmente, los movimientos disponibles para un usuario son verticales u horizontales. Por lo tanto para usuarios dinámicos los datos del canal que se toman son los datos de los usuarios que se encuentran en la vertical o la horizontal en que se encuentra el usuario dinámico.

La figura 3.3 muestra esquemáticamente cómo se toman los datos del canal para un usuario dinámico. En particular el esquema muestra la toma de datos para el usuario X, que se mueve verticalmente. En color rojo se indica el usuario del que se toman los datos que van a corresponder al usuario X.

Para poder realizar una simulación con escenarios dinámicos se deben ingresar los parámetros correspondientes al script `channel_generator.py`, que es el que extrae los datos del *framework* DeepMIMO. Es necesario ingresar a este *script* la siguiente lista de parámetros en caso que se quiera simular un escenario dinámico:

- `is_dynamic`: Este parámetro es un booleano que indica si el escenario es dinámico.
- `refresh_rate`: Indica la separación temporal de las escenas de la simulación
- Características de los usuarios dinámicos, a cada usuario dinámico de la simulación se le debe ingresar los siguientes campos:
  - `type_of_movement`: Indica la dirección en la que se mueve el usuario, puede ser vertical u horizontal.
  - `speed`: Indica la velocidad a la que se mueve el usuario en metros por segundo.
  - `position`: Indica la posición inicial del usuario.

En esta implementación se tomó la siguiente decisión de diseño para los casos en que el movimiento de un usuario hace que este escape a la grilla de usuarios.

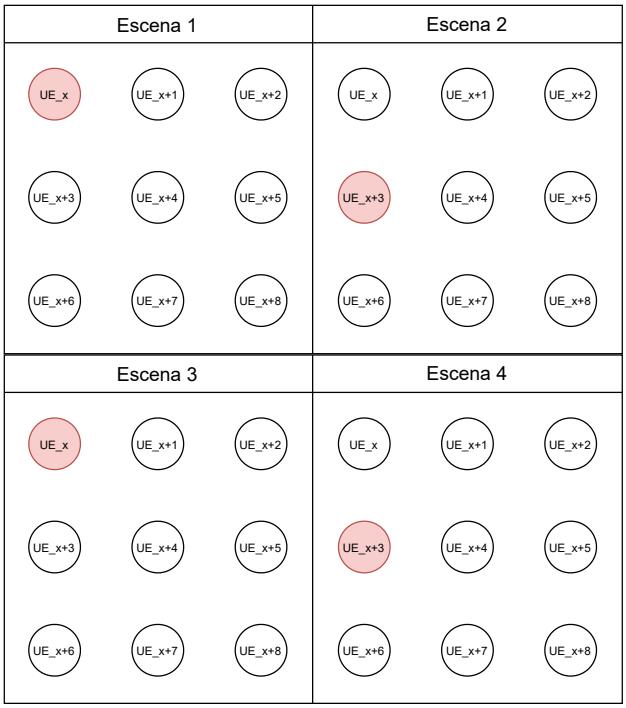

Figura 3.4: Movimiento de un usuario que escapa a su grilla de usuarios.

En ese caso se considera que el usuario rebota una vez que alcanza los límites de la grilla. La figura 3.4 muestra la decisión de diseño tomada para un usuario que se mueve verticalmente y escapa de su grilla de usuarios.

3.2.2. Escenarios con MIMO

Una de las principales ventajas con la que cuenta el *framework* DeepMIMO es que permite calcular los datos del canal de los usuarios cuando éstos emplean el método de comunicación MIMO. El *framework* permite elegir la cantidad y distribución de antenas que utiliza cada usuario y radiobase, a su vez permite visualizar el estado del canal para cada par de antenas entre un usuario y una radiobase. Este conjunto de datos permite calcular valores útiles en el caso que se desee simular que los usuarios utilizan MIMO.

En el caso que se desee realizar una simulación con MIMO se debe ingresar el número de antenas con el que cuenta cada usuario. Al ejecutar el *script* `channel_generator.py` se calculan los siguientes valores:

- `Rango`
- `SNR`
- `Rayo Principal`

Para calcular los primeros dos valores, que dependen de las condiciones del canal entre los usuarios y la radiobase, se tomaron una serie de simplificaciones.

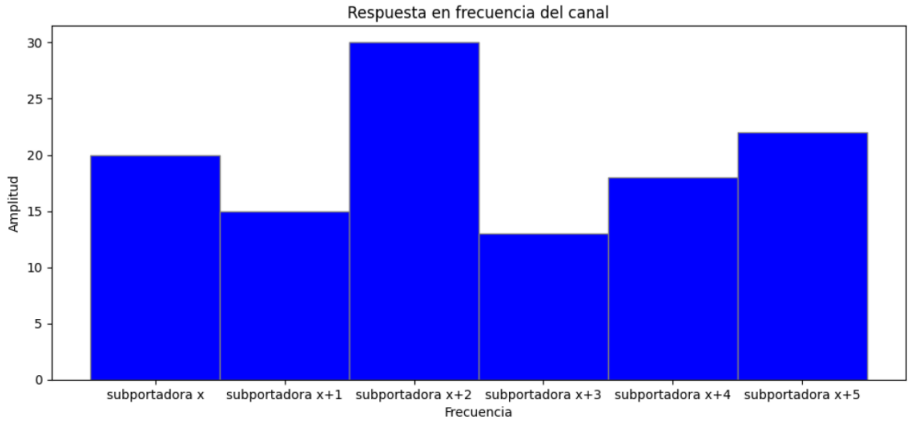

Figura 3.5: Respuesta en frecuencia del canal tomando la simplificación que el espectro contenido entre dos portadoras es constante

En primer lugar se asumió que el espectro del canal entre dos portadoras consecutivas es constante. Esta primera condición la impone el *framework* DeepMIMO, ya que no puede dar más granularidad a las condiciones del canal. En la figura 3.5 se muestra como se ve la respuesta en frecuencia del canal tomando en cuenta la primera simplificación tomada. En la figura 3.5 se tiene una porción del espectro conformado por seis subportadoras, el espectro contenido entre subportadoras consecutivas es constante.

La segunda simplificación que se tomó consistió en considerar que la transferencia del canal para un PRB está compuesta por un sólo valor. Este valor es el promedio de la transferencia de las portadoras que componen el PRB.

En las siguientes subsecciones se explica cómo se calcula y el significado de cada uno de estos valores.

Rango

Este valor indica cuantos flujos independientes (*layers*) se pueden establecer para cada par radiobase - usuario. La cantidad de antenas en recepción y transmisión y el canal entre la radiobase y el usuario determinará este valor. Teóricamente se calcula siguiendo la definicion matemática, pero para este proyecto se eligió utilizar una aproximación ya que esta última se adapta más a una situación realista.

Cálculo

La definicion matematica del rango es la siguiente:

Rango 1 *El rango de una matriz es el número máximo de columnas (filas respectivamente) que son linealmente independientes. [10]*

Dada una matriz M, que en nuestro caso representa el canal entre un usuario y la radiobase. Una manera de determinar el rango es realizar la descomposicion SVD y ver cuantos valores propios distintos de 0 posee la matriz V. Los valores

propios de la matriz V, representan la atenuación que experimentan los flujos de señal de las antenas.

La cantidad de valores propios distintos de cero en la matriz V representa la cantidad flujos independientes que un usuario puede recibir de una radiobase. En muchos casos se puede obtener una matriz V con todos sus valores propios distintos de cero, pero muy distintos entre ellos. Un ejemplo de esto es la matriz que se muestra en la ecuación 3.1, su valor propio central es del orden de 100 veces menor que los otros dos valores propios. Esta situación indicaría que el flujo central se atenúa mucho más que los otros dos flujos, lo que implicaría que fuese un error considerar que la matriz tiene un rango de 3. Para contemplar este tipo de situaciones se tomó la decisión de diseño siguiente:

Rango 1 *El rango de una matriz es la cantidad de valores propios mayores a un décimo del valor propio de mayor valor de la matriz V de la descomposición SVD de la matriz original.*

$$V = \begin{pmatrix} 0,6 & 0 & 0 \\ 0 & 0,002 & 0 \\ 0 & 0 & 0,4 \end{pmatrix} \tag{3.1}$$

Teniendo en cuenta la decisión de diseño anterior, para calcular el rango se toma para cada PRB la transferencia del canal entre un usuario y la radiobase. Al tener el usuario y la radiobase varias antenas existirán varias transferencias, estas transferencias se pueden representar como una matriz de transferencias. A esta matriz se le aplica la descomposición SVD y se le calcula el rango siguiendo la definición 1.

SNR

Este valor indica la relación entre la potencia de la señal y la potencia del ruido recibida por el usuario (ec. 2.1). A continuación se explica como se calcula la SNR cuando el usuario cuenta con una sola antena de recepción. Más adelante se extiende la explicación para la situación en que el usuario cuenta con dos o más antenas en recepción.

DeepMIMO devuelve el estado de canal para cada subportadora OFDM entre el usuario y la radiobase, esta situación permitiría utilizar la ecuación 2.2 para calcular el valor de SNR. [11] Pero en este caso, al calcularse la SNR por PRB la ecuación no se puede aplicar directamente. Lo que se hizo fue una extensión de lo que plantea la ecuación 2.2, calcular la potencia de la señal recibida en cada PRB y dividirla por la potencia del ruido en el PRB. La SNR se puede expresar mediante la siguiente ecuación:

$$SNR_{PRB} = \frac{P_{PRB}}{N_o \times B_{PRB}}$$

Donde P_{PRB} es la potencia de la señal recibida en el PRB, N_o es el nivel de ruido de los usuarios del escenario y B_{PRB} es el ancho de banda del PRB. Para

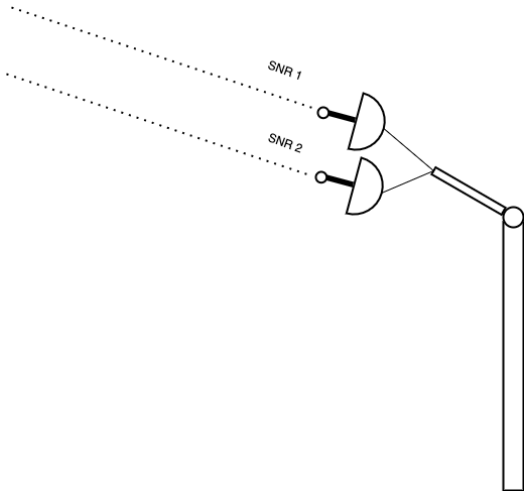

Figura 3.6: Recepción de datos de un usuario que cuenta con dos antenas, cada señal que se recibe contará con un SNR.

calcular la potencia de la señal recibida en el PRB, se sumaron las potencias de las portadoras recibidas en el PRB. Por lo tanto la potencia de la señal en el PRB vendrá dada por la siguiente fórmula:

$$P_{PRB} = \sum_{i=1}^{CS} P_i$$

Donde P_i es la potencia de cada subportadora y CS es la cantidad de suportadoras en un PRB, en 5G este valor es 12 [9]. Para calcular P_i nos basamos en [11]:

$$P_i = PT_i \times \alpha_i$$

Siendo PT_i la potencia de transmisión de la subportadora i y α_i la transferencia del canal en la subportadora i, que es otorgada por DeepMIMO en el parámetro `dataset[i]['user']['channel'][j]`. Se asume que la potencia transmitida se divide equitativamente entre las subportadoras, y el ruido gaussiano AWGN de valor arbitrario. Finalmente se promedia el SNR calculado en las portadoras que componen cada PRB.

Los valores de la potencia de la señal transmitida por la radiobase y el nivel de ruido quedan a elección del usuario. Pero es importante tener en cuenta que si se eligen valores que conlleven a un SNR muy pequeño la simulación mostrará que los usuarios tienen una mala comunicación con la radiobase.

En el caso que se este utilizando MIMO, el usuario contará con más de una antena, por lo tanto se tendrá una SNR por cada antena receptora, pues por cada antena se recibe una señal con su correspondiente SNR. En la figura 3.6 se muestra la situación anterior.

Es importante destacar que la señal que se recibe en las antenas se encuentra precodificada para poder maximizar la cantidad de *layers* entre el usuario y la

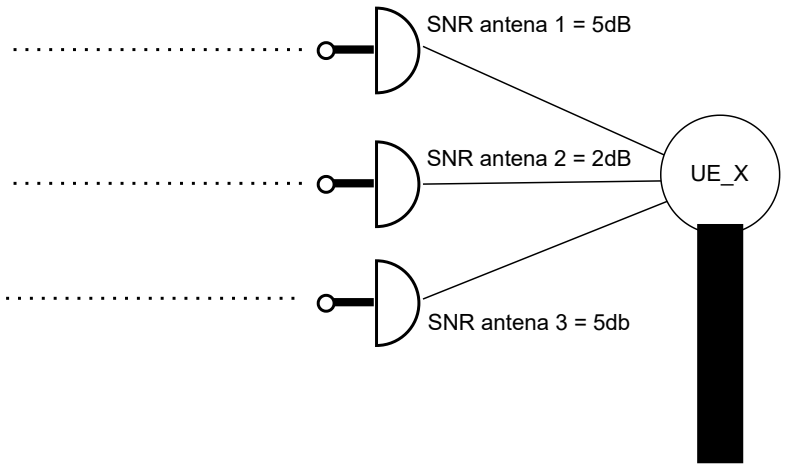

Figura 3.7: Ejemplo de recepción de señal para el usuario X, este cuenta con tres antenas, la SNR en la antena central es menor que en las otras dos antenas.

radiobase. Teniendo en cuenta que se utilizan matrices de precodificación y decodificación unitarias, la señal luego de ser decodificada por la antena receptora no sufre ninguna atenuación. Por lo tanto la potencia de la señal a la entrada de la antena es la misma que la señal luego de ser decodificada. Se puede calcular el SNR tomando la potencia de la señal recibida en las antenas y considerar un nivel de ruido en las mismas. Este cálculo es equivalente a calcular el SNR por *layer*.

La decisión de diseño tomada para asignar un SNR al usuario fue tomar el SNR de menor magnitud de los presentes en sus *layers*. Esta decisión es robusta en cuanto a que se toma la peor condición de ruido para el usuario. En la figura 3.7 se muestra un ejemplo para poder entender la situación, se tiene al usuario X que cuenta con tres antenas de recepción. La SNR en sus antenas 1 y 3 vale $5dB$, en la antena 2 se tiene la SNR más baja $2dB$, luego la SNR del usuario X será $2dB$.

En el caso que no se utilice MIMO, el usuario utilizará sólo una antena, por lo tanto el SNR del usuario será el SNR de la antena.

Rayo principal

Este valor indica los angulos acimutal y cenital con el que parte de la radiobase el rayo que llega con mayor potencia al usuario. En la figura 3.8 se observa una definición de estos ángulos en un sistema de coordenadas cartesianas de ejes x, y, z. Convenientemente este valor ya lo devuelve el *framework* en forma de vector cuyas componentes son el ángulo acimutal y cenital, por lo que no fue necesario calcularlo a partir de otros parámetros. El rayo principal es utilizado luego para poder asignar recursos eficientemente, en la sección 3.4 se explica en mayor detalle.

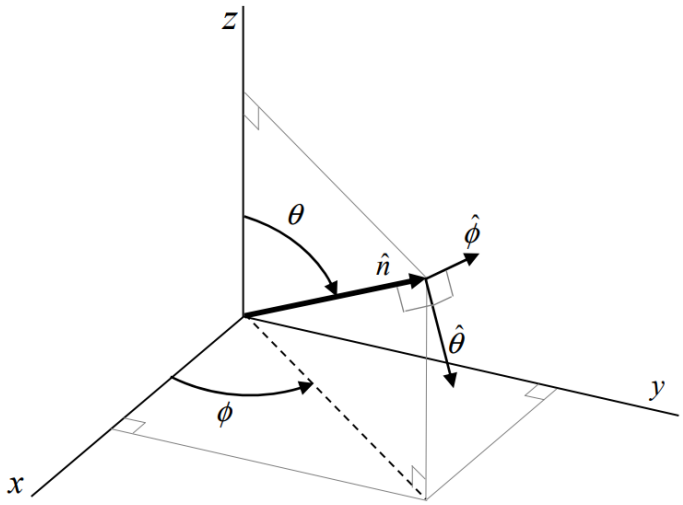

Figura 3.8: Definición del ángulo cenital θ y el angulo acimutal ϕ en un sistema de coordenadas cartesianas. Figura extraída de [12]

3.3. Mejoras en Py5cheSim

En esta sección se explican los cambios y los agregados que se realizaron sobre el código del simulador Py5cheSim. Detallando los objetivos específicos sobre este punto junto con las decisiones de diseño y la implementación realizada, la cual es explicada mediante su descripción funcional. Los detalles particulares de implementación a nivel de módulos y clases junto con la estructura de archivos se pueden encontrar en el anexo B.

3.3.1. Objetivos específicos

Con los nuevos datos del canal explicados en la sección 3.2 se desea adaptar el simulador para que integre los mismos, para que puedan ser utilizados por ejemplo por un nuevo *intra slice scheduler*. Por otro lado, se desea mantener la compatibilidad con simulaciones desarrolladas para la versión 1.

3.3.2. Compatibilidad hacia atrás

Para mantener la compatibilidad con la versión anterior y debido a que esta fue desarrollada bajo el paradigma de programación orientada a objetos, es que se optó por separar los métodos y atributos en clases padres e hijas según corresponda. En términos generales, para todas las clases modificadas en la nueva versión se hizo una distinción entre métodos y atributos comunes entre las funcionalidades ya soportadas y las nuevas. Dejando todo lo común a ambas en una clase base y sus particularidades en dos clases hijas distintas. Esto se puede ver esquemáticamente en la figura 3.9, en donde los métodos y atributos comunes a ambas se posicionan

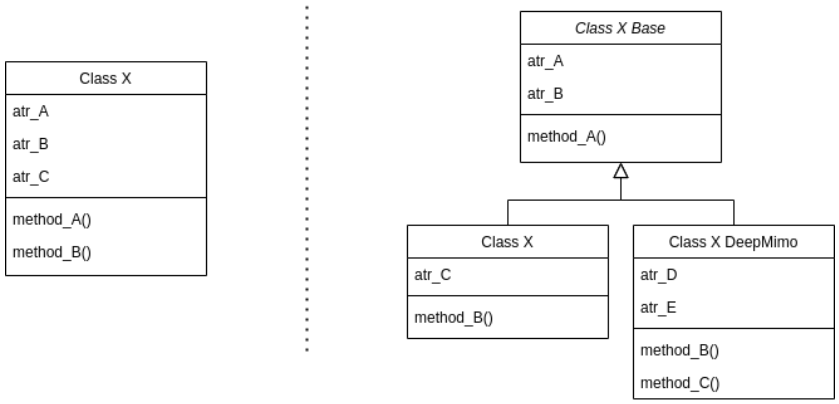

Figura 3.9: Esquema general de transición entre clases: a la izquierda se encuentran las clases en la versión 1 y a la derecha las clases en la versión 2.

en la clase base y los específicos a la versión 1 y 2 en las clases X y $XDeepMimo$ respectivamente.

