Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

Posicionamiento de puntos de acceso móviles para el servicio de VoIP

Autor: Vicente Jesús Mayor Gallego

Tutor: Antonio José Estepa Alonso

Dep. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Telecomunicación

Posicionamiento de puntos de acceso móviles para el servicio de VoIP

Autor:

Vicente Jesús Mayor Gallego

Tutor:

Antonio José Estepa Alonso

Profesor Titular

Dep. de Ingeniería Telemática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Máster: Posicionamiento de puntos de acceso móviles para el servicio de VoIP

Autor: Vicente Jesús Mayor Gallego
Tutor: Antonio José Estepa Alonso

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mi pareja
A mis padres

Resumen

En este documento se propone el posicionamiento de drones para dar servicio de Voz sobre IP (VoIP) en una región acotada. Para ello, se desarrollan una serie de herramientas que facilitarán el posicionamiento y el estudio de la calidad de servicio ofrecida.

Con el fin de lograr lo anteriormente descrito, se propone la división del proyecto en tres fases. La primera tratará el posicionamiento de los puntos de acceso para cumplir condiciones de cobertura basados en la intensidad de la señal; la segunda tratará de evaluar, mediante simulación, la calidad del servicio que se prestaría según el posicionamiento anterior y; en la tercera y última fase, se propone la validación de un modelo analítico de multiplexación homogénea mediante la herramienta desarrollada en la segunda fase.

En definitiva, el resultado del proyecto es un marco de trabajo que permite generar un escenario, posicionar un conjunto de drones que harán de punto de acceso y, por último, simular el despliegue anterior para evaluar la calidad de servicio.

Los resultados obtenidos indican que la capacidad de las redes de área local inalámbricas se encuentra fuertemente relacionada tanto con la configuración del estándar WiFi a utilizar, como con la densidad de usuarios que atiende cada punto de acceso. Debido a lo anterior, se propone que en investigaciones futuras se incorpore al algoritmo de posicionamiento inicial la estimación de la calidad de servicio.

Abstract

This document proposes the use of drones to provide Voice over IP (VoIP) services in a delimited area. To this end, a series of tools will be developed in order to accomplish the positioning and to study the quality of service provided.

In order to achieve the above, it is proposed to divide the project into three phases. The first one will address the positioning of access points to meet some coverage conditions based on signal strength; the second one will try to evaluate, by simulation, the quality of the service that would be provided according to the previous positioning and; in the third and last phase, it is proposed the validation of an analytical model of homogeneous multiplexing using the tool developed in the second phase.

In short, the result of the project includes a framework that allows generating a scenario, positioning a set of drones that will act as access points and, finally, simulating the previous deployment to evaluate the quality of service.

The results obtained indicate that the capacity of wireless local area networks are strongly related both to the given WiFi standard configuration and to the density of users that each access point serves. In consequence, for future research, the estimation of quality of service should be added into the initial positioning algorithm.

Índice

1 Introducción

Con el fin de acercar el interés del proyecto al lector, se presenta este capítulo de introducción, en el que se abordará tanto la motivación que impulsa al desarrollo del proyecto, como los principales aspectos que determinan la metodología utilizada para alcanzar los objetivos.

1.1 Contexto

Los protocolos de Voz sobre IP (VoIP) permiten cursar llamadas de voz sobre el Protocolo de Internet (IP), aprovechando la infraestructura de la conocida red global. Su principal ventaja es el coste, ya que al aprovechar la conmutación de paquetes sobre la red IP, no se requiere de una infraestructura de transporte dedicada.

Los primeros protocolos de VoIP datan del año 1973 por parte de ARPANET, no obstante, la calidad de las llamadas era pobre debido al reducido ancho de banda del que se disponía [1]. Sin embargo, a partir del año 2000, el ancho de banda disponible ha experimentado un gran crecimiento y ha permitido que las llamadas VoIP sean cada vez más populares [2] como se puede ver reflejado en la Figura 1.1.

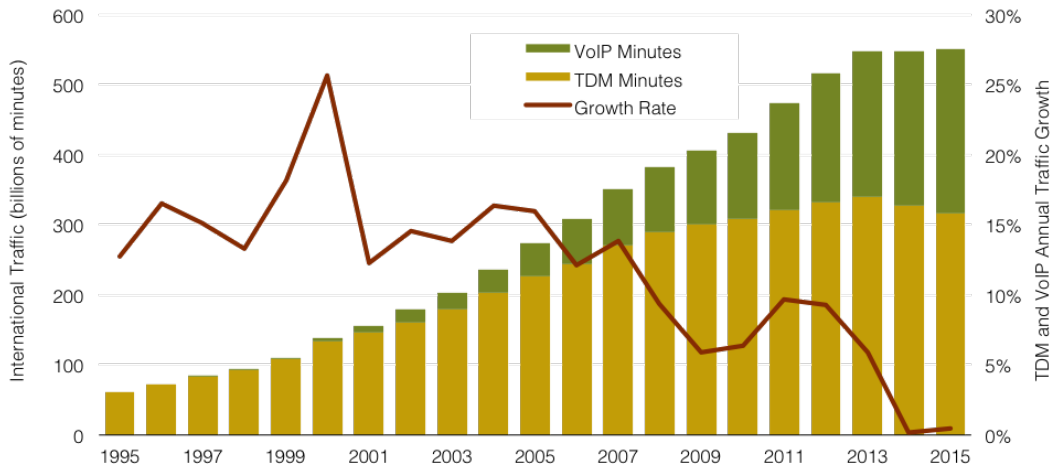


Figura 1.1 Crecimiento del tráfico VoIP.

Por otro lado, en los últimos años el mercado de la telefonía móvil ha experimentado un gran crecimiento con la llegada de los teléfonos inteligentes (conocidos popularmente como *smartphones*). Gracias a estos dispositivos, se pueden realizar llamadas VoIP de manera inalámbrica, accediendo tanto a través de redes móviles (GSM, UMTS, etc.), como de redes inalámbricas de área local (WLAN).

En consecuencia de ambos crecimientos, resulta interesante pensar en el despliegue de redes inalámbricas orientadas a dar un servicio de Voz sobre IP, de modo que los usuarios puedan cursar llamadas de este tipo utilizando sus teléfonos inteligentes.

En este documento se pretende orientar la idea anterior hacia un tipo de escenario particular, como pueden ser, por ejemplo, un concierto o una zona accidentada. Este tipo de eventos presentan una serie de características comunes:

- Hay un recinto de *dimensiones acotadas*.
- Son *temporales*.
- Existe un *número cambiante de usuarios que pueden moverse, entrar y salir* del recinto.

Debido a las características anteriores, en lugar de instalar una infraestructura fija que ofrezca la cobertura del servicio, se propone desplegar puntos de acceso móviles (por ejemplo, mediante drones) que se posicionarán dinámicamente para garantizar el servicio a las personas del evento.

1.2 Problemas

El posicionamiento de los puntos de acceso para satisfacer las condiciones de las llamadas VoIP no es un asunto trivial. Por este motivo se presenta esta sección, con el objetivo de analizar los diferentes problemas que se deberán resolver para poder ofrecer el servicio propuesto. Además, se hará una pequeña revisión del estado del arte con el fin de comprender cómo otros han abordado la resolución tanto de los problemas individuales, como de proyectos similares.

Con el fin de facilitar el análisis, se propone la siguiente división del problema en dos fases:

- *Posicionamiento inicial* de los puntos de acceso.
- *Actualización de la posición* de los puntos de acceso.

Una vez realizada esta aclaración, se procede a estudiar cada uno de los problemas por separado.

1.2.1 Posicionamiento inicial

Como situación de partida, se tiene un recinto de dimensiones acotadas en el que un cierto número de usuarios se encuentra distribuido de manera aleatoria.

Ante esta situación, se pretende desplegar una serie de puntos de acceso que faciliten el servicio de VoIP a un número mínimo de usuarios. Con el fin de garantizar un buen comportamiento del sistema, se impone el cumplimiento de las siguientes restricciones:

- *Minimizar coste*, por ejemplo desplegando el menor número de puntos de acceso posible.
- *Maximizar el número de usuarios que disfrutarán del servicio*, que implica:
  - Que reciban una intensidad de señal suficiente.
  - Que cumplan las condiciones de calidad de servicio.

En la actualidad se han desarrollado numerosos estudios que se centran, en mayor o menor medida, en tratar algunos de los problemas anteriormente mencionados. A continuación se presentan algunas referencias sobre el estado del arte actual.

En cuanto a la intensidad de señal recibida, los artículos [3, 4] proponen nuevos modelos de propagación para WLAN basadas en IEEE 802.11. Estos modelos, parten de los ya existentes para ofrecer una versión simple aunque más precisa en exteriores.

En cuanto al desempeño de los estándares IEEE 802.11, el artículo [5] realiza un estudio tanto teórico, como simulado y experimental de la capacidad que ofrecen las WLAN bajo el estándar IEEE 802.11b. Sin embargo, al tratarse de un estudio de 2009, no estudia la capacidad de estándares más actuales; como IEEE 802.11g, 802.11n u 802.11ac; que permiten mejorar significativamente la capacidad de las WLAN.