En este punto es necesario aclarar que el nombre original de la clase en la versión 1 lo conserva la clase hija utilizada para implementar las funcionalidades de la primera versión. Esto es con el objetivo de mantener la compatibilidad con la versión anterior, de modo que los archivos de configuración generados para la versión 1 continúen funcionando en la nueva versión sin la necesidad de realizar ninguna modificación a los mismos.

3.3.3. Diagrama de clases

Para realizar una descripción funcional de las nuevas características es necesario presentar el diagrama de clases sobre el cual fueron implementadas. En la figura 3.10 se muestra dicho diagrama, en el cual se indican solamente los métodos y atributos principales, junto con un código de colores que indican si estos fueron agregados, modificados o removidos con respecto a la primera versión. En este punto es necesario aclarar que en virtud de simplificar el diagrama, para las clases modificadas solo se representa la nueva versión, las cuales se identifican con el posfijo *DeepMimo*. Cada una de estas clases debería ser remplazada con la estructura de clases presentada en la sección 3.3.2 la cual se puede ver en la figura 3.9.

Como se puede observar en la figura 3.10, las clases: *PacketFlow*, *PacketQueue*, *Packet*, *Bearer* y *TransportBlock*, encargadas de manejar los paquetes, no presentan modificaciones. Igualmente estas fueron incluídas para servir de apoyo en las descripciones funcionales que se presentan en las siguientes secciones.

3.3.4. Actualización del estado del canal

Como se explicó en la sección 3.2, la primera etapa de la simulación genera archivos que contienen el estados del canal entre los UEs del grupo y la radiobase. Como se representa en la figura 3.11, en esta nueva versión, la clase *UeGroupDeep-*

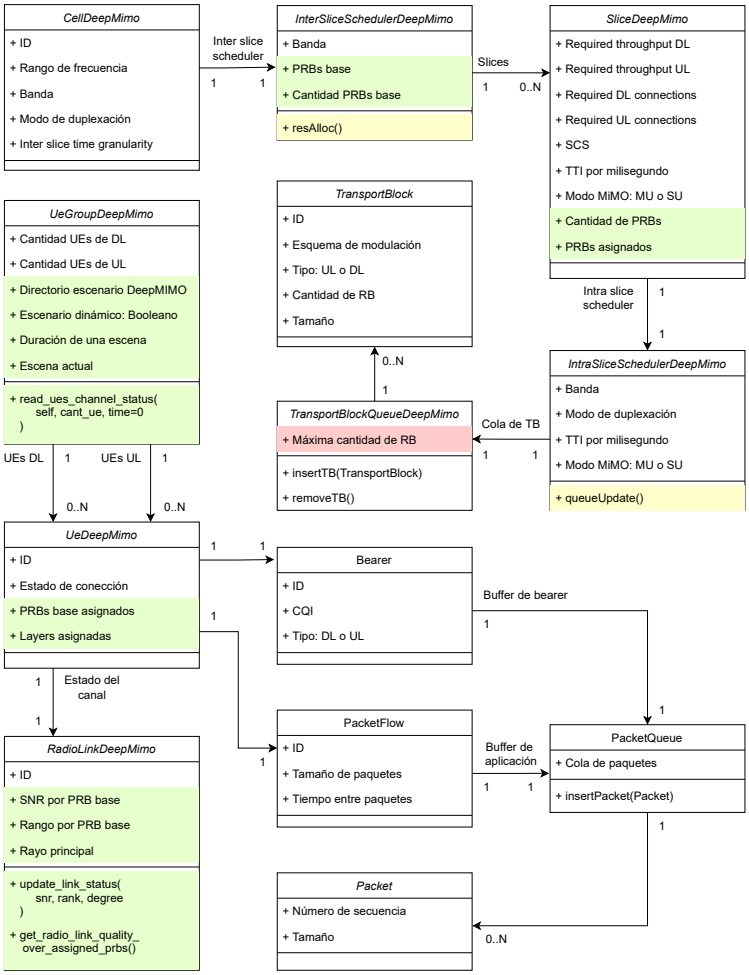

Figura 3.10: Diagrama de clases: En verde se muestran los métodos y atributos nuevos, en amarillo los modificados y en rojo los eliminados con respecto a la versión anterior.

Mimo almacena el directorio en donde se encuentran estos archivos y al momento de actualizar los datos, la misma carga la información del canal desde dichos archivos según corresponda en base al momento de la simulación. Luego, esta misma clase también se encarga de actualizar el estado del canal en la clase *RadioLinkDeepMimo* a través del usuario de la clase *UeDeepMimo*.

Este proceso se repite con un período igual a la duración de una escena, el cual es un parámetro que se genera en la primera etapa de la simulación. Por otro lado, los datos que se actualizan para especificar el estado del canal son: SNR y rango para cada PRB base del escenario, y el ángulo del rayo principal.

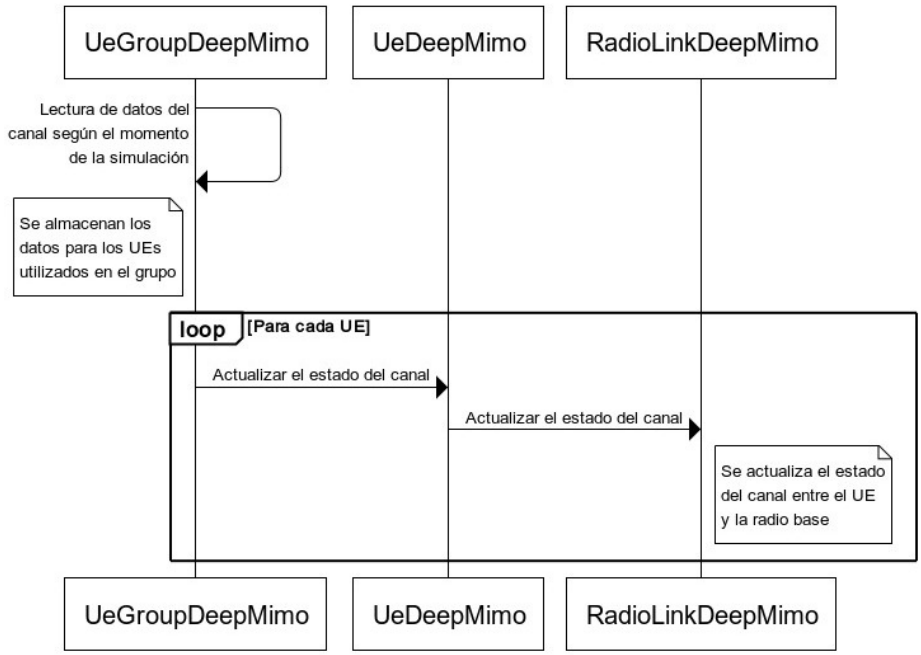

Figura 3.11: Diagrama de secuencia: proceso de actualización del estado del canal.

3.3.5. Network slicing

Si bien la primera versión del simulador ya soporta *network slicing*, debido a que este era un requirimiento fundamental para la misma, lo hace mediante la implementación de dos *schedulers* diferentes: un *inter slice scheduler* y varios *intra slice schedulers*, los cuales cambiaron de forma sustancial dado que en la segunda versión el recurso a asignar ya no es más la cantidad de PRBs, sino que se asignan PRBs particulares y esto influye en el estado del canal de los UEs con la radiobase debido a que dicho canal es selectivo en frecuencia.

Inter Slice Scheduler

En la figura 3.12 se muestra el proceso mediante el cual las diferentes *slices* obtienen recursos. En esta nueva versión, el *InterSliceSchedulerDeepMimo* divide los PRBs base definidos en la simulación de forma equitativa entre las diferentes *slices*, de forma tal que a lo largo de la simulación cada una de estas obtengan la misma cantidad de PRBs asignados. Un PRB base corresponde a un PRB cuyo espaciado entre subportadoras es de 15 kHz, con el que se pueden formar el resto de las configuraciones de PRBs con diferentes SCS. Por ejemplo, para formar un PRB con SCS de 30 kHz se necesitan dos PRBs base. Esta asignación básica fue realizada a modo de ejemplo, para que posteriores *inter slice scheduler* hereden de *InterSliceSchedulerDeepMimo* e implementen su propia asignación, en la cual pueden tener en cuenta el estado del canal de los UEs de las *slices* en los diferentes PRBs.

Este proceso de asignación de recursos entre *slices* se realiza de forma periódica,

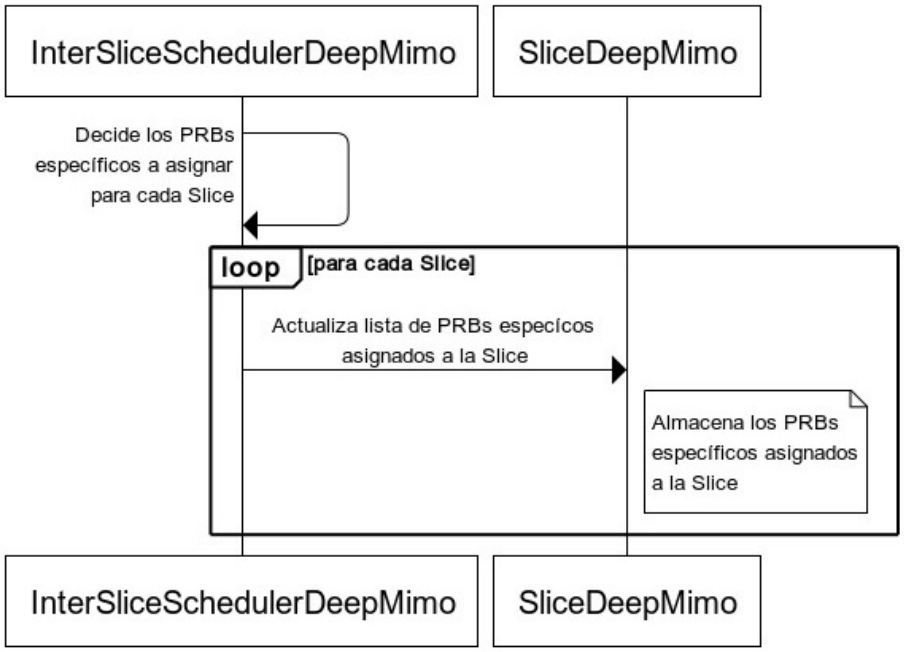

Figura 3.12: Diagrama de secuencia: proceso de asignación de recursos a las diferentes *slices*.

cada una cantidad de tiempo definido en el archivo de configuración. El mismo debe ser mayor que el tiempo máximo de TTI, el cual es 1 ms y corresponde al SCS de 15 kHz. En caso de ser menor, la asignación de recursos a nivel de *intra slice* pierde sentido.

Intra Slice Scheduler

Por otro lado, la siguiente etapa en la asignación de recursos es la realizada dentro de cada *slice*, donde cada uno de los *intra slice schedulers* se encargan de asignar recursos a los UEs que pertenecen a la misma. En la versión nueva, los *schedulers* que hereden de *IntraSliceSchedulerDeepMimo* tienen la posibilidad de explorar el estado del canal entre los UEs y la radiobase en los diferentes PRBs y decidir en base a eso cuales asignar junto con la cantidad de layers a utilizar.

A pesar de las particularidades que cada implementación de *intra slice scheduler* pueda tener, en términos generales debe apegarse al esquema presentado en la figura 3.13. En este, cada *scheduler* debe conocer los PRBs base específicos que están asignados a la *slice*, los cuales puede asignar a los diferentes UEs, para luego repartirlo entre los diferentes UEs según algún criterio.

Este proceso de asignación de recursos a nivel de UEs es periódico y se repite en cada TTI. El valor del TTI depende de la *slice*, y esta lo establece según los requerimientos de latencia tolerada que se indicó en el momento de especificar el grupo de usuarios el cual sirve esa *slice*.

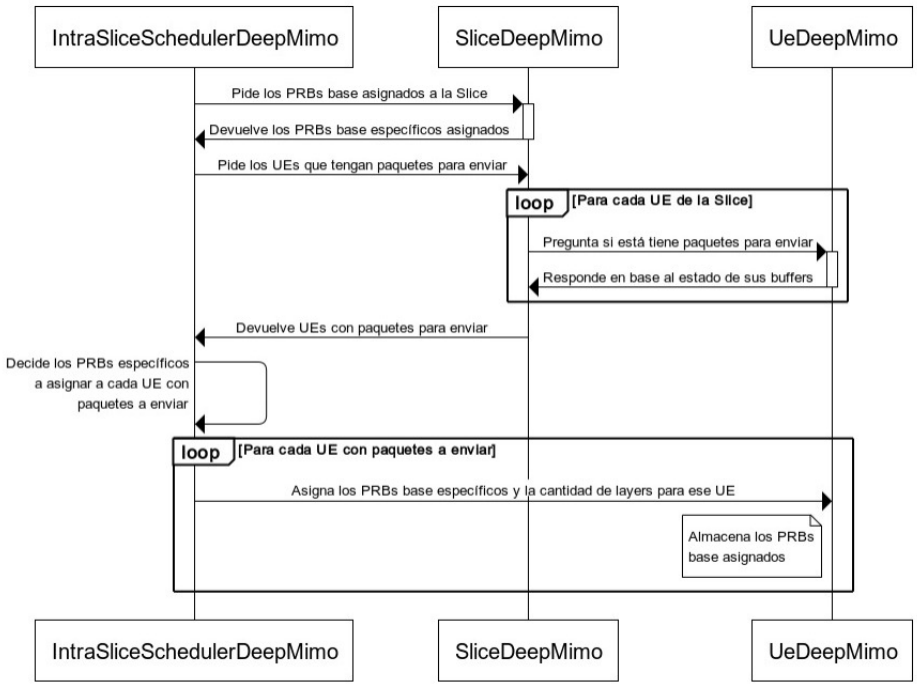

Figura 3.13: Diagrama de secuencia: proceso de asignación de recursos dentro de una *slice* a los diferente UEs.

Flujo de paquetes

El flujo de los paquetes en el simulador es el mismo que en la primera versión, el cual fue explicado en la sección 2.3.3, con la única diferencia que para obtener el MCS debe consultar el SNR del UE con la radiobase sobre los PRBs base asignados al mismo. Para esto el UE consulta el estado del SNR sobre los PRBs asignados mediante la clase *RadioLinkDeepMimo* y toma un promedio de los mismos.

Soporte para MIMO masivo

El soporte para MIMO está contemplado en el proceso de asignación de recursos entre UEs de una *slice*. Como se puede ver en el diagrama de la figura 3.13 el *IntraSliceSchedulerDeepMimo* asigna los PRBs base junto con la cantidad de layers que el mismo seleccionó para dicho UE. Notar que este cambio representa una mejora con respecto a la versión anterior, debido a que en ella se asignan la misma cantidad de layers para todos los UEs del grupo sin tener en consideración el estado del canal u otros aspectos, como la posición del UE relativa a la radiobase. En la nueva versión, la asignación de *layers* es específica a cada UE y queda a cargo del *IntraSliceSchedulerDeepMimo*, el cual puede tomar en consideración las características previamente ignoradas.

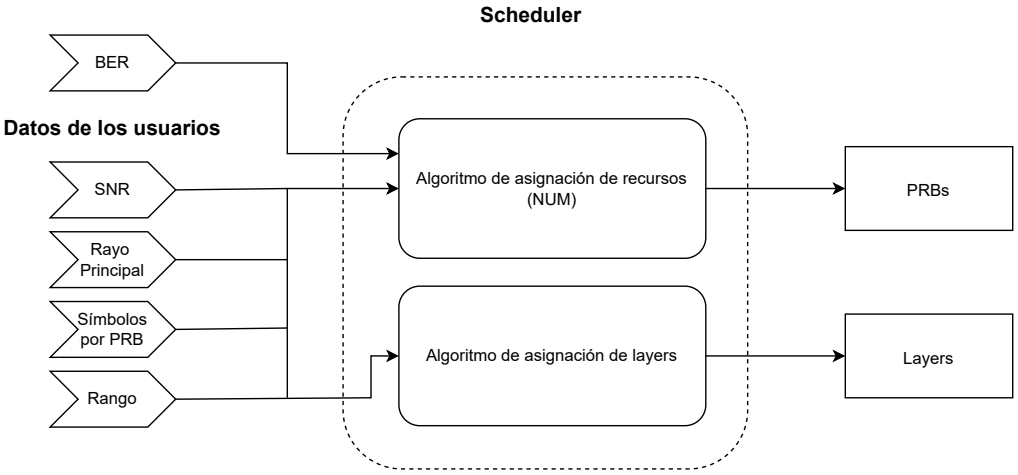

Figura 3.14: Diagrama del funcionamiento general del Scheduler.

3.4. Implementación de scheduler

Dadas las mejoras desarrolladas, un nuevo *intra slice scheduler* es soportado por Py5cheSim. El mismo se basa en un algoritmo de asignación de recursos óptima en sistemas inálambricos MU-MIMO OFDMA [7], que es justamente el motivo por el que se eligió este algoritmo y no otro. Este *scheduler* se encarga de dos tareas, asignar recursos a los usuarios de una *slice* y decidir cuantas *layers* utiliza cada usuario. Para la primer tarea, se siguió exclusivamente el algoritmo [7], para la segunda se tomó una decisión de diseño que más adelante se explicará.

En la figura 3.14 se ilustra el funcionamiento general del asignador de recursos construido. Este utiliza dos tipos de entradas, por un lado los datos de los usuarios que ya se encuentran definidos al momento de asignar recursos, estos son la SNR, el rayo principal definido en la sección 3.2.2, el rango y la cantidad de símbolos a enviar por PRB. Por otro lado se ingresan parámetros que se desea lograr en la simulación, estos únicamente consisten en la tasa de error de bits (BER).

Los datos de entrada ingresan a distintos bloques para que estos arrojen la salida correspondiente. La SNR, el rango, el rayo principal, la cantidad de símbolos por PRB y el BER entran al bloque asignador de recursos. Este implementa un algoritmo basado Network Utilization Maximization (NUM) para determinar que PRBs se le asignan a cada usuario, en la subsección 3.4.1 se explica este algoritmo. Por otra parte también el rango de los usuarios se ingresa al bloque asignador de *layers*, este asigna la cantidad de *layers* que utilizará cada usuario. En la subsección 3.4.3 se explica el funcionamiento de esta asiganación.

3.4.1. Algoritmo NUM

Network Utilization Maximization (NUM) es un acercamiento para implementar algoritmos de asignación de recursos, el algoritmo utilizado para asignar PRBs se basa en este acercamiento [7]. El mismo está diseñado para asignar recursos

cuando los usuarios utilizan MU-MIMO, su fin es intentar asignar uno o más PRBs a un grupo de usuarios.

En modo resumido el algoritmo NUM funciona como se explica a continuación. Se forman todos los grupos de usuarios posibles a los que se le pueden asignar PRBs, luego para cada PRB de los disponibles se calcula una métrica. Finalmente, se asigna el PRB que se quiere asignar a los usuarios que componen el grupo que obtuvo la mejor métrica [7]. A continuación se detallan los pasos de este funcionamiento.

En primer lugar se forman los grupos de usuarios, la cantidad de grupos de usuarios posibles es 2^{M-1} siendo M la cantidad de usuarios. A medida que M crece la cantidad de grupos se vuelve muy grande, esto requeriría grandes capacidades de cómputo. Por esta razón se optó por crear grupos cuyos usuarios puedan en principio compartir recursos, estos son los usuarios que no comparten rayo principal. Si los ángulos que componen el rayo principal distan lo suficiente, parece razonable considerar que los usuarios se encuentran alejados. La diferencia entre los ángulos que componen el rayo principal se fijó en 5 grados, pero es un parámetro fácilmente cambiable. También es razonable asumir que si los usuarios se encuentran alejados pueden entonces compartir PRBs. En el anexo se detalla una prueba empírica de esta situación A.1.2.

Una vez formados los grupos de usuarios se calcula para cada PRB la métrica asociada al grupo. Para poder entender la métrica que el algoritmo utiliza para asignar PRBs, es necesario primero definir algunas métricas intermedias:

- $\mathcal{M}_c$: Indica la cantidad de grupos posibles, considerando que se está agrupando utilizando el criterio del rayo principal.

- $\mathcal{M}_j$: Los conjuntos de usuarios no vacíos y que cumplen con la restricción de no compartir el rayo principal.

- c_i: A este valor se le llama *throughput* e indica el *throughput* del usuario i en el PRB k, viene dado por la siguiente ecuación:

$$c_i(k,l) = \sum_{s=1}^{\eta_i} N_{PRB}^{RE} \log_2(1 + \beta \rho_{i,s}^{\mathcal{M}_j}(k,l))$$

  Siendo l el instante donde se calcula c_i, el parámetro β es definido en base al BER que se desea lograr y vale $\beta = \frac{1{,}5}{-\log(5BER)}$. N_{PRB}^{RE} es la cantidad de símbolos OFDM por portadora. $\rho_{i,s}^{Mj}(k,l)$ es la SINR, en nuestro caso sustituímos este valor por la SNR ya que del *framework* DeepMIMO no podemos obetener el primer valor. η_i es la cantidad de layers del usuario i, este valor lo sustituimos por el rango del usuario. Es importante aclarar que esto último es una simplificación, pues el rango de un usuario no tiene porque coincidir con la cantidad de *layers* que se le asignan.

- r_i: A este parámetro se le llama *throughput* total, e indica la suma de los *throughputs* en los grupos que el usuario i pertenece. Este valor se puede

expresar mediante la siguiente expresión:

$$r_i = \sum_{j=1}^{\mathcal{M}_c} \sum_{k \in D_j} c_i(j)$$

Donde D_j es el conjunto de PRBs asignado al conjunto de usuarios j.

- $U(x)$: Es una función de utilidad a elección del usuario del algoritmo. Una comúnmente utilizada es $\log(x)$ [7].
- $\overline{r_i}$: Es el promedio del *throughput* total. Para calcular este valor se puede no seguir la definición estricta de promedio, una manera es utilizar el método *moving exponential average filter* [7]:

  $$\overline{r_i}(l) = \delta \overline{r_i}(l-1) + (1-\delta) r_i(l-1)$$

  Donde $l-1$ es el instante anterior y $\delta \in (0,1)$ es un valor a elegir por el usuario. A mayor δ, menos atenuado queda el término del promedio de la escena anterior, esta situación prioriza los valores de *throughput* pasados. Si se disminuye el valor de δ, se le da más peso al *throughput* de la escena pasada. Lo que significa un *scheduler* que tiene más en cuenta la situación actual para asignar un PRB.

La métrica asociada a los grupos de usuarios viene dada por la siguiente ecuación:

$$r_i(l) = \sum_{i \in M_j} c_i(k,l) U'(\overline{r_i(l)})$$

El grupo de usuarios que maximiza esta métrica será el óptimo para asignarle el PRB k en el instante l [7]. En la figura 3.15 se observa un diagrama de flujo del algoritmo.

3.4.2. Algoritmo de asignación de layers

Para asignar la cantidad de *layers* a los usuarios se tomó un criterio. Si a un conjunto de usuarios se le asigna un grupo de PRBs, la cantidad de *layers* de estos usuarios será la misma y será igual al menor de los rangos de estos. De esta manera se asegura que un usuario no reciba más *layers* de las que puede recibir. En la figura 3.16 se muestra el funcionamiento del algoritmo asignador de *layers*. Asumiendo que los usuarios 3, 7 y 10 fueron asignados a un PRB, la cantidad de *layers* a asignar a estos usuarios será 1, pues el menor de los rangos de estos es 1.

Como contrapartida esta decisión de diseño podría permitir que se utilicen más *layers* que las que permiten las recomendaciones de 5G que es de 4 *layers* por usuario [9]. Por ejemplo, si el grupo de usuarios al que se le asigna un PRB se encuentra conformado por usuarios que soportan 4 o más *layers*, el algoritmo de asignación no respetaría las recomendaciones de 5G. Por más que se pueda dar esta situación que no respeta las recomendaciones, el algoritmo permite extender las funcionalidades del simulador.

3.4.3. NUM_Scheduler

Para la implementación del scheduler se creó una nueva clase de tipo `IntraSlice`. Es en esta clase donde se ingresan parámetros necesarios para algoritmo de asignación de recursos. Se ingresa el umbral en el que se considera que usuarios comparten rayo principal, el BER deseado, la función de utilidad y el valor de δ.

A continuación se detallan algunas funciones importantes de esta clase.

- **resAlloc**: Método que dado el UE con mejor métrica, asigna todos los PRBs de la slice al mismo en cada *time slot*. Este método sobreescribe resAlloc en la clase *IntraSliceScheduler*.
- **convert_PRBs_base_to_PRBs**: Dado que la cantidad de PRBs disponibles cambia dependiendo del espacio entre subportadoras, este método devuelve una lista de los PRBs de la *slice*.
- **set_sched_groups**: Teniendo en cuenta el ángulo del rayo de partida de cada UE, este método divide a todos los UEs en grupos que pueden compartir recursos.
- **get_sched_groups_num_factors**: En este método se guarda el valor de la métrica para cada grupo de UEs dado un PRB.
- **compute_NUM_factor**: Se computa la métrica para un grupo de UE dado.
- **compute_UE_sched_groups_throughput**: Este método devuelve para un PRB dado, la suma del *throughput* en los UEs asignados al mismo.
- **compute_UE_throughput**: El método que devuelve el *throughput* de un UE en un PRB específico.

3.5. Resumen del capítulo

En el presente capítulo se detalló el diseño y el desarrollo de la solución del proyecto. Se modificó el modelo de canal del simulador Py5cheSim para obtener una medida más realista que la que otorgaba en su primera versión, integrando el *framework* DeepMIMO y sus salidas. Además, se implementó un nuevo *scheduler* utilizando un algoritmo de asignación que tuviera en cuenta el soporte para MIMO masivo desarrollado.

A continuación se desarrolla la validación de las modificaciones e implementaciones desarrolladas.

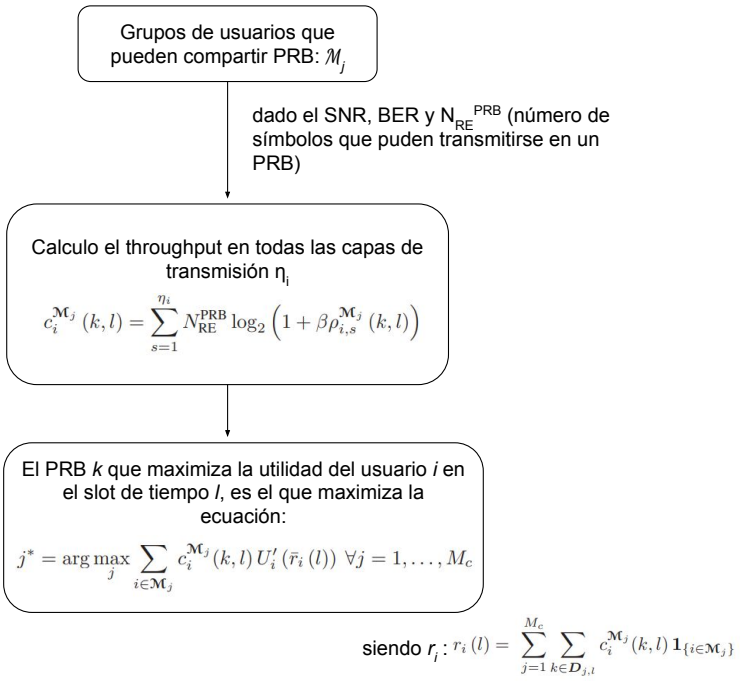

Figura 3.15: Esquema del algoritmo de asignación de recursos.

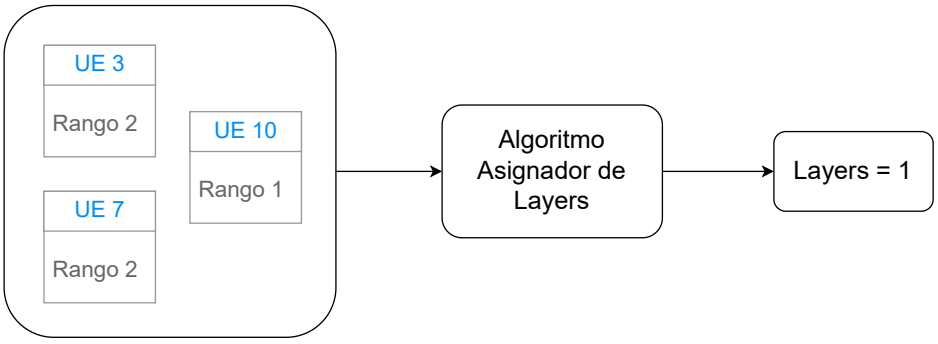

Figura 3.16: Ejemplo de asignación de *layers* cuando se cuenta con dos usuarios con rango 2 y uno con rango 1.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Validación

En este capítulo se presentan las diferentes validaciones realizadas a las funcionalidades descritas en el capítulo 3. Las mismas fueron divididas en tres grupos distintos debido a que sus implementaciones se llevaron a cabo de manera independiente de las demás. Estos grupos son los siguientes: validación del modelo de canal inalámbrico, validación de las modificaciones realizadas sobre el simulador Py5cheSim y validación del *scheduler*. Estas validaciones serán desarrolladas en detalle en las siguientes secciones.

4.1. Canal inalámbrico

Como se explicó en la sección 3.1, la simulación consta de dos etapas: la primera genera los datos del canal inalámbrico, y la segunda se encarga de ejecutar la simulación, utilizando o no los datos generados previamente. En este punto nos interesa validar la primera parte de la simulación, la cual es explicada en detalle en la sección 3.2.

La validación de esta etapa, la realizamos de dos maneras distintas: primero de forma teórica, explorando directamente el estado del canal inalámbrico desde el *framework* DeepMIMO y comparando dichos valores contra el resultado obtenido de aplicar el modelo teórico de canal mediante la ecuación de Friis, ecuación 4.1. Por otra parte, la segunda validación la realizamos sobre el resultado de una ejecución del simulador, preparando un escenario en donde el estado del canal tenga una evolución predecible a lo largo del tiempo, y verificando con el gráfico de SNR producido por la misma. En el resto de esta sección desarrollamos las simulaciones descritas.

4.1.1. Modelo de canal teórico

Se validó una de las salidas que devuelve el *framework* utilizado, la salida *path loss*. Ésta se define como el cociente entre la potencia de la señal transmitida y la recibida. DeepMIMO obtiene este valor para cada par radiobase/usuario en cada escenario, originalmente el encargado de calcular este parámetro es el *software*

Wireless InSite. Posteriormente el *framework* levanta este valor y otros, construidos por el programa antes mencionado.

Se utilizaron dos métodos para validar esta salida, el primero fue mediante el modelo de pérdidas de espacio libre de Friis, el segundo método fue mediante ecuaciones otorgadas por el estándar de 5G [12]. Por un lado, según el modelo de Friis el *path loss* en condiciones de espacio libre se puede calcular mediante la expresión 4.1.

$$PathLoss = \frac{P_{Tx}}{P_{Rx}} = \left(\frac{4 \times \pi \times f \times D}{c}\right)^2 \times \frac{1}{G_{Tx} \times G_{Rx}} \tag{4.1}$$

Siendo:

- f la frecuencia central de la transmisión
- c la velocidad de la luz
- D la distancia euclídea entre transmisor y receptor
- G_{Tx} la ganancia de la antena transmisora
- G_{Rx} la ganancia de la antena receptora

Asumiendo unitarias las ganacias de la antena transmisora y receptora y pasando a decibeles la ecuación 4.1, se obtiene el resultado dado por la expresión 4.2.

$$PathLoss = 20 \times \log_{10}\left(\frac{4 \times \pi \times f \times D}{c}\right) \tag{4.2}$$

Por otro lado la recomendación del estándar de 5G [12] propone distintos modelos para calcular el *path loss* dependiendo del entorno y la distancia entre la radiobase y el usuario. Considerando que el escenario que se tiene es urbano, la distancia entre la radiobase y el usuario es menor a 2 kilómetros siempre y hay línea de vista entre ellos, se puede utilizar la ecuación [12]:

$$PathLoss = 28{,}0 + 22 \times \log_{10}(D) + 20 \times \log_{10}(f) \pm n \times \delta_{SF} \tag{4.3}$$

En esta ecuación D es la distancia entre la radiobase y el usuario, f la frecuencia central de transmisión y δ_{SF} la desviación estándar del *shadow fading* en escala logarítmica. Este último término corresponde a una atenuación que sucede en los canales inalámbricos debido a obstáculos entre el usuario y la radiobase, en este caso usuarios y otras radio bases [17]. Esta atenuación se modela estocásticamente como una variable aleatoria de distribución *log-normal* [17], esto implica que el logaritmo de esta variable tiene una distribución normal. Considerando que estamos tomando el *path loss* en dB, el término correspondiente al *shadow fading* también debe estar en dB. Al pasar esta atenuación a dB, el *shadow fading* pasa a tener una distribución normal. Esta situación permite acotar el *path loss*, en la ecuación 4.3 el término correspondiente al *shadow fading* ya se encuentra en escala logarítmica.

Por lo tanto, la ecuación 4.3 brinda dos cotas para el *path loss* dependiendo de cuantas desviaciones estándar del *shadow fading* en escala logarítmica se tomen. Si se toma una desviación estándar podemos acotar el *path loss* con una probabilidad de $68,2\,\%$, mientras que si se toman dos desviaciones estándar se puede acotar con una probabilidad del $95,4\,\%$. Con el parámetro n de la ecuación 4.3 ajustamos la cantidad de desviaciones estándar que se consideran.

La prueba que se eligió, consistió en un usuario acercándose a la radiobase verticalmente y comparar el *path loss* arrojado por DeepMIMO con el calculado mediante la ecuación de Friis 4.2 y la ecuación del estándar 4.3. El escenario elegido fue *Outdoor 1* del *framework* DeepMIMO, el cual se explica en detalle en el anexo C.2 y se puede ver en las figuras 4.3 y 4.4, la frecuencia central de este escenario es 28 GHz. En cuanto al usuario, se eligó una posición inicial de 12 metros a la radiobase y una velocidad de $5,8$ m/s. La radiobase utilizada fue la número 12, en la figura 4.4 se la puede identificar. En la figura 4.1 se muestran las posiciones que toma el usuario a medida que se toman los datos. Teniendo en cuenta que el escenario es urbano y hay línea de vista entre la radio base y el usuario, y este último se aleja distancias menores a $5km$ de la radiobase, el valor δ_{SF} vale 4 dB [12].

En la figura 4.2 se muestra una gráfica que compara el *path loss* arrojado por DeepMIMO con el obtenido mediante la ecuación de Friis y el obtenido mediante la ecuación 4.3 utilizando $n = 1$ y $n = 2$. Se puede apreciar que cuando el usuario se encuentra a menos de 100 metros de la radiobase el modelo de Friis se asemeja en gran medida a la salida otorgada por DeepMIMO, las diferencias son menores a 4 dB. A medida que el usuario se aleja de la radiobase, los modelos se alejan también, existen puntos donde las diferencias son significativas, alcanzando los 9 dB. Comparando los resultados arrojado por DeepMIMO y las cotas dadas por la ecuación 4.3, salvo por dos puntos, el *path loss* se encuentra contenido entre las cotas obtenidas cuando se consideran dos desviaciones estándar. Si se consideran las cotas obtenidas al utilizar una desviación estándar, el $80\,\%$ de los puntos del *path loss* se encuentra contenido entre estas.

4.1.2. Gráfico de SNR con Py5cheSim

Otra forma de validar el estado del canal es utilizando el gráfico de SNR por UE que genera una simulación de Py5cheSim, el cual en la nueva versión es actualizado periódicamente a partir de los archivos del estado del canal generados en base al *framework* DeepMIMO. Por otro lado, necesitamos conocer de antemano la evolución del canal para los UE utilizados, para compararlos luego con los gráficos de SNR generados.