Por otro lado, en cuanto a las técnicas de medición de la calidad de servicio (QoS) en comunicaciones VoIP, en [6, 7] se trata el estudio de la misma mediante un método basado en la percepción del usuario, la MOS (*Mean Opinion Score*).

Por último se proponen algunos artículos que tratan temas más tangenciales al proyecto propuesto, aunque también de interés. Por ejemplo, en [8] se trata la eficiencia energética en IEEE 802.11, en [9] se realiza un estudio sobre el efecto de las largas distancias en comunicaciones VoIP y en [10] se lleva a cabo un análisis mediante simulación del efecto del canal inalámbrico sobre la capa física del IEEE 802.11n.

1.2.2 Actualización de la posición

En esta fase se requiere actualizar la posición de los puntos de acceso para ofrecer un servicio continuado a los usuarios aunque éstos se desplacen.

Para lograrlo, se suma la elección de un algoritmo para la actualización de la posición a los problemas desarrollados en la fase anterior, que además debe cumplir las siguientes restricciones:

- *Convergencia rápida* en una solución.
- *Minimizar costes* de desplazamiento.

La reducción de los costes de desplazamiento se centran, sobre todo, en una correcta decisión a la hora de añadir o retirar un punto de acceso, así como elegir correctamente el punto de acceso que debe desplazarse en cada caso.

En cuanto al estado del arte de este tipo de algoritmos, existen algunos artículos que los tratan. Por ejemplo, en [11] se estudian dos métodos de posicionamiento dinámico de drones, que se tratan como estaciones base de una red celular, con el objetivo de maximizar la eficiencia espectral de la señal.

Cabe mencionar que la resolución de este problema queda fuera del alcance del proyecto, dejándolo para futuras investigaciones.

1.2.3 Estado del arte

Aunque en cada una de las fases se han comentado algunas referencias donde se tratan algunos de los problemas individualmente, se va a dedicar este apartado a mencionar algunos artículos cuya idea global sea similar al que se propone aquí.

En primer lugar, la patente [12] presenta una temática muy similar: ofrecer un método de extensión de la cobertura de red. Con este fin, proponen el uso de dispositivos como globos aerostáticos, drones, etc. Sin embargo, la patente presenta un carácter muy general, sin limitarse en ofrecer un servicio en concreto.

En segundo lugar, los artículos [13, 14] tratan el despliegue de redes ad-hoc para establecer una red inalámbrica en entornos accidentados sin necesidad de ninguna infraestructura. Estos documentos se centran en analizar los posibles protocolos de enrutamiento mediante simulación.

Por último, el artículo [15] trata de resolver el mismo problema que los anteriores, pero centrándose en el despliegue de los drones que ofrecerán la cobertura. Por lo tanto, estudia y ofrece un algoritmo para el posicionamiento de los drones que pueden resultar de interés para el desarrollo de este proyecto.

Como se puede observar, aunque con diferentes enfoques, los problemas a resolver son similares en cualquiera de los casos y se trata de un reto que resulta de interés en el campo de la investigación.

1.3 Objetivo

Puesto que abordar todos los problemas anteriormente mencionados conllevaría demasiado tiempo y esto resulta incompatible con la naturaleza del documento (Trabajo Fin de Máster), se ha decidido descartar la segunda fase (actualización de la posición), de modo que el proyecto se centrará en desarrollar herramientas y técnicas que solucionen, total o parcialmente, los problemas del posicionamiento inicial de los puntos de acceso.

En definitiva, el principal objetivo que se persigue es el modelado y simulación del posicionamiento inicial de drones para ofrecer un servicio de VoIP.

En detalle, lo que se propone es la elaboración de un marco de trabajo que permita:

- *Generación aleatoria de un escenario*.
- *Despliegue de drones* (puntos de acceso) para satisfacer unas condiciones de cobertura dadas.
- *Simulación del despliegue* anterior para estudiar condiciones de calidad de servicio.

Puesto que la simulación no es una herramienta adecuada para la aplicación que se propone, se utilizará con el fin de validar un modelo analítico que permita estimar los parámetros de calidad del servicio.

Una vez confirmado dicho modelo, sería posible la integración del mismo en las tareas de despliegue, de modo que en lugar de tener en cuenta únicamente las condiciones de cobertura, se le sumarían las de calidad del servicio. Esta integración se propone como línea futura, quedando fuera del alcance del proyecto.

En los capítulos siguientes se tratan tanto la presentación de la metodología seguida para la resolución de los objetivos propuestos, así como la descripción en detalle de cada una de las fases en las que se divide el proyecto.

2 Metodología

Este capítulo tratará de aclarar la metodología utilizada en la resolución del proyecto. Con el fin de cumplir los objetivos descritos en el capítulo anterior, se propone la división del proyecto en tres fases, que son las enumeradas a continuación:

- Posicionamiento inicial por señal.
- Refinamiento por servicio.
- Confirmación del modelo analítico.

Cada fase depende del resultado de la anterior, de modo que es importante definir correctamente las entradas y salidas de cada fase. En consecuencia de la división propuesta, para cada una de las fases se le reserva una sección independiente en este capítulo.

2.1 Posicionamiento inicial por señal

La primera fase tratará de resolver el posicionamiento inicial de los drones tomando como criterio la intensidad de señal recibida por los usuarios. En definitiva, lo que se propone es la elaboración de una herramienta que permita lo siguiente:

- *Generar un escenario* sobre el que trabajar.
- *Posicionar los drones* de modo que se cumplan unas condiciones de cobertura dadas.

Para desarrollar esta fase se ha utilizado MATLAB [16] como plataforma. Con el fin de comprender mejor el desarrollo de esta fase, se adjunta en la Figura 2.1 el flujo de las tareas principales a realizar.

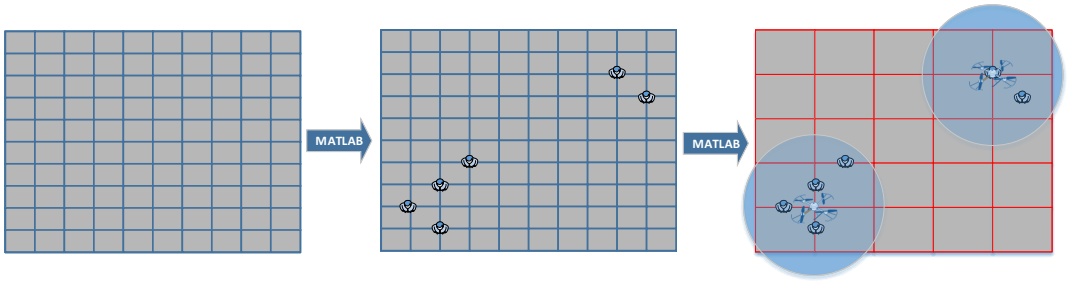


Figura 2.1 Metodología utilizada en el posicionamiento inicial por señal.

Como se puede deducir de la figura anterior, la aplicación partirá de un escenario vacío, sobre el que se generarán usuarios y, posteriormente, se desplegarán drones en función de la cobertura.

2.2 Refinamiento por servicio

Una vez calculado el posicionamiento de los puntos de acceso, se desea evaluar la calidad de servicio que perciben los usuarios mientras cursan llamadas de VoIP. Puesto que la calidad de una llamada telefónica es un concepto subjetivo, típicamente [6] se evalúa con lo que se conoce como MOS (*Mean Opinion Score*).

La MOS es una medida comúnmente utilizada en la ingeniería de telecomunicación para cuantificar la calidad percibida por usuarios ante un estímulo (en el caso actual, un servicio).

La ITU-T en su recomendación P.800.1 [17] define diferentes métodos para estimar la MOS dependiendo del tipo de servicio a evaluar. En el caso de la calidad de una llamada, los principales parámetros que se necesitan para estimar la MOS son los siguientes:

- Caudal (*Throughput*).
- Tasa de pérdidas de paquetes (*Packet Loss*).
- Retardo (*Delay*).
- Variación en el retardo (*Jitter*).

El resultado es un valor entero comprendido entre 1 y 5 que pondera la calidad percibida tal y como se indica en el Tabla 2.1.

Tabla 2.1 Relación entre MOS y calidad según ITU-T.

MOS	Calidad percibida
5	Excelente
4	Buena
3	Regular
2	Mediocre
1	Mala

En cuanto a la herramienta a desarrollar en esta fase, deberá calcular los parámetros anteriores mediante la simulación de la topología suministrada ante un tráfico de VoIP.

Respecto a la plataforma utilizada para el desarrollo de la herramienta, se va a trabajar sobre el simulador NS (*Network Simulator*) en su versión NS-3 [18]. Este simulador ofrece una serie de librerías para C++ y Python que facilitan la implementación y simulación de modelos de red ya existentes. En el Capítulo 4 se hará una descripción con más detalle acerca de la simulación y su implementación.

2.3 Confirmación del modelo analítico

Una vez desarrollada la herramienta de simulación, se propone la confirmación de modelos analíticos ya existentes que estiman la calidad de servicio percibida.

En el caso de este proyecto, se pretende comprobar la validez de un modelo analítico homogéneo sencillo, en el que todos los usuarios gozan de la misma tasa física. Para ello, se deben resolver las siguientes tareas:

1. *Generación de un escenario* en el que todos los usuarios disfruten la peor tasa física para un determinado estándar (misma distancia al punto de acceso).
2. *Simulación* del caso anterior para evaluar la calidad de servicio.
3. *Estimación* de la QoS del caso anterior mediante el modelo analítico homogéneo.
4. *Comparación* entre los resultados simulados y los obtenidos analíticamente.