En base a esto, utilizamos para esta etapa el escenario *Outdoor 1* del *framework* DeepMIMO, el cual se explica en detalle en el anexo C.2 y se puede ver en las figuras 4.3 y 4.4. En este escenario, generamos los datos del estado del canal para tres UEs, los cuales comienzan su recorrido a 350 metros de la radiobase 12 y recorren aproximadamente 336 metros en dirección a la misma, como se puede ver en la figura 4.5. Para dichos UEs, según la ecuación de Friis presentada en la

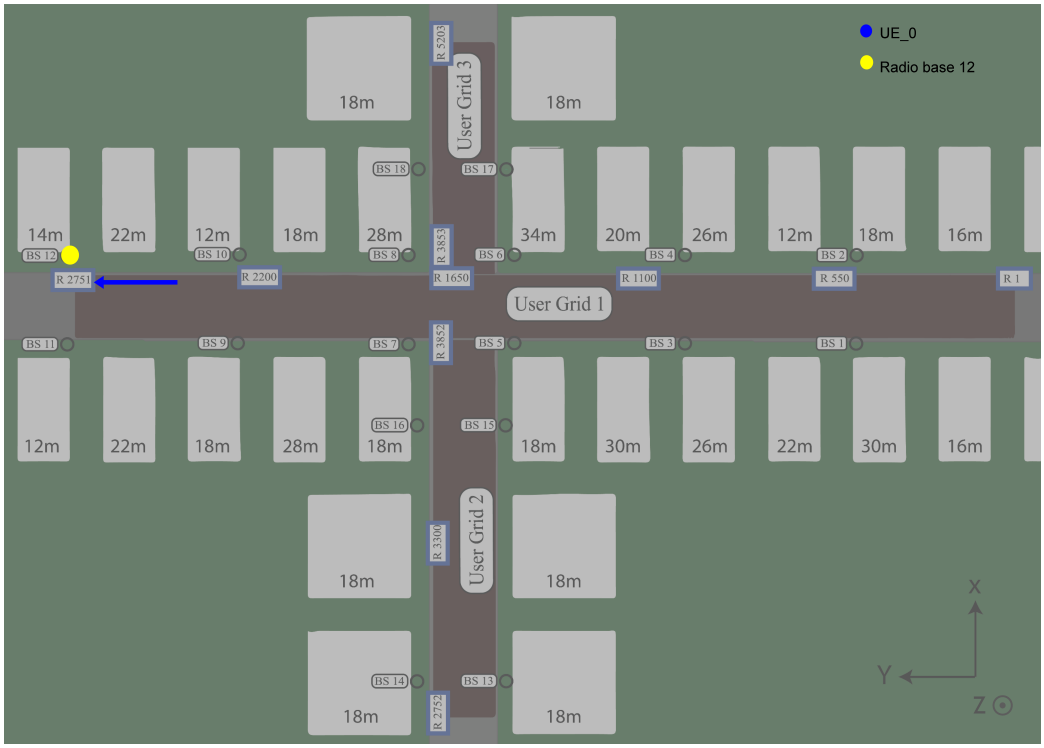

Figura 4.1: Movimiento del usuario en el escenario *Outdoor 1* para la validación del *scheduler*, el usuario se mueve en línea recta acercándose a la radiobase.

sección 4.1.1, la cual aplica en esta situación debido a que existe línea de vista entre los UEs y la radiobase, es lógico esperar que la calidad del canal aumente en el transcurso de la simulación, debido a que la distancia entre la radiobase y los UEs es cada vez menor.

En la gráfica de la figura 4.6 se puede ver la evolución del SNR de los tres UEs en el tiempo para el escenario de 28 GHz, las cuales coinciden con el comportamiento esperado. Además, se puede observar una leve diferencia entre ellos, la cual se debe a que el *UE_1* siempre está más lejos de la radiobase que el *UE_2*, por lo cual, aplicando otra vez la ecuación de Friis, explica dicha diferencia. Lo mismo sucede entre *UE_2* y *UE_3*. De esta manera validamos de forma primaria que la etapa de generación de datos del canal y la compatibilidad de los mismos con el simulador funcionan como esperábamos.

4.1.3. Validación de movimiento

Una validación que se realizó, fue la validación del movimiento de los usuarios. Particularmente la situación en que un usuario rebota, ya que el movimiento sin rebotes se valida también en otras pruebas, como la detallada en la subsección 4.1.1.

Para poder realizar esta tarea se eligió el escenario *I2 (Indoor 2) Blockage Scenario*, explicado en la subsección 2.4, en la figura 2.7 se muestra una visión

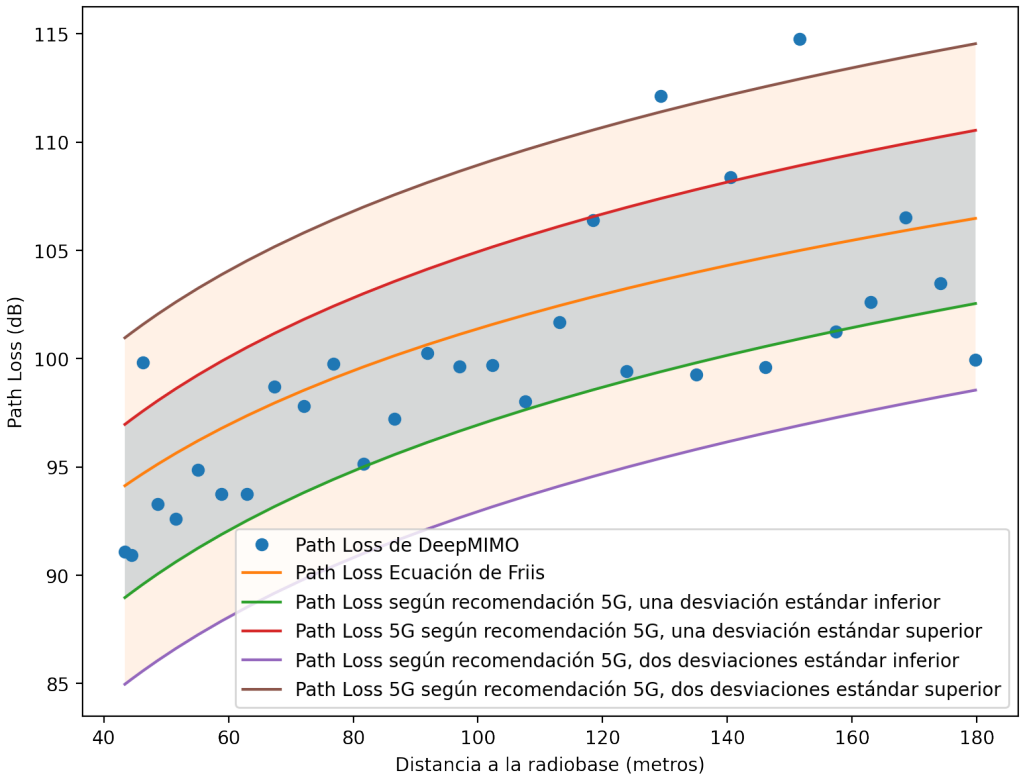

Figura 4.2: Comparación entre *path loss* arrojado por DeepMIMO, el obtenido mediante la ecuación de Friis y las cotas obtenidas mediante la ecuación 4.3 considerando $n = 1$ y $n = 2$.

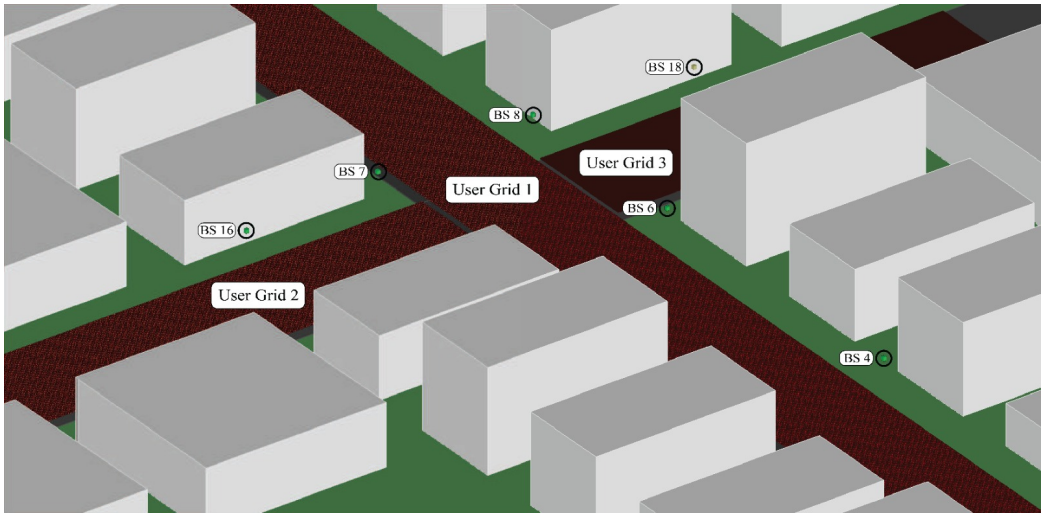

Figura 4.3: Vista tridimensional del escenario *Outdoor 1* del *framework DeepMMIMO*. Figura extraída de [3].

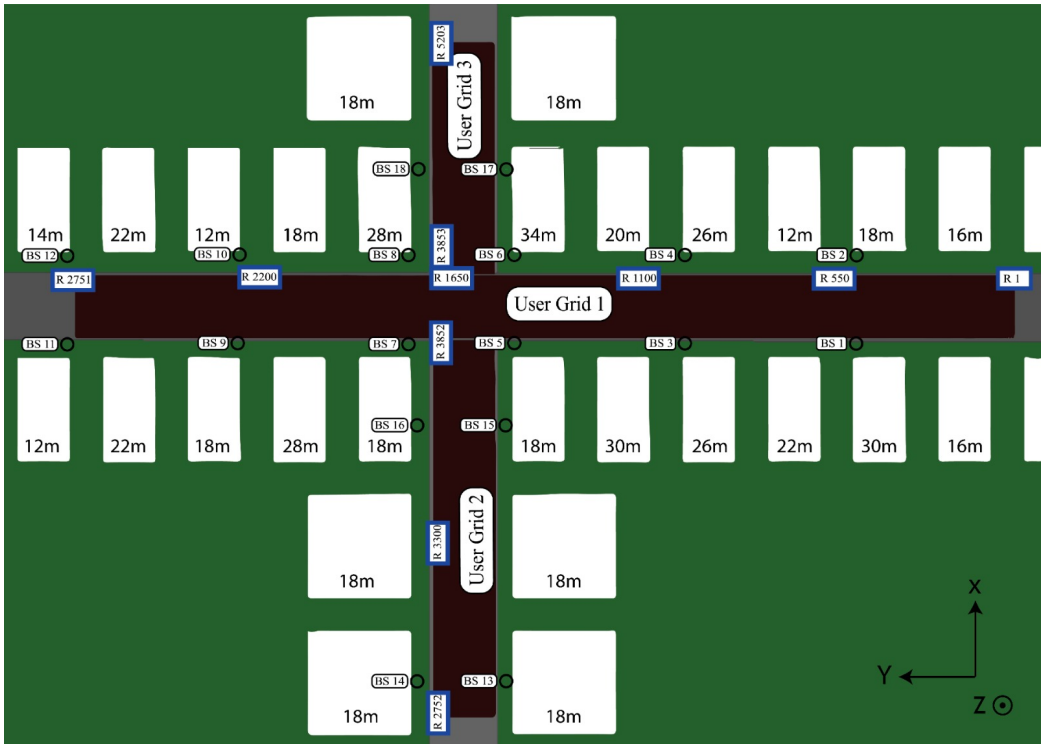

Figura 4.4: Vista de planta del escenario *Outdoor 1* del *framework DeepMMIMO*. Figura extraída de [3].

tridimensional del mismo. Se eligió un usuario que se mueve horizontalmente, la posición inicial del usuario es justo detrás del muro en el límite izquierdo, la velocidad es de $0{,}22m/s$ a la derecha. La ventana de tiempo en la que se toman los datos del escenario es de 25 segundos y la cantidad de escenas 250. Considerando la velocidad del usuario y la ventana de tiempo mencionada anteriormente, el usuario recorrerá una distancia de $5,5$ metros. Teniendo en cuenta que la posición inicial del usuario es el límite izquierdo y que la grilla de usuarios del escenario I2 tiene un longitud horizontal de 2 metros, el usuario rebotará 3 veces. Particularmente rebotará cada $9,09$ segundos. En la figura 4.8 se muestran las posiciones que toma el usuario.

En la figura 4.9 se puede observar el SNR del usuario obtenido al realizar la simulación. Como era de esperarse este SNR es periódico, siendo el período $18,18$ segundos. También se puede apreciar en la figura 4.9 como la SNR decae al acercarse al centro de la fila. Esto último se explica por el muro que bloquea a los usuarios, en el centro de las filas la cantidad de rayos que puede llegar a los usuarios es mucho menor que en los laterales, por lo tanto el SNR decae.

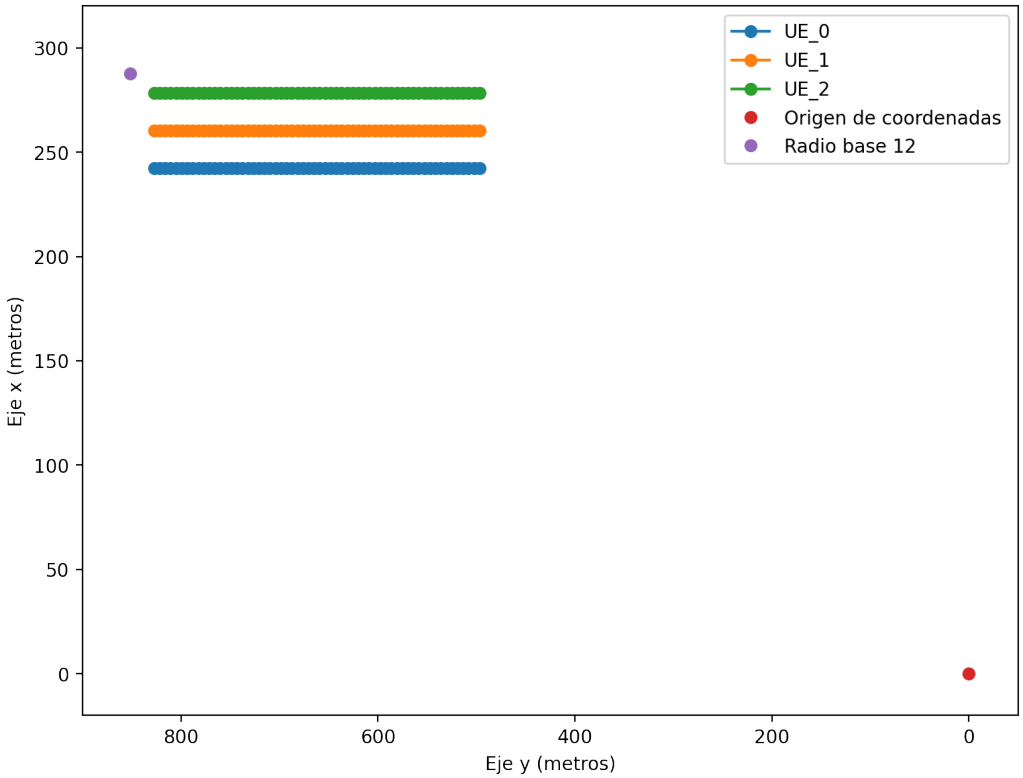

Figura 4.5: Gráfica de la posición de los tres UEs generados a partir del escenario *Outdoor 1* del *framework* DeepMIMO. Los ejes fueron manipulados para que coincidan con los ejes de la figura 4.4.

4.2. Py5cheSim

En esta sección nos enfocamos en validar el núcleo del simulador, es decir, aquellas funcionalidades que no están relacionadas con la generación de datos del canal, ni con los *schedulers* desarrollados, sino que son las que se encargan de la generación y transporte de paquetes, y en el envío efectivo de estos por la interfaz de radio. Si bien esta etapa no fue modificada considerablemente, es necesario verificar su funcionamiento con escenarios en la nueva versión para conseguir una validación más confiable.

Para esto, tomamos dos grupos de UEs con dos UEs en cada uno, para los cuales sus condiciones de canal sean óptimas en todos los PRBs, y cuenten con el mismo perfil de tráfico, por ejemplo 3,2 Mbps implementado como paquetes 40 kB generados cada 100 ms. Por otro lado, utilizamos las implementaciones por defecto de los *schedulers* tanto *intra slice* como *inter slice*, los cuales dividen equitativamente los recursos sin tomar en consideración ninguna métrica. En este escenario, si el manejo de paquetes funciona correctamente, se espera ver que cada UE obtenga un *throughput* similar.

En este punto es relevante aclarar que las funcionalidades que están siendo verificadas son únicamente las del núcleo del simulador, debido a que la generación

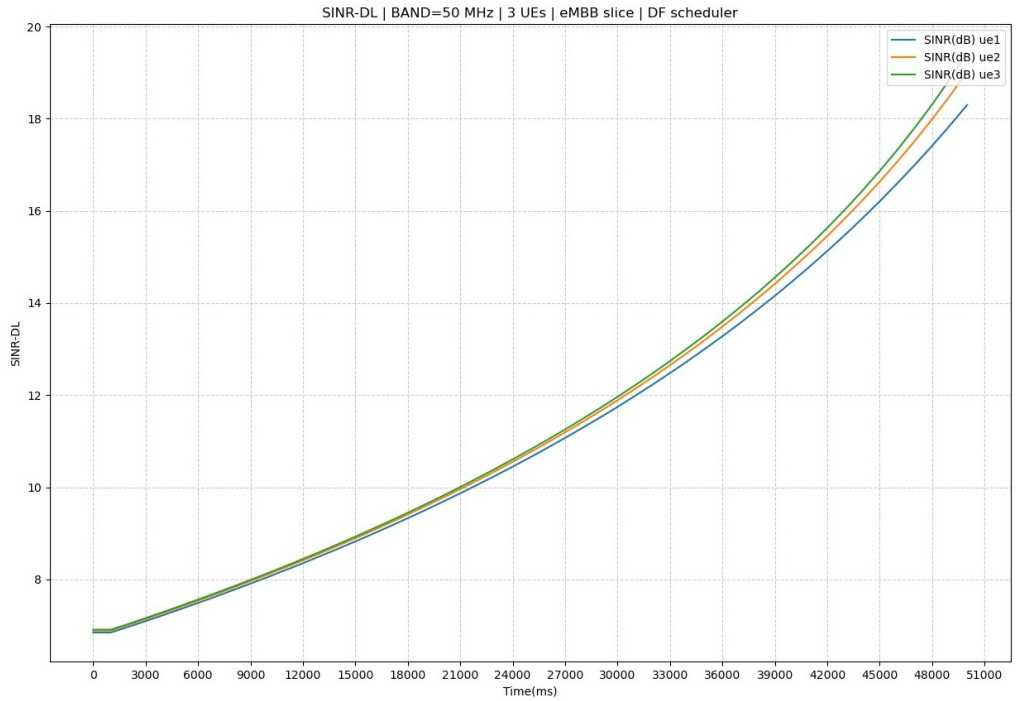

Figura 4.6: Gráfica de SNR de tres UEs generados a partir del escenario *Outdoor 1* del *framework* DeepMMIMO.

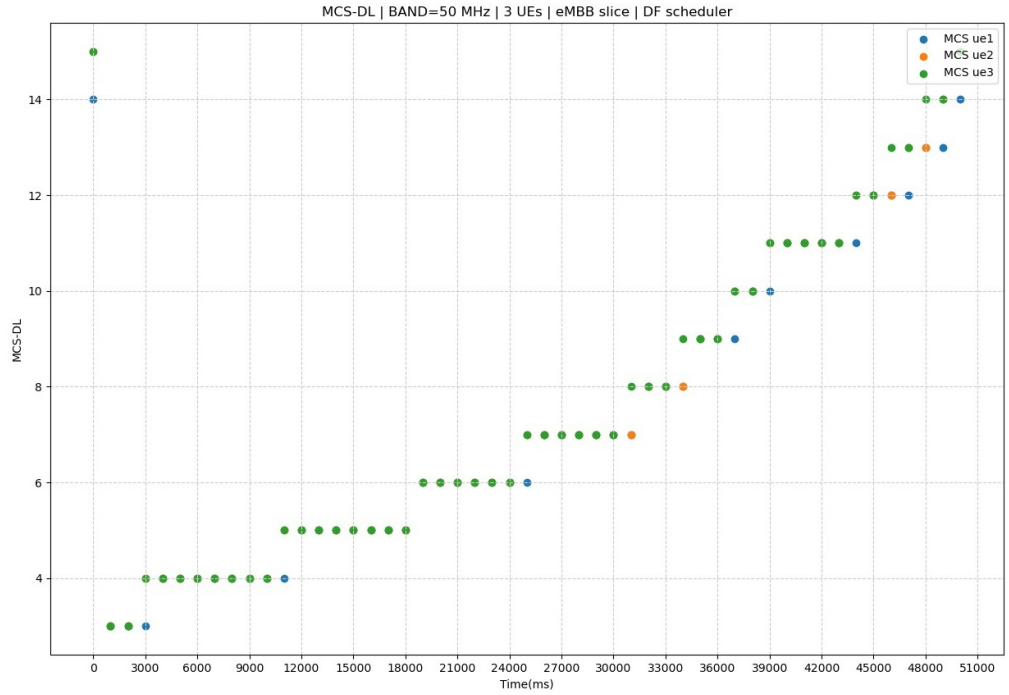

Figura 4.7: Gráfica de MCS de tres UEs generados a partir del escenario *Outdoor 1* del *framework* DeepMIMO.

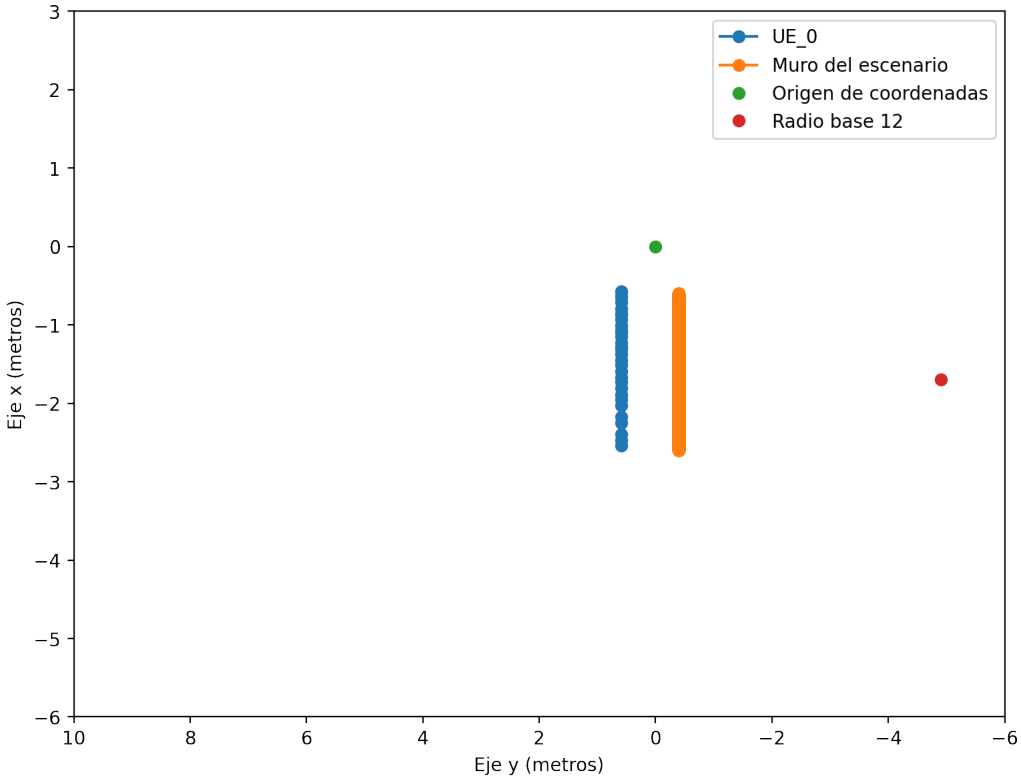

Figura 4.8: Posiciones que toma el usuario en el escenario *I2 (Indoor 2) Blockage Scenario* en la validación de rebotes, el usuario se encuentra justo detrás del muro del escenario.

de datos del canal y los procesos de *schedulers* no influyen. Esto se debe a que, por un lado, los valores de estado del canal fueron construídos de forma tal que siempre valgan 40 dB en todos los PRBs y por otro lado, a que los *schedulers* utilizados tienen implementaciones muy sencillas, los cuales solamente dividen los recursos equitativamente.

En la figura 4.10 se puede ver que el *throughput* varía alrededor de 6,4 Mbps para cada *slice*, que corresponde al doble del *throughput* para cada UE, lo cual es el comportamiento esperado. Por otro lado, en las figuras 4.11 y 4.12 se muestra el *throughput* para los grupos de usuarios *UEgroup_0* y *UEgroup_1* respectivamente, los cuales oscilan al rededor de 3,2 Mbps que es lo esperado para cada UE. Si bien las gráficas presentan muchas variaciones debidas a la aleatoriedad del modelo de generación de paquetes, podemos validar a grandes rasgos que el *throughput* se comporta como esperábamos. Con esta verificación validamos de forma primaria que las funcionalidades del núcleo del simulador funcionan correctamente.

4.3. Compatibilidad hacia atrás

Para validar la compatibilidad entre versiones, es decir, que los *scripts* creados para la primera versión continúen funcionando y su resultado continúe siendo el

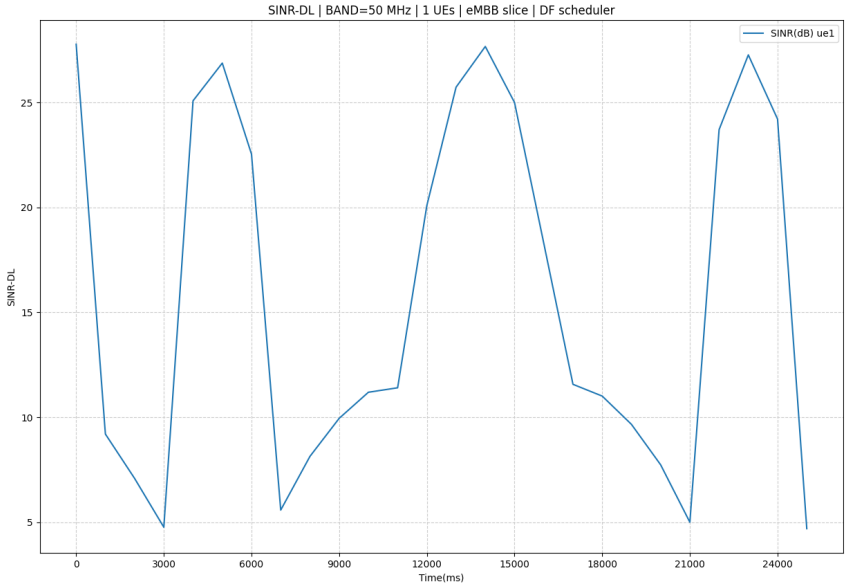

Figura 4.9: Gráfica de SNR de un UE que se mueve horizontalmente y rebota, generado a partir del escenario *I2 Blockage Scenario* del *framework* DeepMIMO.

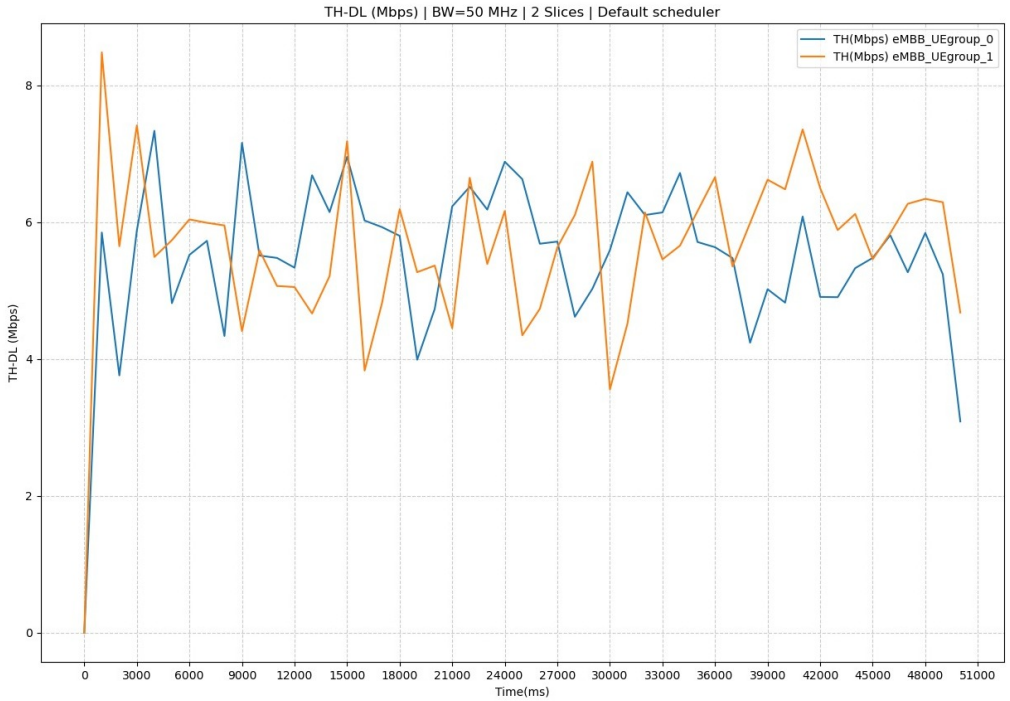

Figura 4.10: Gráfica del *throughput* de ambas *slices* de la simulación

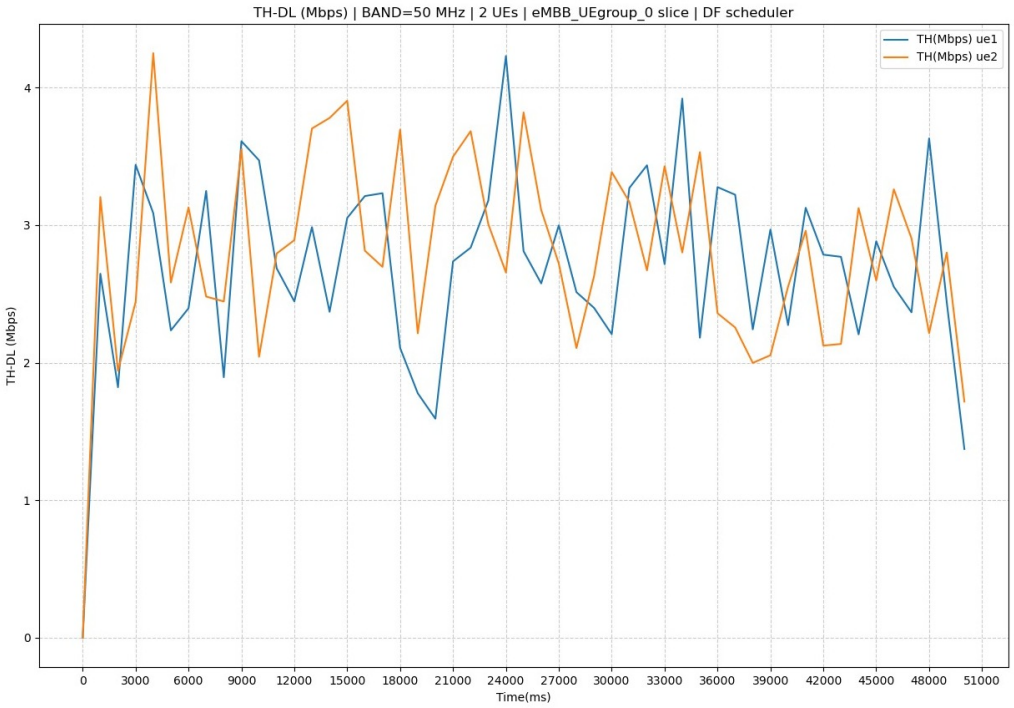

Figura 4.11: Gráfica del *throughput* en la *slice* del grupo de usuarios *UEgroup_0*.

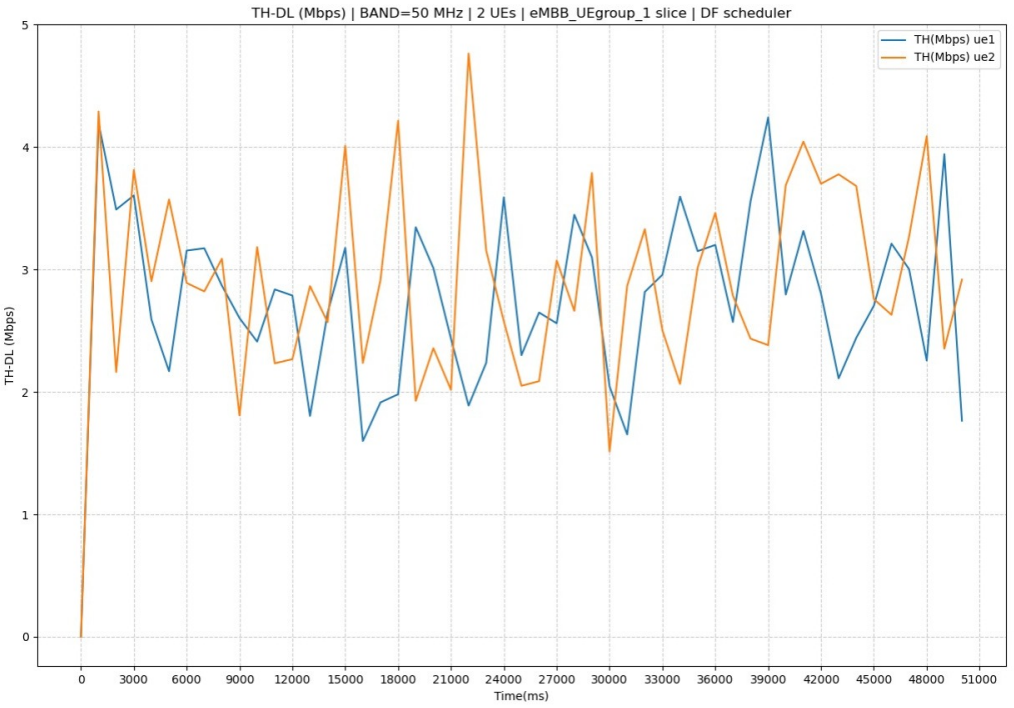

Figura 4.12: Gráfica del *throughput* en la *slice* del grupo de usuarios *UEgroup_1*.

mismo [1], es que ejecutamos *scripts* creados para la primera versión en ambas versiones y comparamos sus resultados.

Para esta validación elegimos utilizar tres grupos de UEs, cada uno utilizando un perfil de tráfico distinto entre los descritos en la sección 1.1, con la finalidad de cubrir la mayor cantidad de casos de uso posibles. A continuación se describen cada uno de estos grupos de UEs y luego se comparan los resultados.

El grupo de UEs con perfil de tráfico eMMB fue definido con dos UEs de *downlink*, cada uno con *throughput* de 4 Mbps implementados mediante una tamaño de paquetes de 5 kB y un tiempo entre ellos de 10 ms [2]. Luego, el grupo de UEs URLLC fue definido exigiendo una latencia menor a 5 ms, para el cual se utilizaron dos UE de *uplink* con un *throughput* de 2 Mbps cada uno. Por último, el grupo de UEs de tipo mMTC fue definido mediante diez UEs de *uplink*, en donde cada uno de ellos tiene un *throughput* de 12 kbps. Para una explicación más detallada de los distintos grupos de usuarios ver el anexo C.

En las gráficas de las figuras 4.13 y 4.15 se muestra para ambas versiones la evolución del *throughput* de *downlink* para los grupos de UEs definidos. En estos se puede ver que el compartamiento general es el mismo: la *slice* del grupo eMMB tiene un *throughput* que varía entre 7,5 Mbps y 8 Mbps para ambas versiones, y el *throughput* de los demás UEs es 0 Mbps.