El objetivo de esta fase es, por tanto, validar modelos analíticos que estimen la calidad percibida por los usuarios para su posterior integración en la fase de despliegue de drones, de modo que dicho despliegue pueda optimizar, en lugar de la intensidad de la señal recibida, la calidad del servicio. Se puede encontrar el desarrollo de esta fase con más detalle en el Capítulo 5.

3 Posicionamiento inicial por señal

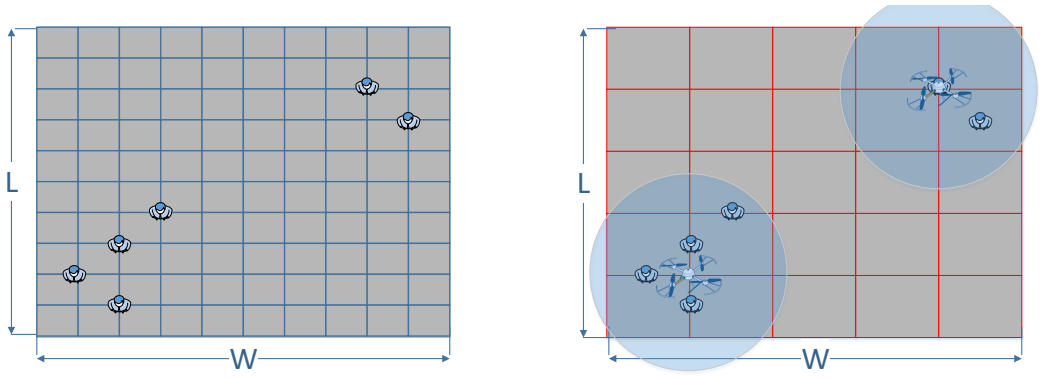
En este capítulo se estudia la primera de las fases, consistente en desarrollar una herramienta que resuelva el posicionamiento inicial de los drones en función de la señal que reciben los usuarios.

Con el fin de abordar ordenadamente el estudio de la fase, se propone la división de la misma en tres secciones: *descripción del problema* (Sección 3.1), donde se tratará de dar una visión detallada del desglose de los problemas; *solución desarrollada* (Sección 3.2), donde se presentará la propuesta de resolución para cada uno de los problemas y; por último, *salidas y resultados* obtenidos (Sección 3.3) que, como su propio nombre indica, tratará de explicar el uso e interpretación de los resultados generados.

3.1 Descripción del problema

Con el fin de describir los problemas implicados en la fase actual, se propone a continuación el siguiente desglose de tareas a tratar:

1. *Lectura* del fichero de configuración de entrada y almacenamiento de los parámetros.
2. *Generación* de un recinto vacío de las dimensiones leídas.
3. *Adición* aleatoria al recinto del número *de usuarios* especificado (Figura 3.1a).
4. *Evaluación* de la SNR (*Signal-to-Noise Ratio*) percibida por los usuarios para las diferentes combinaciones posibles de despliegue de drones.
5. *Selección* del mejor posicionamiento que cumpla las restricciones de SNR propuestas (Figura 3.1b).
6. *Cálculo de otros parámetros* orientados a la siguiente fase a partir de la SNR.
7. *Exportación de los resultados* en un fichero de salida fácilmente interpretable.



(a) Tablero tras reparto de usuarios. **(b)** Tablero tras reparto de drones.

Figura 3.1 Resumen del proceso de despliegue.

En el *fichero de entrada* se deben contemplar todos los parámetros necesarios para llevar a cabo la simulación. En la Tabla 3.1 se recogen los parámetros mínimos que se deben definir. Su significado y necesidad se explicarán con mayor detalle a continuación.

Tabla 3.1 Parámetros definidos en el fichero de entrada.

Clasificación	Nombre
Parámetros geométricos	Dimensiones del tablero
	Matriz de posiciones de usuarios
	Matriz de posiciones de drones
Parámetros de simulación	Número de usuarios a distribuir
	Porcentaje de usuarios cubiertos
	SNR mínima para considerar cubierto
	Función objetivo a maximizar
	Número máximo de drones
Parámetros de radiopropagación	Exponente de propagación
	Función de pérdidas
	Directividad de las antenas
	Potencia de transmisión
	Frecuencia de transmisión
	Ruido
Parámetros del estándar	Estándar
	Ancho de banda del canal
	Intervalo de guarda

En cuanto al *fichero de salida*, se pretende volcar los resultados de esta fase para poder ser utilizada posteriormente como entrada de la siguiente. Por este motivo, el fichero deberá presentar una estructura fácilmente interpretable como puede ser XML (*eXtensible Markup Language*).

De este modo, se pretende que el fichero de salida presente una estructura similar al del siguiente ejemplo:

Código 3.1 Fichero de salida de ejemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<escenario ChannelWidth="20" ExpPropagation="3" Frequency="2.4" L="200"
    NumActiveUsers="2" NumUsers="2" Ptx="20" ShortGuardEnabled="true" W="400"
    WifiPhyStandard="WIFI_PHY_STANDARD_80211n_2_4GHz">
<dron id="1" x="200" y="100" z="110">
<usuario Active="true" Gain="-0.70928" PhyRate="14.4" WifiPhy="HtMcs1" id="1" x
    ="12" y="39" z="1.5"/>
<usuario Active="true" Gain="-0.47636" PhyRate="14.4" WifiPhy="HtMcs1" id="2" x
    ="12" y="141" z="1.5"/>
</dron>
</escenario>
```

En definitiva, como mínimo será necesario recoger los datos que se indican en el Tabla 3.2.

En cuanto a la resolución técnica, el desglose de tareas anterior puede agruparse en los procesos lógicos que se proponen a continuación:

- Generación del escenario.
- Búsqueda de soluciones.
- Comprobación de la cobertura.
- Comprobación de la tasa.

Tabla 3.2 Parámetros definidos en el fichero de salida.

Entidad a la que afecta	Parámetro
Todos	Estándar
	Ancho de banda de canal
	Exponente de propagación
	Frecuencia
	Dimensiones del recinto
	Potencia de transmisión
	Parámetros del estándar
Drones	Identificador
	Posición
	Usuarios que abastece
Usuarios	Identificador
	Posición
	Actividad
	Tasa física
	MCS (Modulation and Coding Scheme)

Conforme a la división propuesta, se presenta a continuación la descripción de cada uno de los problemas particulares.

3.1.1 Generación del escenario

Puesto que no se parte de un escenario real ya existente, el primer problema que se debe abordar es la generación del escenario, es decir, la creación del tablero y el reparto inicial de usuarios.

Con el fin de resolver este problema, se requiere aportar, como mínimo, la siguiente información de entrada:

- *Dimensiones* del tablero ($W \times L$).
- *Número de usuarios*.
- *Posiciones válidas para los usuarios*.
- *Altura de los usuarios*.

La información anterior es requisito para la resolución de este problema, de modo que obligatoriamente se deben contemplar en el fichero de configuración de entrada.

En cuanto a los retos que supone este problema, el reparto de usuarios debe ser aleatorio, de modo que sería interesante ofrecer la posibilidad de personalizar la función de probabilidad que regula reparto para permitir seguir cualquier distribución probabilista.

La salida de esta tarea será un tablero de dimensiones dadas y un conjunto de usuarios repartidos sobre él.

3.1.2 Búsqueda de soluciones

Una vez generado el escenario a resolver, el siguiente problema al que se debe hacer frente es la búsqueda de posibles posicionamientos de drones. Un posicionamiento es posible cuando los drones se sitúan en posiciones válidas, esto es que respetan la matriz de posiciones impuesta en la simulación.

Para abordar este problema se requiere, como mínimo, la siguiente información de entrada:

- *Definición del escenario* a resolver (incluye dimensiones y posición de usuarios).
- *Posiciones válidas para los drones* (tridimensional).
- *Número máximo de drones* a desplegar.
- *Función objetivo* a maximizar.

A partir de los parámetros anteriores, se debe diseñar un algoritmo que pruebe diferentes posibilidades de despliegue y elija la que mejor satisfaga la función objetivo indicada.

Respecto a los retos que supone la resolución de este apartado, es importante que las combinaciones de posicionamiento a probar no se repitan y que, además, converjan siempre en una solución.

La salida obtenida tras esta sección será la generación iterativa de un vector de posiciones que representará la posición de cada dron en el escenario (tridimensional).

3.1.3 Comprobación de la cobertura

Por cada posible solución obtenida en el apartado anterior se deberá evaluar si cumple los requisitos de cobertura impuestos. Para la evaluación de un posicionamiento, por tanto, se requieren los siguientes datos de entrada:

- *Definición del escenario* a estudiar (incluye dimensiones, posición de usuarios y posición de drones).
- *Potencia de transmisión*.
- *Función de pérdidas* según un modelo de propagación dado.
- *Ruido* a considerar.
- *Porcentaje de usuarios* que deben cumplir requisitos.
- Requisito de *SNR mínima* a cumplir.
- *Frecuencia* de transmisión.
- *Directividad* de las antenas.

En definitiva, lo que se requiere en este apartado es el diseño de un algoritmo que establezca el emparejamiento entre los usuarios y los drones, calcule la SNR apreciada por cada usuario y que, por último, evalúe si éstos cumplen los requisitos de cobertura.

La salida de esta tarea será: la SNR de cada usuario, la relación de emparejamiento de cada usuario y dron, y una bandera que indicará si cumple o no los requisitos de cobertura.

3.1.4 Comprobación de la tasa

Por último, una vez elegida la solución que optimiza la función objetivo de entre todas las posibles, se requiere la estimación de la tasa física a la que va a poder transmitir cada usuario. Para lograr esta estimación, se requieren los siguientes datos:

- *SNR* que percibe cada usuario.
- *Estándar*.
- *Frecuencia*.
- Número de *flujos espaciales*.
- *Ancho de banda* del canal.
- *Intervalo de guarda*.

A partir de lo anterior es posible, mediante la búsqueda en una serie de tablas, determinar la tasa física que corresponde una determinada SNR en un estándar con los parámetros dados. La salida de esta subfase es, por tanto, la tasa física y la MCS (*Modulation and Coding Scheme*) que corresponde a cada usuario.

3.2 Solución desarrollada

A partir del desglose de problemas a tratar descrito en la sección anterior, se han desarrollado un conjunto de *scripts* y funciones en MATLAB que automatizan la simulación. Los diferentes ficheros que forman parte de la herramienta son los indicados en la Tabla 3.3.

Como se puede apreciar, la separación de problemas que se realizó en la Sección 3.1 se ha mantenido a la hora de diseñar su solución, de modo que cada problema particular se trata en una función o *script* diferente. Esta decisión ha desembocado en una solución modular fácil de modificar o ampliar siempre que se respete el formato de entrada y salida de cada proceso.

En cuanto a su descripción, se propone seguir la siguiente metodología u orden:

Tabla 3.3 Ficheros que componen la solución en MATLAB.

Nombre del fichero	Tipo	Cometido
configura.m	Script	Permite configurar los parámetros necesarios para la simulación.
simula.m	Script	Lanza la simulación en base a los parámetros del fichero configura.m
Escenario.m	Clase	Define una entidad llamada Escenario, que facilita la generación del escenario.
busca_solucion.m	Función	A partir de un escenario generado, busca posibles soluciones y elije la mejor siguiendo una F.O.
comprueba_cobertura.m	Función	A partir de un escenario resuelto, comprueba la validez de la solución según la SNR.
comprueba_tasa.m	Función	A partir de una SNR, devuelve la tasa física que le corresponde.
escribe_salida.m	Función	Exporta los resultados de la simulación a un fichero XML.
Constante.m	Clase	Define un conjunto de constantes que se utilizan a lo largo de la simulación.
*dibuja_****.m*	Función	Conjunto de funciones que automatizan la representación de los resultados en gráficos.

1. Visión general de la resolución del problema.
2. Datos a definir en el fichero de entrada y formato en que se describen.
3. Resolución del desglose de problemas anteriormente descrito.

Siguiendo el guión se comienza aportando una visión general de la resolución problema.

3.2.1 Visión general

Con el fin de facilitar la comprensión de la interacción entre los componentes mostrados en la Tabla 3.3, se presenta, en la Figura 3.2 (página 12), el diagrama de flujo completo de la solución propuesta.

El diagrama anterior presenta cinco divisiones verticales de modo que cada sección determina el fichero desde el cual se realiza el proceso o decisión. A partir de esta división es sencillo comprender la sucesión de eventos que se llevan a cabo y en qué entidad del programa se resuelven.

3.2.2 Entrada

En cuanto a los parámetros de entrada, estos se encuentran definidos en el fichero *configura.m*. Este fichero con extensión MATLAB sigue la sintaxis de la plataforma, de modo que es posible definir tanto variables como funciones que regularán el comportamiento del programa.

Las posibilidades que ofrece este fichero de configuración se recogen en la Tabla 3.4. Además, se proporcionan algunos valores de ejemplo.

Como se puede observar en la tabla anterior, se pueden definir tanto variables como funciones, el formato a seguir es sencillo:

Para las *variables*, MATLAB propone la sintaxis `clave=valor`, de modo que si por ejemplo se quisiera fijar el número máximo de drones a 5, lo único que se debería escribir en el fichero *configura.m* sería `max_drones=5`.

En cuanto a la *definición de funciones*, se realizan con lo que en MATLAB se conoce como funciones anónimas [19]. Gracias a este tipo de declaración se pueden declarar funciones flexibles en una línea. Por ejemplo, si se quiere declarar la directividad en función del ángulo, basta con ejecutar `D_dB = @(theta) 10*log10(10^(D_max_dB/10)*cos(deg2rad(theta)).^2)`. El uso de este tipo de declaración supone

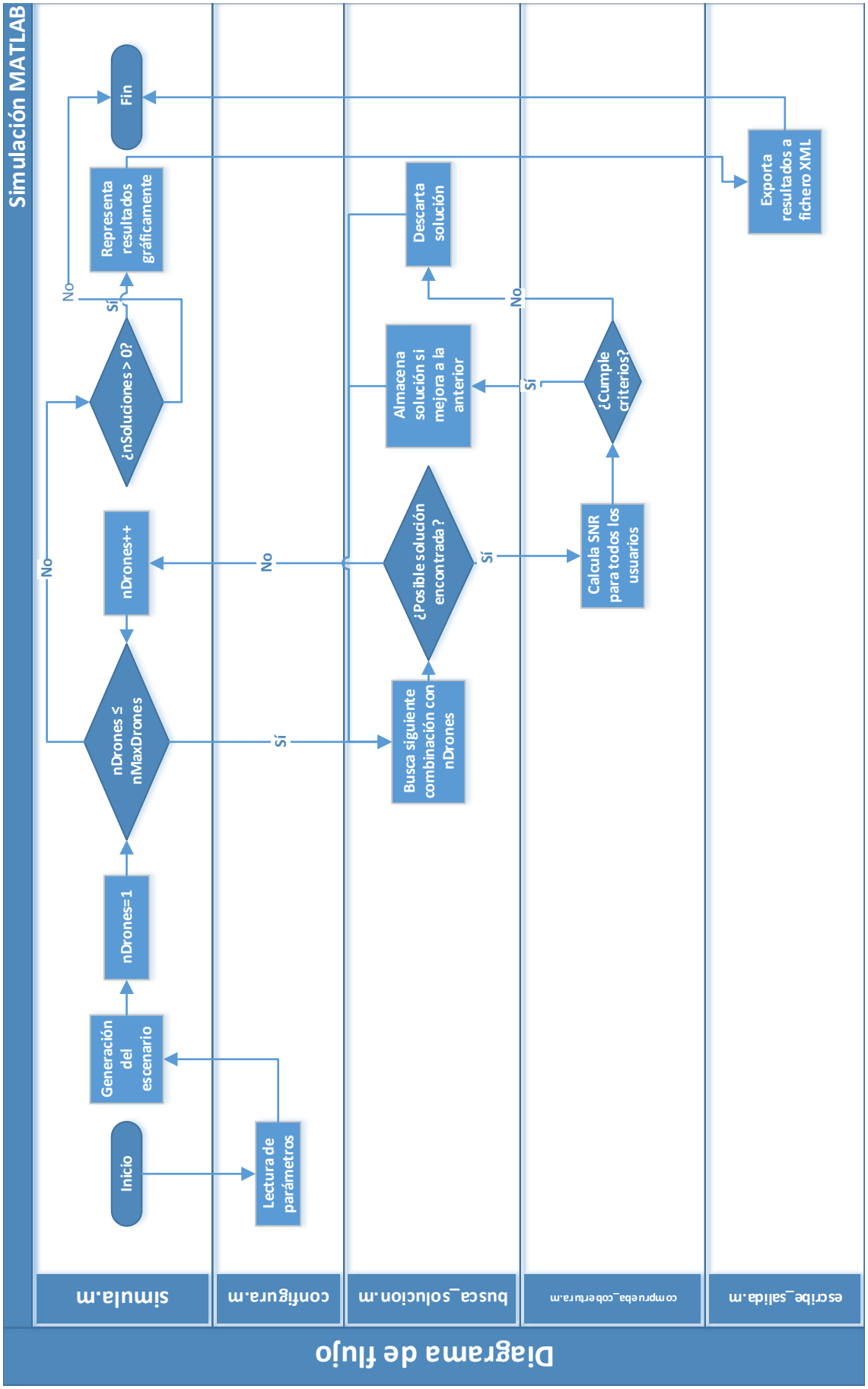


Figura 3.2 Diagrama de flujo de la solución en MATLAB.

Tabla 3.4 Parámetros de entrada de la solución en MATLAB.

Nombre	Unidad	Ejemplo de valor	Descripción
L	m	200	Largo del tablero
W	m	200	Ancho del tablero
paso_real	m	0,5	Distancia en X-Y mínima entre usuarios
xy_step	m	10	Distancia en X-Y mínima entre drones
z_step	m	10	Distancia en Z mínima entre drones
min_altura	m	10	Altura mínima de los drones
max_altura	m	150	Altura máxima de los drones
num_personas	-	100	Número de personas a repartir
num_personas_activas	-	40	Número de personas a marcar como activas
f_FO	-	10*minimo_rx+media_rx	Función objetivo a maximizar
porcentaje_cubierto	%	90	Porcentaje de usuarios a cubrir
max_drones	-	5	Número máximo de drones a desplegar
hr	m	1,5	Altura de los usuarios
D_max_dB	dBi	4,5	Máxima ganancia directiva
D_dB	-	10*log10(10^(D_max_dB/10)* cos(deg2rad(theta)).^2)	Función que calcula la ganancia directiva en función del ángulo
exponente_propagacion	-	3	Exponente de propagación
p_loss	-	-147.55 - g + 10*exponente_propagacion *log10(d) + 20*log10(f)	Función de pérdidas en la propagación
p_tx	dBm	20	Potencia de transmisión de las antenas
margen	dB	3	Márgen a considerar a la hora de elegir tasa física
snr_min	dBm	10	SNR mínima para considerar cubierto
modo	-	Constante.MODO_24_GHZ	Modo de transmisión (frecuencia)
intervalo_guarda	-	Constante.IG_CORTO	Intervalo de guarda
estandar	-	Constante.WIFI_n	Estándar IEEE 802.11
CBW	-	Constante.CBW_20_MHZ	Ancho de banda del canal
KTBF	dBm	-174+10*log10(CBW*10^6)	Ruido a considerar

una ventaja (flexibilidad), pero también afecta al rendimiento ya que la llamada a funciones supone ciclos de procesamiento extra.

Por último, existen otros parámetros que no toman su valor directamente de una variable, sino que se hace por medio de *constantes* definidas en la clase `Constante` (fichero *Constante.m*). Los parámetros que se declaran con esta nomenclatura son aquellos con un conjunto acotado de posibilidades, como puede ser el estándar WiFi a utilizar. Por ejemplo, si se quiere fijar el estándar al IEEE 802.11b, basta con escribir `estandar = Constante.WIFI_b`.

Aunque no es necesario, tras cada línea se recomienda finalizar con la orden punto y coma (`;`). La diferencia es que si no se añade, cuando la línea es ejecutada se imprime por pantalla el resultado de la asignación, de modo que es interesante añadirla siempre que se desee suprimir las salidas.

Como se puede observar, la simulación es totalmente parametrizable y según el diseño que se propone es muy sencillo añadir o eliminar tanto nuevos parámetros, como nuevas constantes.

3.2.3 Soluciones particulares

Una vez presentado tanto el comportamiento general del programa, como la definición del fichero de entrada; se va a proceder con la descripción detallada del desglose de tareas que se llevan a cabo durante la simulación.

Para ello, se va a respetar la división propuesta en la Sección 3.1. Es decir, se comenzará con la generación del escenario, seguida por la búsqueda de soluciones, la comprobación de la cobertura y, por último, la comprobación de la tasa física.

Además, puesto que para cada problema se resuelve en un fichero diferente, se relacionará cada tarea con la entidad que trata su solución.

Generación del escenario

La generación del escenario se realiza en el *script simula.m* con la ayuda de la clase `Escenario` declarada en *Escenario.m*. La clase `Escenario` nos permite crear un editor que facilita la creación de un escenario y la ubicación de los usuarios sobre él. En Código 3.2 se puede ver un ejemplo de generación con la ayuda del editor.

Código 3.2 Ejemplo de generación del escenario.