Para el caso de *uplink* ocurre algo similar, si vemos las gráficas de las figuras 4.14 y 4.16 las cuales corresponden a la evolución del *throughput* de *uplink* para ambas versiones, se puede observar que el mismo oscila en torno a 4 Mbps para el grupo de UEs de URLLC, y en torno a 250 kbps para el grupo de m-MTC. Por otro lado, el grupo de UEs de eMMB no tienen *throughput* de *uplink* y por lo tanto se mantienen en 0 Mbps.

Del análisis anterior queda claro que ambas versiones del simulador se comportan de la misma manera tanto para *uplink* como para *downlink*, con lo cual validamos de manera primaria la compatibilidad entre versiones.

4.4. *Scheduler*

La validación del *scheduler* es una tarea compleja de realizar, ya que en muchos casos es díficil determinar que decisiones tomará. Pero existen algunas situaciones en las que la salida es esperable.

Para realizar la validación se eligió el escenario *Outdoor 1* C.2 y dos usuarios dinámicos. Estos usuarios comienzan alejados y sin compartir rayo principal, pero a medida que se mueven, comienzan a acercarse hasta que la diferencia en sus rayos principales es de 10 grados. En la figura 4.17 se muestra el movimiento de los usuarios de esta validación y en la figura 4.18 se muestra la diferencia en el ángulo de sus rayos principales. Un resultado esperable en este escenario, sería que

[1] Debido a que hay varios componentes aleatorios en la simulación, tales como el tiempo entre paquetes generados y el tamaño de los mismos, es que entendemos que el resultado sea el mismo como que tenga un comportamiento similar.

[2] En la sección 2.3 se explica la forma en la cual se definen simulaciones para la primer versión.

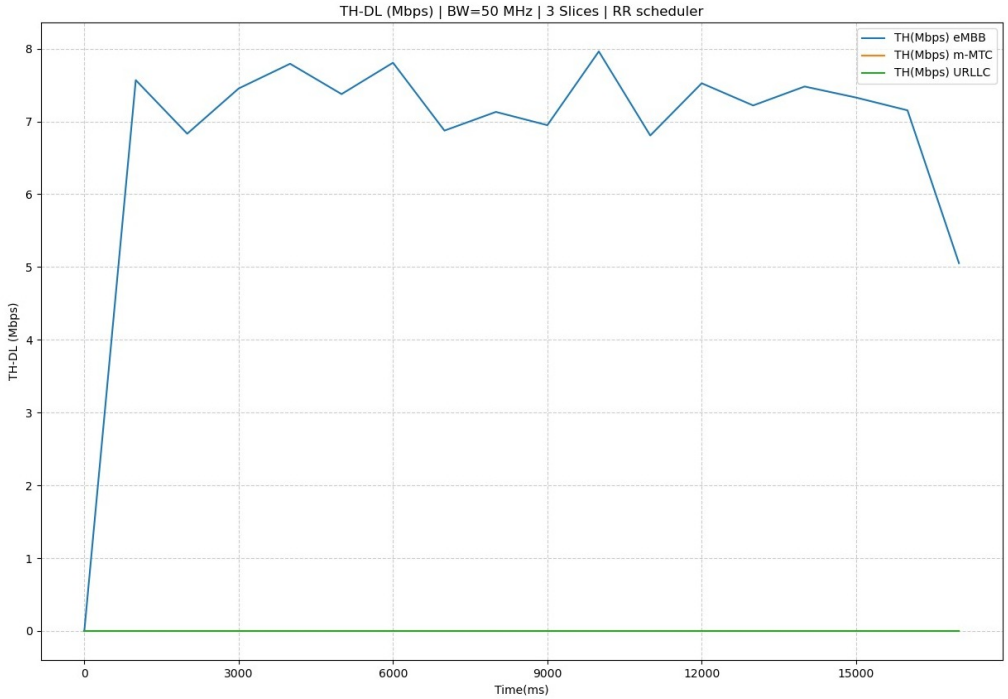

Figura 4.13: Gráfica del *throughput* de *downlink* dividido por *slice* mediante la versión 1 del simulador.

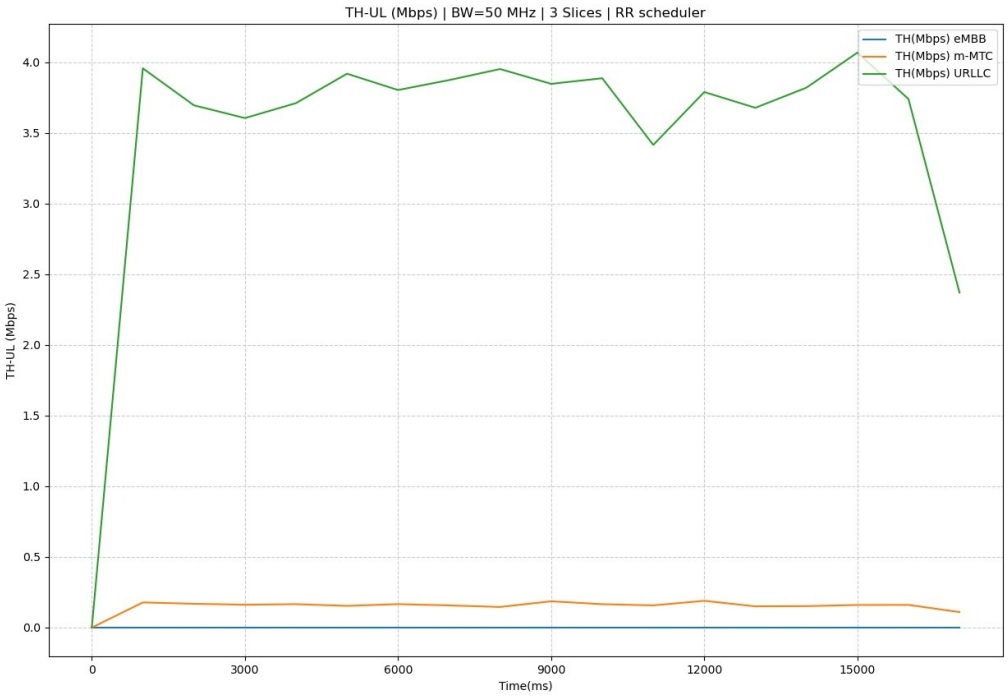

Figura 4.14: Gráfica del *throughput* de *uplink* dividido por *slice* mediante la versión 1 del simulador.

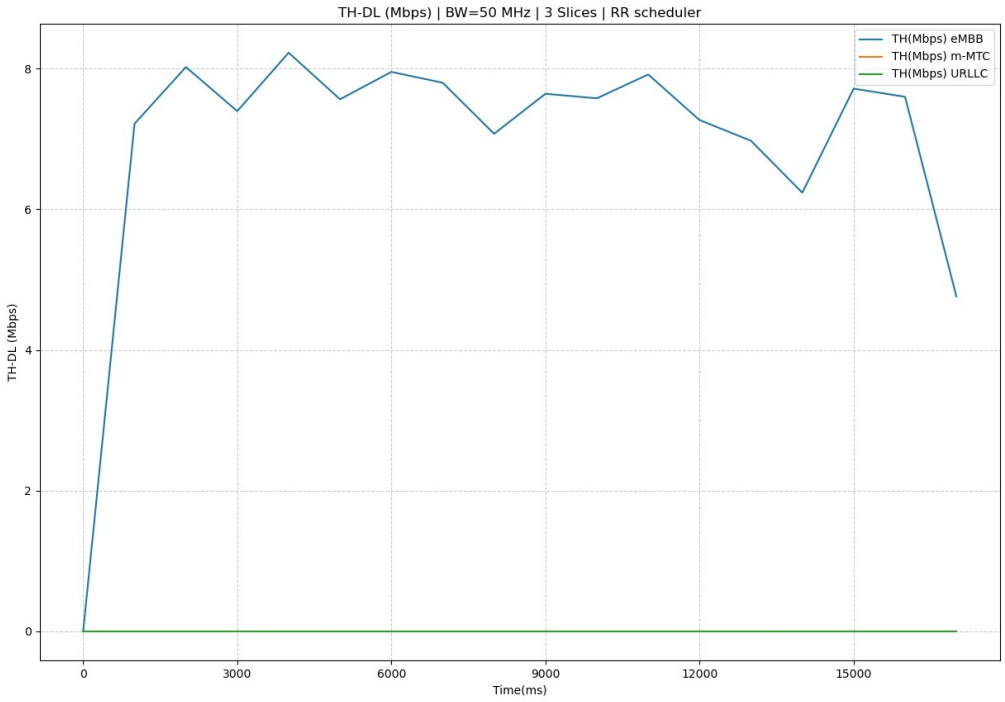

Figura 4.15: Gráfica del *throughput* de *downlink* dividido por *slice* mediante la versión 2 del simulador.

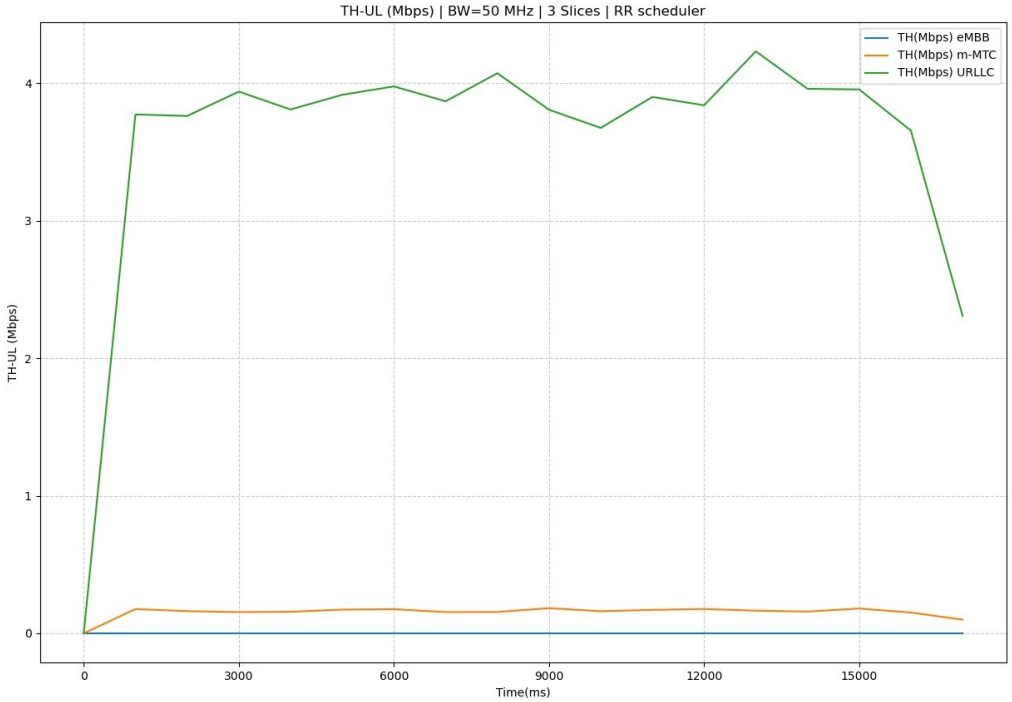

Figura 4.16: Gráfica del *throughput* de *uplink* dividido por *slice* mediante la versión 2 del simulador.

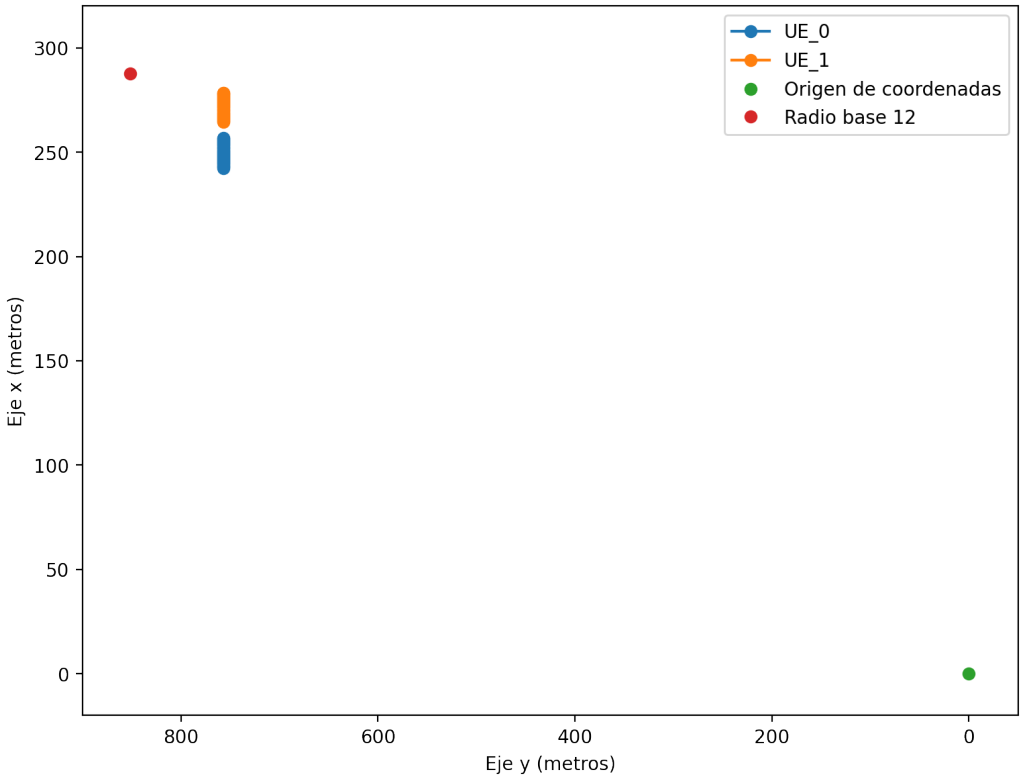

Figura 4.17: Gráfica del movimiento realizado por los usuarios en el escenario *Outdoor 1* utilizado para la validación del *scheduler*.

los usuarios comiencen compartiendo recursos hasta que se acerquen lo suficiente y dejen de compartir rayo principal. Es importante destacar que para esta prueba se fijo en 10 grados en el *scheduler*, el umbral mediante el cual se decide si dos usuarios comparten rayo principal.

Se verificó que al utilizar los datos generados como entrada al simulador se obtuvo uno de los resultados posibles. Al comienzo de la simulación hasta los 4 segundos todos los PRBs fueron compartidos por los usuarios presentes. Una vez que los usuarios se acercaron lo suficiente y la diferencia entre ambas componentes de su rayo principal pasó a ser menor a 10 grados, los PRBs disponibles se dividieron entre los dos usuarios. En las figura 4.19 se muestra la asignación de recursos obtenida, donde las líneas azules muestran los recursos asignados al usuario 1, las amarillas los recursos asignados al usuario 2,las verdes los recursos asignados a ambos usuarios y en color violeta se muestran los instantes en que ningún recurso fue asignado a los usuarios. Esto último sucede cuando los usuarios se desconectan momentáneamente de la slice por estar inactivos, al ser los únicos usuarios de la slice ningún recurso es asignado. En la figura 4.20 se muestra ampliada la zona de la figura 4.19 en que los usuarios dejan de compartir recursos, que coincide con el instante en que los usuarios comienzan a compartir rayo principal. Esto es aproximadamente a los 4 segundos de simulación, a partir de ese instante ya no se

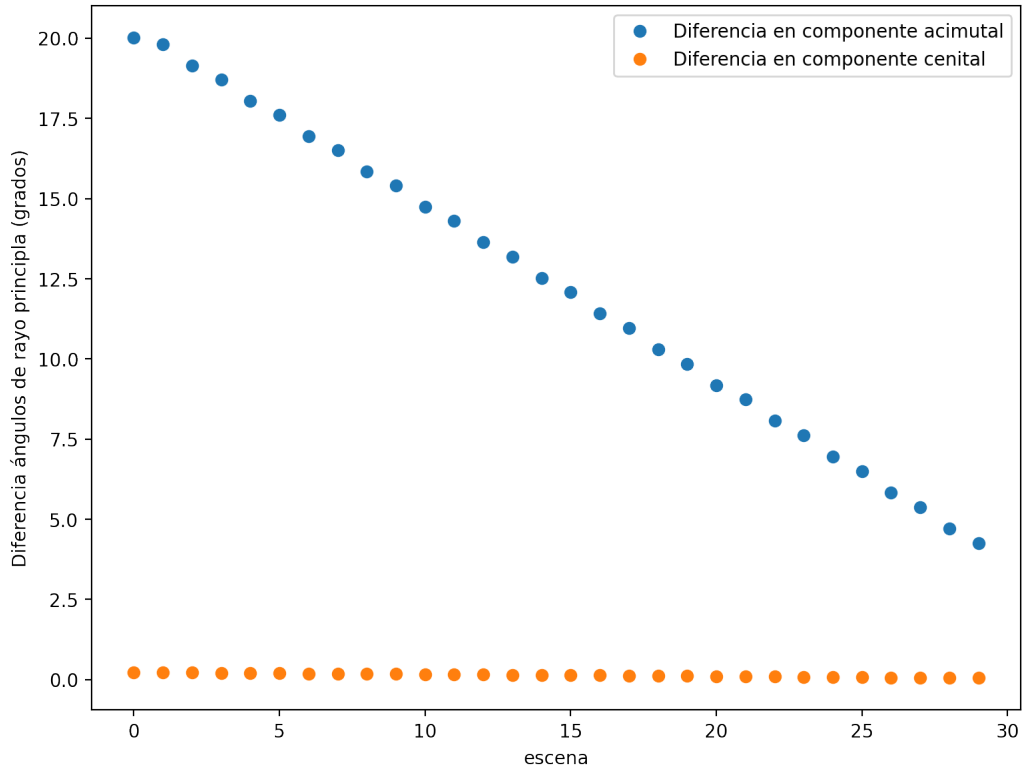

Figura 4.18: Diferencia en las distintas escenas de las componentes cenital y acimutal del rayo principal de los dos usuarios utilizados para la validación del *scheduler*.

pueden apreciar líneas verdes en la imagen.

4.5. Resumen del capítulo

En este capítulo se validaron de forma aislada las funcionalidades desarrolladas las cuales se dividen en tres grupos: validación del modelo de canal inalámbrico, validación de las modificaciones realizadas sobre el simulador Py5cheSim y validación del *scheduler*.