```
% Iniciamos el editor_escenario
editor_escenario = Escenario(W, L, distancia_entre_personas);

% Creamos la función masa de probabilidad de una distribución uniforme 2D
fu = ones([(L/distancia_entre_personas)+1 (W/distancia_entre_personas)+1]);
fu = fu/(((L/distancia_entre_personas)+1)*((W/distancia_entre_personas)+1));

% Añadimos los puntos al editor_escenario siguiendo la distribución generada
editor_escenario.addPoints(fu, num_personas);

% Obtenemos la matriz que utilizaremos para simular
escenario_matriz = editor_escenario.getMatrix;
```

En el caso anterior, los usuarios se han repartido siguiendo una distribución uniforme, aunque se pueden utilizar otras o incluso una combinación entre diferentes modelos (llamando varias veces al método `addPoints`). Conceptualmente, lo que ha ocurrido en la ejecución del código, es lo siguiente:

1. Definición de la función masa de probabilidad a seguir en el reparto.
2. Adición de los puntos al escenario siguiendo la distribución dada.
3. Obtención de la matriz que representa al escenario.

Siguiendo el esquema anterior, en primer lugar se genera la función masa de probabilidad. Para ello, lo primero que ocurre es la construcción de una matriz del tipo M_{ONES}.

$$M_{ONES} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}_{(W/d+1)\times(L/d+1)}$$

En la matriz anterior, donde W es el ancho del tablero, L su largo y d la distancia mínima entre usuarios; cada uno de los elementos supone una posible posición para un usuario.

Lo que a continuación ocurre es la conversión de la matriz anterior en una función de probabilidad correcta que sigue una distribución uniforme. Para ello se divide la matriz anterior entre el número total de elementos de la matriz (D):

$$M_{FU} = M_{ONES}/D = \begin{bmatrix} 1/D & 1/D & 1/D & \dots & 1/D \\ 1/D & 1/D & 1/D & \dots & 1/D \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1/D & 1/D & 1/D & \dots & 1/D \end{bmatrix}_{(W/d+1)\times(L/d+1)}$$

Por ejemplo, en el caso de $W = L = 3$ metros y $d = 1$ metro, la matriz resultante sería:

$$M_{FU} = \begin{bmatrix} 1/16 & 1/16 & 1/16 & 1/16 \\ 1/16 & 1/16 & 1/16 & 1/16 \\ 1/16 & 1/16 & 1/16 & 1/16 \\ 1/16 & 1/16 & 1/16 & 1/16 \end{bmatrix}$$

La matriz anterior se puede interpretar del siguiente modo: la probabilidad de que se genere un usuario en la posición (x,y) es el valor que encontramos en la matriz en la fila y y la columna x.

A partir de una matriz de este tipo, la clase `Escenario` permite añadir usuarios aleatoriamente siguiendo la distribución descrita. Una vez generados todos los usuarios, basta con llamar a los métodos `getMatrix` o `getCoordinates` dependiendo del formato con el que se quiera leer.

En la mayor parte de la simulación, se utilizará el formato *matriz* (obtenido mediante `getMatrix`) pero, sin embargo, en algunas tareas secundarias como puede ser la representación de gráficos, se hace uso del formato *coordenadas* (obtenible mediante `getCoordinates`).

El formato *matriz* mantiene la filosofía anterior, se dispone de una matriz en la que se reserva un elemento para cada posición ocupable por un usuario. El valor de cada elemento puede ser 0, si no hay ningún usuario emplazado; o 1, si existe un usuario ocupando la posición.

Con el fin de ayudar a comprender cómo se deben interpretar los resultados, a continuación se presenta un ejemplo de generación de escenario (Figura 3.3). En este caso, sobre los datos del ejemplo anterior se desean generar 10 usuarios.

En la figura propuesta, se puede apreciar una representación bidimensional (aunque la componente z esté definida, es constante para los usuarios). Puesto que se ha decidido que las personas no puedan estar a menos de un metro, cada eje presenta 4 posiciones válidas.

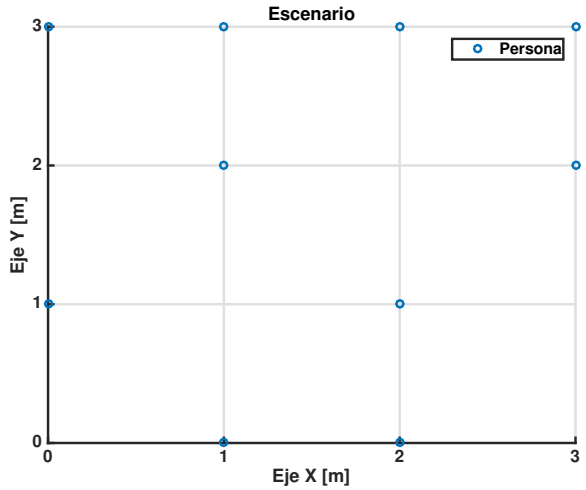


Figura 3.3 Ejemplo de generación de 10 usuarios sobre un tablero de 3×3 metros.

En ejemplos más fieles a la realidad la distancia mínima no debería ser tan grande. Sin embargo, por simplicidad, se ha decidido utilizar este valor en el ejemplo. Es importante comprender que la distancia entre

usuarios no es la distancia entre drones, es decir, la matriz de posiciones de los drones será otra totalmente independiente.

Para la gráfica anterior, su representación matricial sería la siguiente:

$$M_{USUARIOS} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Si expresamos los puntos de la gráfica como (x,y) y los elementos de la matriz como (*fila*,*columna*), se puede observar que la posición (i,j) en la gráfica corresponde con la posición (j,i) en la matriz. Es importante comprender y recordar estas diferencias ya que a lo largo de la ejecución del programa se manejarán estas nomenclaturas y pueden llevar a confusión.

La representación matricial del tablero no se utiliza únicamente para el posicionamiento de los usuarios sino que se reutilizará más adelante para asignar resultados a cada posición, es decir, cuando se obtengan los resultados de la SNR más adelante, se mantendrá el mismo formato que la matriz $M_{USUARIOS}$ pero sustituyendo el valor de cada elemento por la SNR que se percibe en dicho punto.

El uso de este formato tiene como objetivo facilitar las operaciones matriciales frente a la evaluación elemento a elemento mediante bucles. Esto es motivado por la eficiencia que presenta MATLAB en este tipo de operaciones.

El algoritmo utilizado para generar aleatoriamente los usuarios a partir de la función masa de probabilidad es muy sencilla y por ello no se va a entrar en detalles acerca de la implementación. No obstante, se puede consultar el código de la clase `Escenario` comentado en el Apéndice A.

Búsqueda de soluciones

Siguiendo el diagrama de flujo completo de la Figura 3.2, una vez generado el escenario lo siguiente a realizar es la búsqueda de posicionamientos válidos. En otras palabras, se requiere un algoritmo que obtenga posibles combinaciones de posicionamiento para un número de drones dado. De esto se encarga la función `busca_solucion` desarrollada en el fichero *busca_solucion.m*.

El ejercicio a resolver no es más que obtener todas las posibles combinaciones de una combinatoria sin repetición de tantos elementos como drones haya. Aunque cada posición viene dada por un vector (x,y,z), ésta se puede representar con un número entero si se toma como referencia el índice lineal de la matriz, tal como se puede ver en la Figura 3.4.

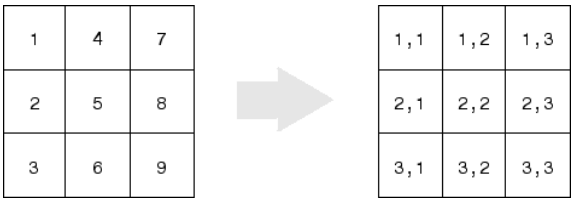


Figura 3.4 Índices escalares y vectoriales.

De este modo, para cada dron se obtendrá un valor entero entre 1 y el número máximo de posiciones. En el ejemplo de la figura anterior, la posición lineal 9 corresponde con el elemento de la fila 3 y columna 3. Esto se puede extrapolar a matrices de mayor orden (en nuestro caso, tridimensionales).

En la solución desarrollada, las combinaciones se obtienen iterativamente a través de la función recursiva `nItera` mostrada en el Código 3.3. Aunque MATLAB ofrece funciones que realizan la enumeración de las combinaciones como puede ser `combnk` [20], debido a los problemas de escalabilidad que se mencionan más adelante, es mejor ir calculándolas y evaluándolas una a una.

Código 3.3 Función recursiva para probar todas las combinaciones.

```
function nItera(nDrones, nPosiciones, pos_ini, dron)
  for pos_act=pos_ini:nPosiciones
    posicionamiento(dron) = pos_act;
    if dron==nDrones
      calcula(posicionamiento, escenario_matriz);
    else
      nItera(nDrones, nPosiciones, pos_act + 1, dron + 1);
    end
  end
end
```

Por ejemplo, si se ejecutase `nItera(3,4,1,1)` se evaluarían las posiciones (en formato escalar) en el siguiente orden:

$$(DRON_1,DRON_2,DRON_3) : (1,2,3) \rightarrow (1,2,4) \rightarrow (1,3,4) \rightarrow (2,3,4)$$

Una vez obtenida la posición en formato escalar, es posible convertirla a una sintaxis vectorial (x,y,z) mediante el comando `ind2sub` [21].

En cuanto a la eficiencia y escalabilidad, el algoritmo, como se puede observar es simple aunque poco eficiente ya que el número total de combinaciones sigue la siguiente expresión:

$$N_{COMBINACIONES} = \binom{N_{POSICIONES}}{N_{DRONES}}$$

A modo de ejemplo, si se quiere desplegar 3 drones sobre un tablero 100×100 con tres capas de altura y distancia mínima entre drones de 10 metros (en cualquier eje), el número de combinaciones posibles asciende a:

$$N_{COMBINACIONES} = \binom{((100/10)+1)\cdot((100/10)+1)\cdot 3}{3} = \binom{363}{3} = 7906261$$

Tal y como se indicaba en el diagrama de flujo global de la Figura 3.2, desde el *script* principal (*simula.m*) se llama a la función de búsqueda incrementando linealmente el número de drones a probar hasta llegar al máximo. Si esto se tiene en cuenta, el número de iteraciones necesarias para resolver un escenario es el siguiente:

$$N_{ITERACIONES} = \sum_{i=1}^{N} \binom{N_{POSICIONES}}{i}$$

Donde N es el menor número de drones para el que se encuentra la primera solución válida. En la Figura 3.5 se muestra gráficamente la tendencia de crecimiento del número de combinaciones conforme aumenta el número máximo de drones.

Como se puede observar, el algoritmo propuesto aunque es sencillo y siempre converge en solución, no es escalable. Por ello, se propone como línea futura la sustitución del mismo por otro como, por ejemplo, un algoritmo genético [22].

Cada posible posición se evaluará mediante la función `comprueba_cobertura` que se explicará más adelante. De todas las posiciones que cumplan las condiciones de cobertura se almacenará la siguiente información:

- *SNR media* recibida.
- *SNR mínima* que se recibe.

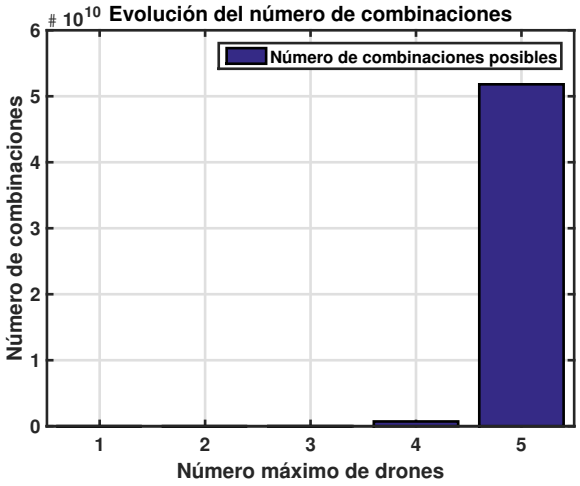


Figura 3.5 Evolución del número de combinaciones según el número de drones.

A partir de estos datos se elegirá la solución que maximice la función objetivo declarada en el fichero de configuración. Por defecto se utiliza la siguiente:

$$FO = 10 \cdot SNR_{min} + SNR_{media}$$

Es decir, se prioriza maximizar la SNR mínima recibida por algún usuario frente a maximizar la SNR media que se recibe entre todos. No obstante, esta función sólo se evaluará para aquellos escenarios que satisfagan las condiciones de cobertura.

Comprobación de la cobertura

Cada posible posicionamiento obtenido en `busca_solucion` debe ser evaluado en la función `comprueba_cobertura` definida en el fichero *comprueba_cobertura.m*.

La evaluación consiste en comprobar si la SNR que recibe cada usuario permite cumplir los requisitos mínimos de cobertura impuestos en el fichero de configuración. Además, se debe emparejar cada usuario con un dron. El criterio utilizado para tomar esta decisión es enlazar cada usuario al dron que le proporciona mayor SNR, de manera que en términos telemáticos la pasarela de un usuario será el dron que se le ha asignado.

El cálculo de la SNR puede ser abordado del siguiente modo:

$$SNR\ [dB] = P_{TX}\ [dB] - P_{LOSS}\ [dB] - N\ [dB]$$

Donde P_{TX} es la potencia de transmisión, P_{LOSS} la función de pérdidas y N el ruido que afecta a la señal. Estos valores son leídos del fichero de configuración y todos son parametrizables.

Por defecto, el modelo de propagación utilizado es el del espacio libre que se presenta a continuación con unidades del Sistema Internacional:

$$P_{LOSS} = -147.55 - G + 10 \cdot exponente \cdot log_{10}(d) + 20 \cdot log_{10}(f)$$

En cuanto a la ganancia, se ha considerado la expresión que sigue a continuación, que representa la directividad mediante dipolo corto [23] que, por supuesto, es parametrizable:

$$G = 10 \cdot log_{10}(D_{MAX} \cdot cos(\theta)^2)$$

Donde θ es el ángulo de desviación (en radianes) en cualquier dirección desde la línea de máxima ganancia y D_{MAX} la máxima ganancia (en unidades naturales) en dicha dirección.A partir de las ecuaciones anteriores se puede comprobar si los usuarios cumplen los requisitos de señal impuestos. En la Figura 3.6 se presenta el algoritmo que se sigue en la función `comprueba_cobertura` para abordar las tareas de emparejamiento y cálculo propuestas.

Una vez ejecutado el algoritmo anterior se debe comprobar si el posicionamiento cumple las condiciones de SNR, que son garantizar un valor mínimo de SNR en un porcentaje mínimo de usuarios. Para ello, en base a la representación matricial del escenario, basta con ejecutar el Código 3.4.

Código 3.4 Función recursiva para probar todas las combinaciones.