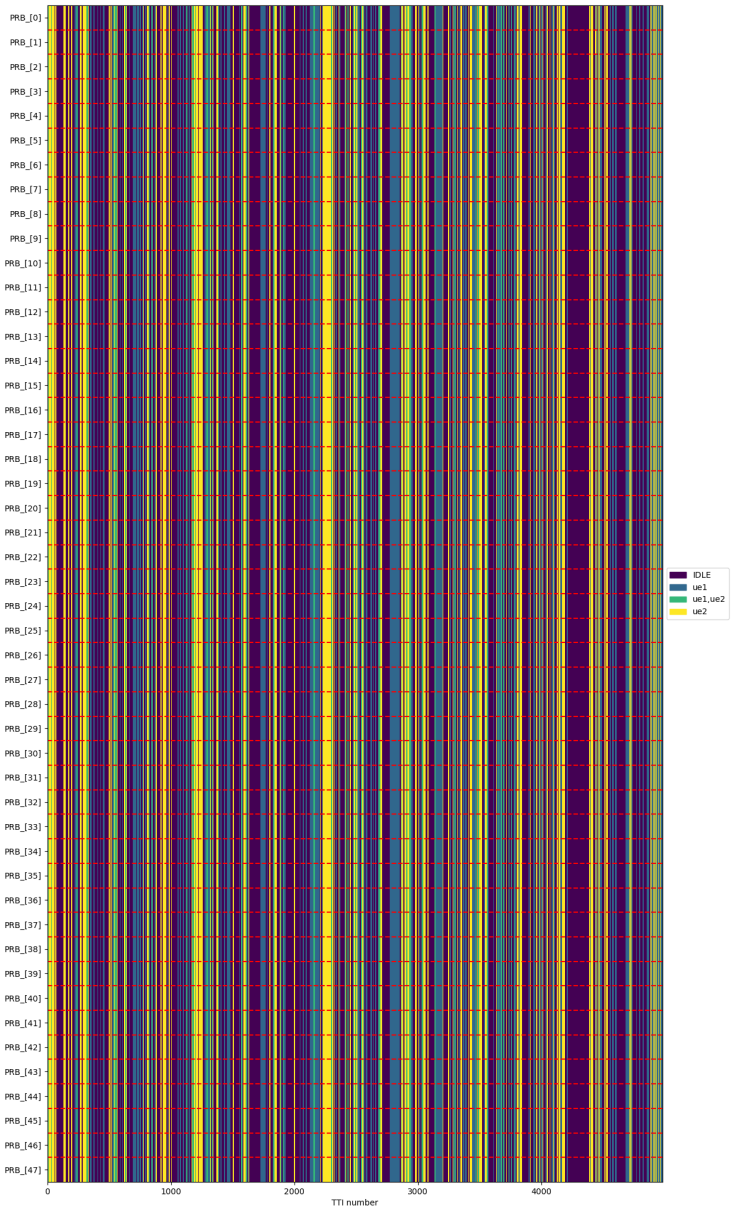

Figura 4.19: Asignación de recursos obtenida al utilizar dos usuarios que se acercan hasta compartir rayo principal. Líneas azules corresponden a recursos asignados al UE1, líneas amarillas recursos asignados al UE2 y líneas verdes recursos asignados a ambos usuarios.

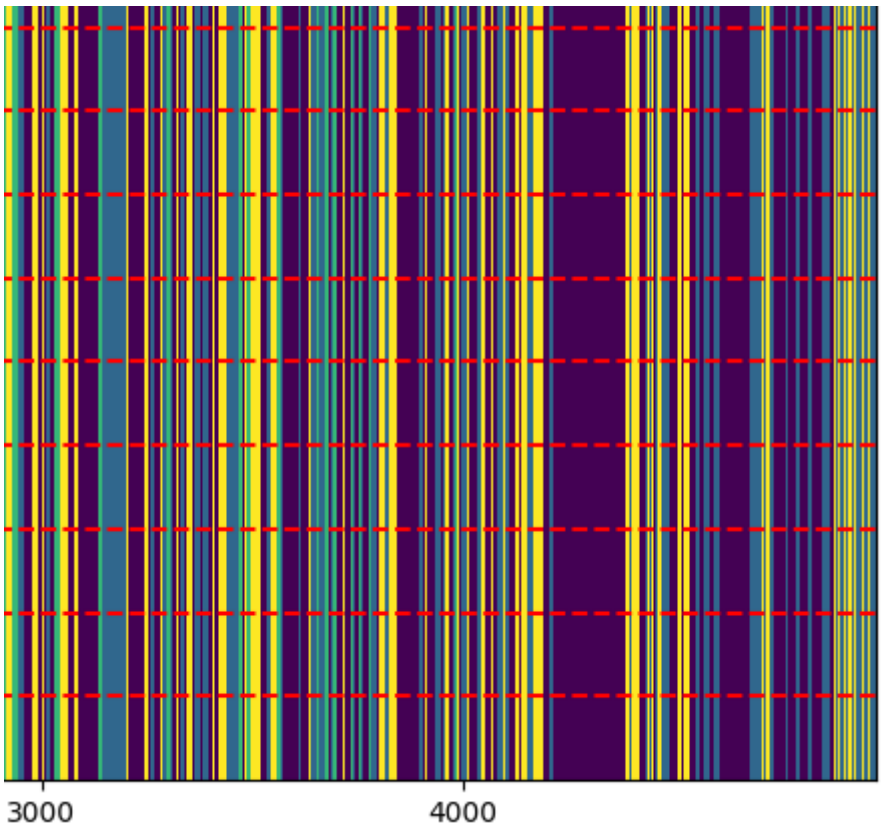

Figura 4.20: Ampliación de la figura 4.19, en la zona donde los usuarios dejan de compartir recursos. A partir de los 4000 milisegundos ya no se pueden apreciar líneas verdes en la figura.

Capítulo 5

Conclusiones

En este proyecto de grado se obtuvo una nueva versión del simulador Py5cheSim, una herramienta gratuita, *open source* desarrollada en Python para el estudio de asignación de recursos en redes 5G. El código del *software* está disponible para su estudio y utilización en GitHub. Para llevar a cabo este objetivo se logró estudiar el simulador en su primera versión, adquiriendo el conocimiento y manejo del *software* para su posterior modificación.

Se investigó la tecnología 5G NR y se incorporaron conocimientos que eran desconocidos por los integrantes hasta el momento, como la estructura de la trama, la asignación de recursos y los sistemas de MIMO masivo entre otros. Estos conocimientos fueron fundamentales para el diseño y desarrollo de las mejoras implementadas en el simulador.

Otro de los objetivos del proyecto era integrar un *framework* para mejorar la implementación del canal del simulador Py5cheSim. Para lograrlo se estudió un conjunto de *frameworks*, entre ellos DeepMIMO, el elegido para ser integrado a Py5cheSim. Al principio de este proyecto se supuso que DeepMIMO otorgaba escenarios dinámicos donde los usuarios tenían la posibiliad de moverse, sin embargo esto no fue así. El estudio y posterior trabajo sobre DeepMIMO fue fundamental para brindar mayor realismo al simulador mejorando el modelo del canal, generando escenarios dinámicos con usuarios que tienen la posibilidad de moverse.

Otro aporte interesante llegó de concretar el objetivo de implementar un nuevo *scheduler* que utilizara las funcionalidades mejoradas. Esto se logró estudiando un algoritmo de asignación de recursos diseñado para asignar uno o más PRBs a un grupo de usuarios que utilizan MU-MIMO. Esto permite una asignación de recursos más acercada a la realidad de las existentes previamente.

Más allá de las mejoras implementadas, esta versión sigue teniendo muchas simplificaciones al igual que la primera versión [14]. En primer lugar, mejorar el modelo de error para obtener resultados incluso más realistas y teniendo en cuenta esto, podría mejorarse la asignación MSC. Otra mejora podría ser que se admita más de un *bearer* por UE e incluir paquetes ACK, por ejemplo. Se puede mejorar la gestión de las *slices*, ya que no se tiene en cuenta los cambios de tráfico durante la simulación. Por último, no se tiene en cuenta la señalización de capas inferiores. Todas estos aportes harían de las simulaciones más precisas, pero agregarían al

simulador una complejidad que hoy en día no presenta [14]. Otro aspecto en el que se podría trabajar es en una interfaz de usuario para permitir definir simulaciones y presentar los resultados de la misma de forma sencilla sin necesidad de escribir el *script* en Python.

Apéndice A

Conceptos teóricos

A.1. Rayo principal

El objetivo de esta sección es fundamentar el uso del rayo principal como un dato relevante para un *scheduler* que soporta el uso de MIMO masivo. El mismo se define mediante las componentes azimutal y cenital del ángulo definido por el camino de mayor potencia entre la radiobase y el UE, obtenido mediante el *framework DeepMIMO*. Para obtener una descripción más detallada ver la sección 3.2.

A.1.1. Fundamento teórico

Como se explicó en la sección 2.2, la comunicación entre la radiobase y un UE está determinada por el uso de una matriz de *precoding*, la cual puede ser implementada de forma digital o analógica, pero en cualquier caso tiene el mismo resultado: crear patrones de radiación en dirección al UE, ya sea de forma directa en el caso de existir línea de vista, o con caminos alternativos en el caso de explotar el multicamino. A continuación se listan las diferentes implementaciones que permiten aprovechar el uso de MIMO masivo mediante esta técnica.

- ***Beamforming***: utilizado en FR2, donde la multiplexación espacial a un UE no es algo crucial debido a que en estas frecuencias el espectro no es un recurso tan escaso, se envía un único flujo de datos mediante un patrón radiación de un solo lóbulo principal.
- **SU-MIMO**: Consta del envío de varias *layers* (flujos de datos distintos) a un mismo UE mediante el multiplexado espacial de las mismas, es decir, cada *layer* se envía sobre un patrón de radiación diferente, lo que en definitiva genera un patrón con varios lóbulos principales.
- **MU-MIMO**: Se utiliza una misma matriz de precoding para servir varios UE simultáneamente. En este caso también se genera un patrón de radiación con varios lóbulos principales, pero cada uno lleva un flujo de datos para un UE diferente.

Debido a lo anterior, un asignador de recursos no debe generar el mismo patrón de radiación para UEs distintos sobre la misma frecuencia, porque en caso de hacerlo, estos recibirán una señal interferida. Por lo tanto, resulta natural dividir los alrededores de la radiobase en regiones donde solo puede existir un rayo principal para una frecuencia determinada.

A.1.2. Validación

Para respaldar la división de los alrededores de la radiobase, la cual fue explicada en la sección anterior, es que tomamos un escenario del *framework DeepMIMO* y buscamos pares de UEs que compartan rayo principal. Si estos tienen una matriz de canal similar[1], entonces deberán utilizar la misma matriz de *precoding* y por lo tanto generar el mismo patrón de radiación.

Para esto utilizamos el escenario *Indoor 2*, el cual fue presentado en la sección 2.4 y se puede ver en las figuras 2.7 y 2.8. En este, buscamos el par de UE los cuales compartan rayo principal y la distancia entre ellos sea la más grande de todo el escenario. En este punto es necesario aclarar que el criterio para decidir que dos UEs: UE1 y UE2 comparten rayo principal es que:

$$|\phi 2 - \phi 1| < \Delta\phi \ \ \& \ \ |\theta 2 - \theta 1| < \Delta\theta$$

En donde $\phi 1$ y $\phi 2$ son las componentes azimutales del rayo principal de UE1 y UE2 respectivamente, y de igual manera, $\theta 1$ y $\theta 2$ son las componentes cenitales de UE1 y UE2. Por otro lado, para los valores de $\Delta\phi$ y $\Delta\theta$ tomamos 0,1° en ambos casos a modo de ejemplo. En estas condiciones encontramos que la distancia entre UE1 y UE2 es de $0{,}33m$ y poseen las siguientes características:

$$\phi 1 = 59{,}597^\circ \quad \theta 1 = 94{,}199^\circ$$
$$\phi 2 = 59{,}530^\circ \quad \theta 2 = 94{,}100^\circ$$

En la figura A.1 se puede ver el valor de las matrices de canal $M1$ y $M2$, las cuales corresponden a UE1 y UE2 respectivamente. Además, en la parte inferior se presenta la resta de ambas, cada una normalizada por su elemento $(1, 1)$. Podemos ver que todos las entradas de la matriz resultante de esta resta son elementos ordenes de magnitud más pequeños que los de cada matriz normaliza, con lo que concluimos que las matrices son similares y por lo tanto la afirmación inicial se ve respaldada.

[1]Se entiende similar como diferir en un factor multiplicativo complejo.

```
M1 = (8.476221182718291e-07-2.1949301753920736e-06j) *

[[ 1. -0.0e+00j -0.9+4.3e-01j  1. -2.3e-01j -0.8+6.2e-01j]
 [-0.9-4.1e-01j  1. -1.3e-02j -1. -2.0e-01j  1. -2.4e-01j]
 [ 1. +2.3e-01j -1. +2.1e-01j  1. +6.8e-08j -0.9+4.3e-01j]
 [-0.8-6.1e-01j  1. +2.2e-01j -0.9-4.1e-01j  1. -1.3e-02j]]

M2 = (1.06908294128516e-06-2.0321131160017103e-06j) *

[[ 1. -0.0e+00j -0.9+4.3e-01j  1. -2.2e-01j -0.8+6.2e-01j]
 [-0.9-4.2e-01j  1. -1.3e-02j -1. -2.0e-01j  1. -2.4e-01j]
 [ 1. +2.2e-01j -1. +2.1e-01j  1. +8.8e-08j -0.9+4.3e-01j]
 [-0.8-6.1e-01j  1. +2.1e-01j -0.9-4.2e-01j  1. -1.3e-02j]]

M1/M1[1,1] - M2/M2[1,1] =

[[ 6.0e-08+0.0e+00j -6.4e-04-1.4e-03j -1.2e-03-5.3e-03j  2.4e-03+3.1e-03j]
 [-6.1e-04+1.3e-03j  4.2e-07+2.7e-05j -1.4e-03+6.8e-03j -1.3e-03-5.3e-03j]
 [-1.2e-03+5.3e-03j -1.5e-03-6.8e-03j  6.0e-08-2.0e-08j -6.4e-04-1.4e-03j]
 [ 2.4e-03-3.2e-03j -1.2e-03+5.4e-03j -6.1e-04+1.3e-03j  3.0e-07+2.7e-05j]]
```

Figura A.1: Matrices de canal $M1$ y $M2$ de los UE1 y UE2 respectivamente, junto con la resta de sus normalizaciones por el primer elemento.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice B

Detalles de implementación

B.1. Estructura de archivos

A continuación se desglozan los archivos que componen el proyecto junto con su funcionalidad principal.

- **simulation.py y simulation_deepmimo.py**: Son los scripts que sirven de ejemplo para definir todo lo necesario para ejecutar una simulación en la primera y en la segunda versión del proyecto respectivamente. En ellos se definen los siguientes parámetros de la simulación:

  - **Ancho de banda**: Indica el ancho de banda en MHz de las portadoras para la simulación. En la primera versión se utiliza para calcular la cantidad de PRBs que se pueden asignar y en la segunda se utiliza solamente para mostrar resultados.
  - **Rango de frecuencia**: Es un *string* que puede tomar los valores FR1 o FR2 e indica si la simulación trabaja en una frecuencia central menor o mayor a $24MHz$ respectivamente.
  - **Banda**: Debe ser un string el cual indica la banda en la cual trabaja la simulación. Su valor debe ser coherente con el modo de duplexación: TDD o FDD.
  - **Modo de duplexación**: Boleano que indica si la duplexación es temporal o no, en caso de no serlo indica que la duplexación es en frecuencia. Se utiliza al momento generar los *Transport Blocks*.
  - **Inter slice scheduler**: Es un string que indica el intra slice scheduler a utilizar. Además, en algunos casos también indica algunos parámetros para el mismo.
  - **Tamaño del buffer de Bearer**: Indica el tamaño total en bytes que se pueden almacenar en los *buffers* de los *bearers*, tanto para transmisión como para recepción, antes de comenzar a descartar paquetes.
  - **Tiempo de simulación**: Indica la cantidad de milisegundos para el que se va a ejecutar la simulación.

- **Intervalo entre medidadas**: Establece la cantidad de milisegundos entre reportes consecutivos. Los mismos son utilizados posteriormente para mostrar gráficamente los resultados de la simulación.
- **Granularidad del inter slice scheduler**: Indica la cantidad de milisegundos entre asignaciones de recursos entre *slices*.
- **Celda**: Se debe indicar los distintos parámetros de la celda para la cual va a correr la simulación.
- **Grupo de UEs**: Se deben crear los diferentes grupos de UE los cuales van a participar en la simulación.

Además, para la versión nueva se debe especificar el directorio del escenario DeepMIMO que se desea utilizar.

- **Cell.py**: Almacena parámetros específicos a la celda. También se encarga de crear el *inter slice scheduler* y de actualizar las estadíscias que se utilizan para el reporte final.
- **InterSliceSch.py y Scheds_Inter.py**: Contienen la implementación base del *inter slice scheduler* y sus clases hijas respectivamente. Se encarga de asignar recursos a las diferentes *slices* definidas en la simulación.
- **UE.py**: Contiene todo lo referido a UEs y a grupos de UEs. Sus métodos se encargan principlamente de instanciar UEs y de manejar la conectivadad de los mismos con la celda.
- **channel.py**: En este módulo se definen las clases que se encargan de manejar el estado del canal de los UEs. Sus métodos se encargan principalmenete de leer los archivos de canal y actualizar el estado del mismo.
- **Slice.py**: Este módulo principalmente se encarga de manejar la configuración de las *slices*, las cuales permiten dar soporte a los diferentes requerimientos de los grupos de UEs.
- **packet.py**: Su función principal es manejar las colas de paquetes, tanto a nivle de aplicación como de *bearer*.
- **Results.py**: Contiene las funciones que se encargan de presentar los resultados gráficamente.
- **IntraSliceSch.py y Scheds_Intra.py**: Contienen la implementación base del *intra slice scheduler* y sus clases hijas respectivamente. Estas se encargan de asignar los rescursos a los UEs en cada TTI. Los nuevos *intra slice schedulers* diseñados deben ser implementados en el ségundo módulo.
- **utilities.py**: Contiene algunas funciones auxiliares para dar soporte a los otros módulos.