```
if (length(find(SNR(escenario==1)<snr_min)))>((1-porcentaje_cubierto/100)*
    num_personas)
  valido = 0;
else
  valido = 1;
  % Hago negativos los emparejamientos que no cumplen especificación de SNR (no
       es necesario cuando la solución no es válida)
  dron_emparejado(SNR<snr_min)=dron_emparejado(SNR<snr_min)*-1;
end
```

Los resultados que son de interés para la resolución de la fase y por tanto, deben ser exportados, son los siguientes:

- *SNR recibida* por cada usuario.
- *Emparejamiento* entre usuarios y drones
- *Ganancia* directiva de la que disfruta cada usuario

Todos ellos se representan del mismo modo: para cada una de ellas se genera una matriz en la que cada elemento se relaciona con un punto del tablero y, dependiendo de qué matriz sea, el valor tiene un significado u otro, tal como sigue a continuación:

- *Matriz de SNR*: Contiene el valor de la SNR en cada punto del escenario.
- *Matriz de emparejamiento*: Contiene el identificador del dron que abastece cada punto del escenario. Si no se satisface la condición de cobertura en dicho punto, el valor es negado.
- *Matriz de ganancias*: Contiene el valor de la ganancia directiva que afecta a cada punto del escenario en decibelios.

Comprobación de la tasa

La comprobación de la tasa física a la que va a sincronizar cada usuario se resuelve en la función `comprueba_tasa`.

En el fichero *Constante.m* se han declarado una serie de tablas que relacionan la MCS (*Modulation and Coding Scheme*) con la SNR y los parámetros relacionados con el estándar utilizado (versión, ancho de banda e intervalo de guarda). A partir de la MCS, la obtención de la tasa es inmediata.

Puesto que la resolución de este problema es trivial, en el Apéndice A se pueden consultar las tablas utilizadas para cada uno de los estándares. A partir de ellas, la relación entre los parámetros es inmediata.

3.3 Salidas y resultados

Para finalizar con este capítulo y con el fin de demostrar que el comportamiento de la herramienta es el esperado, se propone la generación y simulación del conjunto de parámetros recogidos en la Tabla 3.5.

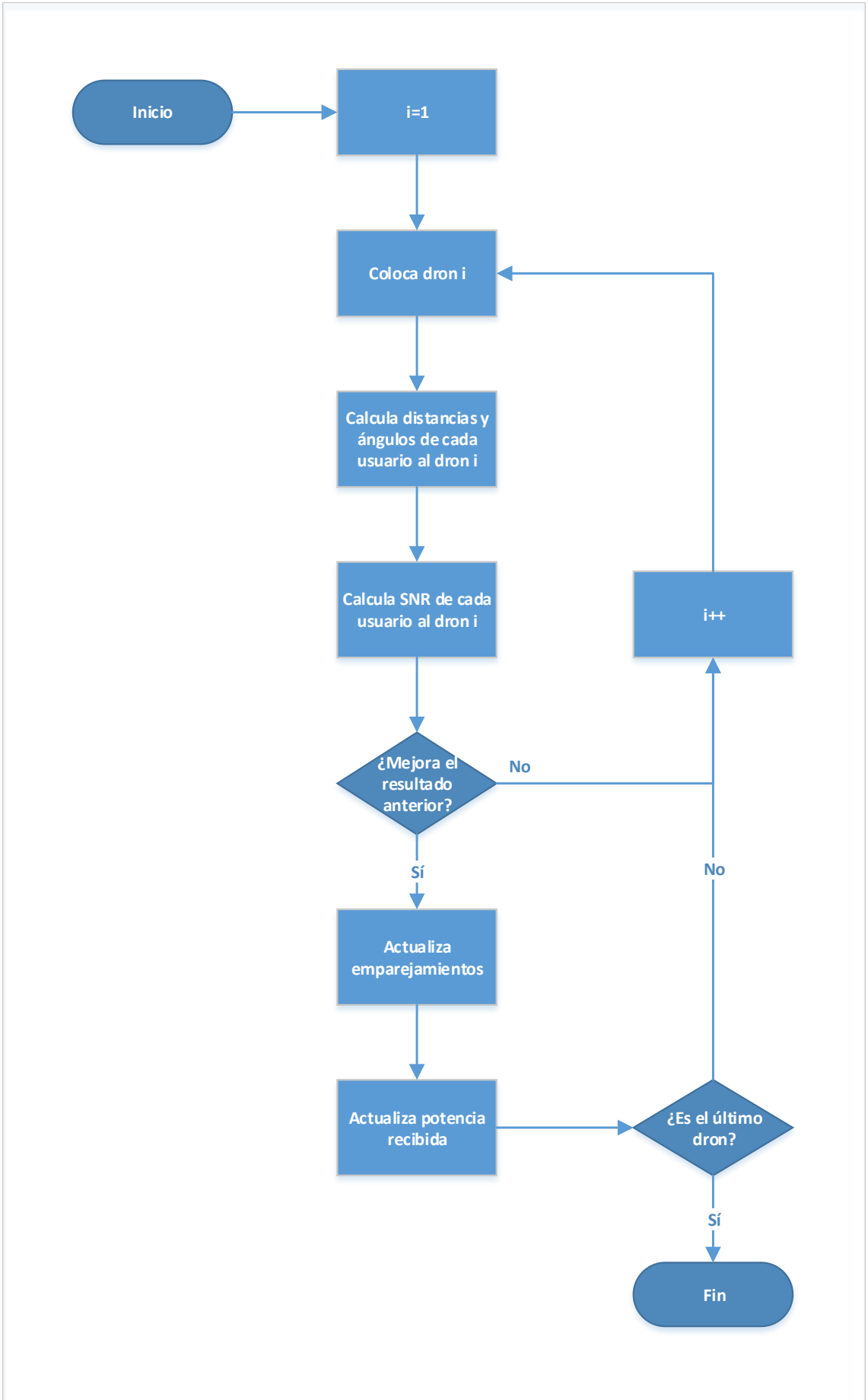


Figura 3.6 Diagrama de flujo del cálculo de la SNR y emparejamiento entre dron y usuario.

Tabla 3.5 Datos de entrada de la simulación propuesta.

Nombre	Ejemplo de valor
L	200
W	400
paso_real	0,5
xy_step	50
z_step	50
min_altura	10
max_altura	100
num_personas	100
num_personas_activas	100
f_FO	10*minimo_rx+media_rx
porcentaje_cubierto	95
max_drones	5
hr	1,5
D_max_dB	4,5
D_dB	10*log10(10^(D_max_dB/10)*cos(deg2rad(theta)).^2)
exponente_propagacion	3
p_loss	-147.55 - g + 10*exponente_propagacion*log10(d) + 20*log10(f)
p_tx	20
margen	3
snr_min	10
modo	Constante.MODO_24_GHZ
intervalo_guarda	Constante.IG_CORTO
estandar	Constante.WIFI_n
CBW	Constante.CBW_20_MHZ
KTBF	-174+10*log10(CBW*10^6)

3.3.1 Ejemplo de ejecución

Si ejecutamos el programa desarrollado con el *script simula.m*, se obtiene la salida (parcial) por línea de comandos adjuntos en el Código 3.5.

Código 3.5 Salida por línea de comandos de la solución en MATLAB.

```
>> simula

Distribuyendo los usuarios en el tablero

Simulando con 1 drones
  Existen 90 posibilidades para 1 drones...: 90 posibilidades probadas
  No se han encontrado soluciones

Simulando con 2 drones
  Existen 4005 posibilidades para 2 drones...: 4005 posibilidades probadas
  Encontrada la mejor solución

  Dron    x     y     z
  ----   ---   ---   ---

  1      100   100   110
  2      300   100   110

  Usuario  x  y  z  Cumple_SNR Dron_asignado Tecnologia Frecuencia_GHz Tasa_Mbps
```

```
-------  -  -  -  ----------  -------------  ----------  --------------  ----------

1          2.5   152.5   1.5   true     1       'n@20MHz' 2.4             43.3
2          3.5      51   1.5   true     1       'n@20MHz' 2.4             43.3
3          3.5    92.5   1.5   true     1       'n@20MHz' 2.4             43.3
4          9.5    78.5   1.5   true     1       'n@20MHz' 2.4             43.3
5           13     181   1.5   true     1       'n@20MHz' 2.4             43.3
... (continúa) ...
48         202    24.5   1.5   true     2       'n@20MHz' 2.4             28.9
49       202.5   169.5   1.5   true     2       'n@20MHz' 2.4             43.3
50       204.5      64   1.5   true     2       'n@20MHz' 2.4             43.3
51         216   108.5   1.5   true     2       'n@20MHz' 2.4             57.8
52         216   180.5   1.5   true     2       'n@20MHz' 2.4             43.3
... (continúa) ...
100        397      39   1.5   true     2       'n@20MHz' 2.4             43.3
```

Como se puede apreciar en la salida anterior, el programa devuelve información detallada sobre el progreso de la simulación así como del resultado que obtiene. Además, la herramienta genera una salida en forma de gráficas que permite comprender mejor el posicionamiento óptimo elegido. Estas gráficas se adjuntan en la Figura 3.7.

Las gráficas anteriores se muestran al usuario en tiempo de simulación, de manera que la primera subgráfica aparece cuando se completa la generación del escenario y la subgráfica de la solución va mostrando en tiempo real el mejor posicionamiento encontrado hasta el momento, actualizándose.

Por último, además de lo anterior, el programa genera la salida del escenario en formato XML. Se ha elegido este formato por el alto grado de estandarización del mismo, de modo que resultará más sencillo su importación en la siguiente fase, que consiste en simular el escenario generado.

La salida (parcial) que corresponde al ejemplo desarrollado en esta sección se puede consultar en Código 3.6. Esta salida es muy sencilla de interpretar, por lo que se considera que no es necesario explicar el significado de cada elemento del mismo.

Código 3.6 Salida XML de la solución en MATLAB.