B.2. Cambios específicos a nivel de clases

B.2.1. Aspectos generales

Como se explicó en la sección 3.2, los datos de SNR y rango para un par radiobase y UE son calculados para cada PRB base definido en la simulación. Esto es importante mencionarlo antes de entrar en detalle con las clases modificadas dado que es un factor común en muchos de los cambios realizados.

B.2.2. CellDeepMimo

Para la nueva versión se modificó el método de inicialización de la clase para que además de los atributos anteriores también maneje los suiguientes:

- **cant_prbs_base**: La celda debe manejar este dato debido a que ahora la cantidad de PRBs para un SCS de $15KHz$ no se calcula más en función de la banda y el ancho de banda, sino que viene dado desde la salida explicada en la sección 3.2.
- **bandwidth**: Ahora los nuevos datos son generados para una única portadora OFDM, lo que implica que es necesario pasar el dato de bandwidth a una lista para ser compatible con la versión anterior.

Por otro lado, se creó el método **json_to_dict_config** el cual se agrega para validar y traducir los parámetros en el archivo config.json con la finalidad de que sean compatibles con el simulador.

B.2.3. IntraSliceSchedulerDeepMimo

Esta clase fue creada para presentar un ejemplo de como manipular los PRBs bases en la nueva versión y como realizar la asignación de recursos a las diferentes *slices*. La misma implementa una asignación equitativa a lo largo del tiempo, siendo indeferente ante el estado del canal de los UEs de la *slice* en los PRBs base asignados y a los requerimientos de las mismas. Nuevas implementaciones de *inter slice schedulers* deben heredar de esta clase si desean utilizar las funcionalidades de la nueva versión. Para esta clase se modificaron y extendieron los siguientes métodos:

- **__init__**: Se modificó el método de inicialización para que tome como parámetros de entrada a cant_prbs_base, el cual es necesario debido a que en esta nueva versión los PRBs base se asignan a cada slice por índice, desde el 0 al $cant\_prbs\_base - 1$. Por otro lado, se agregaron los siguientes atributos a la clase:
  - **PRBs**: Guarda la cantidad de PRBs base que se pueden asignar a las diferentes slices.

  - **list_base_prb**: Almacena una lista conteniendo los índices de los PRBs base que se pueden asignar a las diferentes slices.
  - **assignation_start_index**: Guarda el índice de la slice por la cual debe comenzar la asignación de recursos en la siguiente ejecución de *resAlloc*. Esto parámetro se utiliza para mantener equitativa la distribución de recursos a los largo del tiempo, para el caso en que no se puede repartir la misma cantidad de PRBs base para cada *slice* en una ejecución de *resAlloc*. Este atributo puede ser innecesario en nuevas implementaciones de *inter slice schedulers* en la nueva versión del simulador.

- **resAlloc**: Extiende el método de mismo nombre de la clase padre para asignar recursos a cada *slice*. En esta nueva versión la asignación debe especificar los índices de los PRBs bases que fueron asignados a cada *slice*.

- **get_equitative_prb_division**: Este método se encarga de generar las listas de índices de PRBs que se deben asignar a cada slice teniendo en cuenta el atributo assignation_start_index para mantener equitativa la distribución de PRBs a lo largo del tiempo entre las diferentes *slices*.

- **createSlice**: Este método extiende al de la clase base solamente para instanciar una *slice* de la nueva versión.

B.2.4. SliceDeepMimo

Para esta clase se modifican los siguientes métodos:

- **__init__**: Se extiende el método de la clase base para modificar y agregar los siguientes atributos a la clase hija:

  - **PRBs**: Este atributo se inicializa en 0 y se modifica en las sucesivas ejecuciones de *resAlloc* de la clase *InterSliceSchedulerDeepMimo* para mantener la compatibilidad con la versión 1. Este atributo se escribe con la cantidad de PRBs asignados a la *slice* según corresponda de acuerdo con su SCS y es utilizado para presentar los resultados de la asignación.
  - **assigned_base_prbs**: Almacena una lista con los ínidce de los PRBs base asignados a la *slice* en la ejecución del método *resAlloc* de la clase *InterSliceScheduler*.

- **createSliceSched**: Este método sobrescribe al de la clase padre con la finalidad de instanciar los distintos *intra slice schedulers* creados exclusivamente para la nueva versión. Los *intra slice schedulers* de versiones diferentes no son compatibles entre si debido a que en la versión 1 se asignan solamente la cantidad de PRBs y en la versión 2 es necesario especificar la cantiadad y los índices de los PRBs base asignados al UE.

- **updateConfig**: Sobrescribe al método de mismo nombre de la clase padre para asignar la lista de PRBs base y la cantidad de PRBs asignados a la *slice* según el SCS configurado para la misma.

B.2.5. UeGroupDeepMimo

Esta clase tiene el objetivo de instanciar los distintos UEs y actualizar el estado del canal de los mismos en base a los archivos generados por el script presentado en la sección 3.2. A continuación se presentan los métodos creados y los modificados con respecto a la versión anterior:

- **__init__**: Este método extiende la inicialización de la clase padre para agregar los siguientes atributos:

  - **ue_group_dir**: Guarda el directorio de los archivos de estado del canal designados al grupo de UE.
  - **current_scene**: Se inicia en 0 y tiene el objetivo de guardar el índice del archivo de canal que debe ser leído en la próxima actualización de datos del canal.
  - **is_dynamic**: Es un boleano que indica si el grupo de usuarios está basado en un grupo de usuarios con movimiento o no.
  - **scene_duration**: Indica el tiempo en milisegundos entre dos archivos de estado de canal de índices consecutivos.

- **setInitialSINR**: Sobrescribe el método de la clase padre con el objetivo de establecer el estado del canal para la nueva versión de UEs.

- **initializeUEs**: Extiende el método de la clase padre para inicilizar el nuevo tipo de usuario y en caso que el escenario sea dinámico se encarga de crear el proceso PEM para actualizar los datos del canal para el grupo.

- **read_ues_channel_status**: Este método se encarga de leer los archivos generados en la primer etapa de la simulación en el tiempo especificado y devolver los datos del canal en listas, donde cada índice representa un usuario.

- **update_ue_group_rl**: Actualiza el estado del canal para el grupo de UEs.

- **pem_update_ue_group_rl**: Utilizando los últimos dos métodos explicados, se encarga de leer y actualizar el estado del canal para los UEs del grupo. Por otro lado, actualiza el atributo current_scene que lleva cuenta del índice del próximo archivo a leer.

B.2.6. UeDeepMimo

En esta clase se agregaron y extendieron los siguientes métodos con el objetivo de soportar los nuevos datos de canal:

- **__init__**: Este método se extendió con la finalidad de agregar los siguiente atributos a la clase:
  - **assigned_base_prbs**: Se incializa como una lista vacía pero durante la simualción se utiliza para indicar los PRBs base que le fueron asignados al UE.
  - **assigned_layers**: Se actualiza en el momento que se asignan recursos al UE y es uno de los datos utilizados para calcular la cantidad de bits que soporta el transport block.

  Por otro lado, también se encarga de instanciar el nuevo modelo de *radio link* y de inicializar los datos del canal para dicho UE.
- **update_radio_link_status**: Actualiza el estado del canal para el UE.
- **add_resources**: Agrega los PRBs indicados a la lista de PRBs base asignados al UE. También actualiza la cantidad de layers soportados por el mismo.
- **has_packet_in_bearer**: Este método se creó para ser utilizado por el *intra slice scheduler* cuando necesite saber los UEs que tienen paquetes para enviar.

B.2.7. RadioLinkDeepMimo

En esta clase se almacenan los datos del canal de un UE para cada PRB base definido en la simulación. Para realizarlo se modificaron y crearon los siguientes métodos:

- **__init__**: En la inicialización se crean los siguientes atributos para la clase:
  - **snr**: Es una lista que guarda los valores de SNR del UE para cada PRB base de la simulación. Los valores de SNR se almacenan en decibeles.
  - **rank**: Es una lista que almacena el rango de la matriz de canal entre el UE y la radiobase según se explica en la sección 3.2.
  - **degree**: Es una lista de dos elementos que guarda la dirección de llegada del rayo principal al UE, el mismo se especifica de la siguiente manera: $(componente\_acimutal, componente\_cenital)$ en donde cada componente se expresa en grados.
  - **linkQuality**: Este atributo se mantiene con respecto a la versión anterior por motivos de compatibilidad con la etapa de presentación de datos. Cuando se actualiza el estado del canal, el atributo se calcula como el promedio de los valores del SNR para todos los PRBs base.

- **update_link_status**: Actualiza el estado del canal con los valores especificados en los parámetros.
- **get_radio_link_quality_over_assigned_prbs**: Devuelve una tupla de la forma $(SNR, rank)$ donde:
  - **SNR**: Se calcula como el promedio de los SNRs sobre los PRBs base asignados.
  - **rank**: Se calcula como el rango mínimo sobre los PRBs asignados.

  Ambas componentes puden ser utilizadas por el *intra slice scheduler* para asignar una cantidad de *layers* y un esquema de modulación al UE.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice C

Validación

C.1. Grupos de UEs en compatibilidad hacia atrás

Parámetros generales del escenario:

- Ancho de banda en MHz: 50.
- Rango de frecuencia: FR1.
- Banda utilizada: B1.
- Modo de duplexación preferido: FDD.
- Tamaño del buffer de UE en bytes: 81920.
- *Scheduler inter slice*: *Round Robin*.
- Tiempo de simulación en milisegundos: 20000.
- Tiempo entre medidas en milisegundos: 1000.
- Granularidad temporal del *inter slice scheduler*: 3000.

Los grupos de UEs utilizados tienen las siguientes carecterísticas:

- **eMMB**:
  - Cantidad de UEs de DL: 2.
  - Cantidad de UEs de UL: 0.
  - Tamaño de paquetes de DL en bytes: 5000.
  - Tamaño de paquetes de UL en bytes: 0.
  - Tiempo entre paquetes DL en milisegundos: 10.
  - Tiempo entre paquetes UL en milisegundos: 0.
  - Etiqueta del grupo (identificar en los gráficos): eMMB.

- Latencia máxima tolerada en milisegundos: 20 ms.
- *Intra slice scheduler*: *Round Robin*.
- Modo MIMO: *Single User*. *Capas en MIMO*: 4.
- SINR: Estático de valor 37 dB.

▪ **URLLC**:

- Cantidad de UEs de DL: 0.
- Cantidad de UEs de UL: 2.
- Tamaño de paquetes de DL en bytes: 0.
- Tamaño de paquetes de UL en bytes: 1500.
- Tiempo entre paquetes DL en milisegundos: 0.
- Tiempo entre paquetes UL en milisegundos: 6.
- Etiqueta del grupo (identificar en los gráficos): URLLC.
- Latencia máxima tolerada en milisegundos: $5ms$.
- *Intra slice scheduler*: *Round Robin*.
- Modo MIMO: *Multiple User*. *Capas en MIMO*: 4 UE simultaneos compartiendo los mismos recursos con 3 capas cada uno.
- SINR: Estático de valor 37 dB.

▪ **mMTC**:

- Cantidad de UEs de DL: 0.
- Cantidad de UEs de UL: 10.
- Tamaño de paquetes de DL en bytes: 0.
- Tamaño de paquetes de UL en bytes: 150.
- Tiempo entre paquetes DL en milisegundos: 0.
- Tiempo entre paquetes UL en milisegundos: 100.
- Etiqueta del grupo (identificar en los gráficos): m-MTC.
- Latencia máxima tolerada en milisegundos: 20 ms.
- *Intra slice scheduler*: *Round Robin*.
- Modo MIMO: *Single User*. *Capas en MIMO*: 1.
- SINR: Estático de valor 37 dB.

C.2. Escenario *Outdoor 1*

El escenario *Outdoor 1* fue presentado parcialmente en la sección 4.1.2 mediante las figuras 4.3 y 4.4. A continuación se detallan las características del mismo [3].

radiobases

- 18 radiobases, todas ellas a una altura de 6 m con respecto al suelo.
- Cada elemento del arreglo de antenas es una antena isotrópica.
- La calle principal del escenario tiene 12 radiobases, 6 de cada lado, desde la *BS1* a la *BS12*.
- Las radiobases de un lado de la calle están separadas 52 m con respecto a las del otro lado.
- 100 m de separación entra las radiobases:
  - *BS1*, *BS3* y *BS5*.
  - *BS2*, *BS4* y *BS6*.
  - *BS7*, *BS9* y *BS11*.
  - *BS8*, *BS10* y *BS12*.
- $62m$ de separación entre las radiobases:
  - *BS6* y *BS8*.
  - *BS5* y *BS7*.
- Las radiobases *BS13* a la *BS18* cubren principalmente la calle secundaria, 3 de cada lado de la misma.
- 150 m de separación entra las radiobases:
  - *BS13*, *BS15* y *BS17*.
  - *BS14*, *BS16* y *BS18*.
- $52m$ de separación entre las radiobases:
  - *BS13* y *BS14*.
  - *BS15* y *BS16*.
  - *BS17* y *BS18*.

Usuarios

- Tres grillas uniformes de usuarios (UGs): *UG1*, *UG2* y *UG3*.
- $1,184,923$ usuarios en total.
- El primer usuario en cada grilla siempre tiene las coordenadas (x, y) más bajas.
- La altura con respecto al suelo de cada usuario es 2 m.

- Cada elemento del arreglo de antenas es una antena isotrópica.
- *UG1*:
  - Distribuída a lo largo de la calle principal, con un largo de 550 m y un ancho de 35 m.
  - Comienza desde la derecha de la figura 4.4 a $15m$ del comienzo de la calle principal.
  - Termina en la izquierda de la figura 4.4 antes de alcanzar el final de la calle principal.
  - Contiene 2751 columnas con 181 usuarios por fila (la columna va a lo largo del eje y).
  - 20 cm de separación entre usuarios.
  - 497, 931 usuarios en total.
- *UG2*:
  - Distribuída a lo largo de la mitad inferior de la calle secundaria (ver figura 4.4).
  - Se accede desde la columna 2752 hasta la 3852 con un total de 1101 filas en total.
  - 181 usuarios por fila.
  - 20 cm de separación entre usuarios.
  - 199, 281 usuarios en total.
- *UG3*:
  - Distribuída a lo largo de la mitad superior de la calle secundaria (ver figura 4.4).
  - Se accede desde la columna 3853 hasta la 5203 con un total de 1351 filas en total.
  - 361 usuarios por fila.
  - 10 cm de separación entre usuarios.
  - $487,711$ usuarios en total.

Vista de planta

- La calle principal (la horizontal en la figura 4.4) tiene 600 m de largo y 40 m de ancho.
- La calle secundaria tiene 440 m de largo y 40 m de ancho.
- Las dos calles tienen edificios a ambos lados.

- La altura de cada edificio se puede ver en la figura 4.4.
- Los edificios sobre la calle principal tienen todos una base de 30 m por 60 m.
- Los edificios sobre la calle secundaria tienen todos una base de 60 m por 60 m.
- En el modelo de propagación, cada camino del modelo de propagación tolera hasta 4 reflexiones.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] Wireless InSite 3D Wireless Prediction Software. RemCom Inc. `https://www.remcom.com/wireless-insite-em-propagation-software`.

[2] ARTES: Análisis de Redes, Tráficos y Estadísticas de Servicios. Facultad de Ingeniería, UdelaR. `https://iie.fing.edu.uy/investigacion/grupos/artes`, 2000.

[3] A. Alkhateeb. DeepMIMO: A generic deep learning dataset for millimeter wave and massive MIMO applications. In *Proc. of Information Theory and Applications Workshop (ITA)*, pages 1–8, San Diego, CA, Feb 2019.

[4] M. Alrabeiah, A. Hredzak, Z. Liu, and A. Alkhateeb. Viwi: A deep learning dataset framework for vision-aided wireless communications. In *submitted to IEEE Vehicular Technology Conference*, Nov. 2019.

[5] Hiroyuki Atarashi Anass Benjebbour, Koshiro Kitao. IMT-2020 Radio Interface Standardization Trends in ITU-R. *NTT DOCOMO Technical Journal*, 19(3), Jan. 2018.

[6] Bruno Benedetti. Red de Acceso Móvil 5G. `https://eva.fing.edu.uy/pluginfile.php/405745/mod_resource/content/5/RDA_MOVIL_5G_2022.pdf`.

[7] Chandra S. Bontu, Jagadish Ghimire, and Amr El-Keyi. Optimum resource allocation in mu-mimo ofdma wireless systems. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5, 2020.

[8] E. Dahlman. *5G Physical Layer Technologies*. John Wiley Sons, 2020. ISBN 978-1-11-952552-3.

[9] E. Dahlman. *5G NR The Next Generation Wireless Access Technology*. Academic Press, 2021. ISBN 978-0-12-822320-8.

[10] O. Gil. *Geometría y Algebra lineal Parte 2*. Instituto de Matemática y Estadística Rafael Laguardia, 2005.

[11] A. Goldsmith. *Wireless Communications*. Cambridge University Press, 2005. ISBN 978-0-521-83716-3.

[12] European Telecommunications Standards Institute. 5g; study on channel model for frequencies from 0.5 to 100 ghz. https://portal.etsi.org/webapp/workprogram/Report_WorkItem.asp?WKI_ID=55981, 2017.

[13] Pablo Belzarena. Inteligencia Artificial aplicada a Redes 5G. Agencia Nacional de Investigación e Innovación. https://www.anii.org.uy/proyectos/FMV_1_2019_1_155700/inteligencia-artificial-aplicada-a-redes-5g/.

[14] Gabriela Pereyra. Scheduling in 5g networks : Developing a 5g cell capacity simulator. Master's thesis, Universidad de la República (Uruguay). Facultad de Ingeniería. IIE, sep 2021.

[15] Gabriela Pereyra, Lucas Inglés, Claudina Rattaro, and Pablo Belzarena. An open source multi-slice cell capacity framework. *CLEI Electronic Journal*, 25(2):1–21, may 2022.

[16] Gabriela Pereyra, Claudina Rattaro, and Pablo Belzarena. A 5g multi-slice cell capacity framework. In *ACM SIGCOMM 2021 (online), 23-28 aug*, pages 1–3. ACM, 2021.

[17] J. Salo, L. Vuokko, H.M. El-Sallabi, and P. Vainikainen. Shadow fading revisited. In *2006 IEEE 63rd Vehicular Technology Conference*, volume 6, pages 2843–2847, 2006.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el lunes 3 abril, 2023.
http://iie.fing.edu.uy/