```
<?xml version="1.0" encoding="utf-8"?>
<escenario ChannelWidth="20" ExpPropagation="3" Frequency="2.4" L="200"
    NumActiveUsers="100" NumUsers="100" Ptx="20" ShortGuardEnabled="true" W="
    400" WifiPhyStandard="WIFI_PHY_STANDARD_80211n_2_4GHz">
  <dron id="1" x="100" y="100" z="110">
    <usuario Active="true" Gain="-4.3943" PhyRate="43.3" WifiPhy="HtMcs4" id="1
        " x="2.5" y="152.5" z="1.5"/>
    <usuario Active="true" Gain="-4.3489" PhyRate="43.3" WifiPhy="HtMcs4" id="2
        " x="3.5" y="51" z="1.5"/>
    <usuario Active="true" Gain="-4.2347" PhyRate="43.3" WifiPhy="HtMcs4" id="3
        " x="3.5" y="92.5" z="1.5"/>
    <usuario Active="true" Gain="-4.0781" PhyRate="43.3" WifiPhy="HtMcs4" id="4
        " x="9.5" y="78.5" z="1.5"/>
    <usuario Active="true" Gain="-4.2819" PhyRate="43.3" WifiPhy="HtMcs4" id="5
        " x="13" y="181" z="1.5"/>
    ... (continúa) ...
  </dron>
  <dron id="2" x="300" y="100" z="110">
    <usuario Active="true" Gain="3.3605" PhyRate="28.9" WifiPhy="HtMcs3" id="48
        " x="202" y="24.5" z="1.5"/>
    <usuario Active="true" Gain="3.6442" PhyRate="43.3" WifiPhy="HtMcs4" id="49
        " x="202.5" y="169.5" z="1.5"/>
```

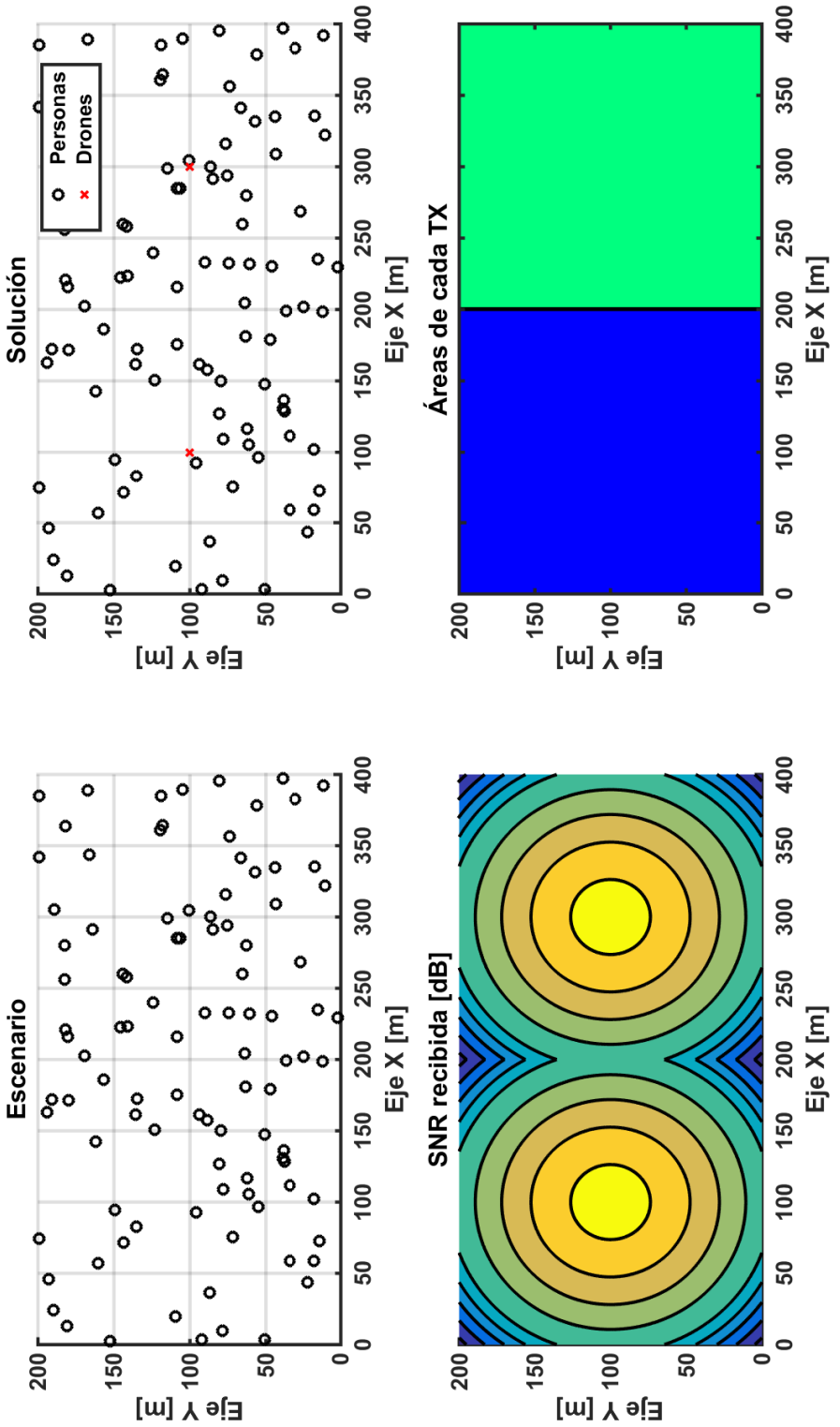


Figura 3.7 Salida gráfica de la solución en MATLAB.

```
    <usuario Active="true" Gain="4.5" PhyRate="43.3" WifiPhy="HtMcs4" id="50" x
        ="204.5" y="64" z="1.5"/>
    <usuario Active="true" Gain="4.5" PhyRate="57.8" WifiPhy="HtMcs5" id="51" x
        ="216" y="108.5" z="1.5"/>
    <usuario Active="true" Gain="3.8936" PhyRate="43.3" WifiPhy="HtMcs4" id="52
        " x="216" y="180.5" z="1.5"/>
    ... (continúa) ...
  </dron>
</escenario>
```

3.3.2 Conclusiones y líneas futuras

La solución desarrollada ofrece una herramienta válida para la generación y evaluación de escenarios según la intensidad de la señal recibida, sin embargo, podría considerarse como un prototipo o marco de trabajo sobre el que se pueden mejorar varios aspectos. Entre ellos, se proponen como líneas futuras las siguientes.

En primer lugar, se propone mejorar la eficiencia del programa del siguiente modo:

- *Sustituir el algoritmo iterativo propuesto* por un algoritmo que converja en una solución necesitando menos iteraciones (por ejemplo, un algoritmo genético).
- Por motivos explicativos, ya que gráficamente es más sencillo de interpretar, en `comprueba_cobertura` la SNR es calculada en cada punto del escenario. Se propone la sustitución de este comportamiento por *evaluar únicamente los puntos ocupados* por usuarios y de este modo se requerirán muchos menos ciclos de procesamiento, aunque en la salida gráfica perderá sentido la representación de la SNR.

En cuanto a la mejora de la fiabilidad de los resultados, se propone lo siguiente:

- *Sustituir las funciones* de propagación y ganancia por otras más similares a los entornos reales.
- *Ampliar las condiciones de cobertura* estimando la calidad de servicio recibida, esto influirá directamente en el número de usuarios que puede soportar un dron. En el último capítulo de este documento se abordará la validación de un modelo analítico de este tipo.

4 Refinamiento por servicio

Hasta ahora, se ha posicionado un conjunto de drones bajo una serie de condiciones de cobertura pero, sin embargo, se desconoce si la QoS que disfrutan los usuarios es suficiente. Lo que se propone a continuación es el refinamiento del algoritmo desarrollado en la fase anterior, de modo que las condiciones de cobertura no impliquen únicamente la intensidad de la señal, sino que además tengan en cuenta la estimación de la MOS percibida por los usuarios.

Para facilitar lo anterior se propone el desarrollo de una *herramienta de simulación* que, a partir del posicionamiento, despliegue la topología y simule el curso de llamadas de VoIP, con el fin de medir estadísticos que ayuden a estimar la calidad de servicio ofrecida.

4.1 Descripción del problema

El principal objetivo de la fase pasa por evaluar la capacidad de respuesta de las WLAN frente al tráfico de VoIP. Para abordarlo, los principales pasos a seguir son los siguientes:

1. *Lectura e interpretación* de los resultados de la fase anterior.
2. *Despliegue de la topología* interpretada.
3. *Caracterización del tráfico* VoIP a simular.
4. *Recolección de los flujos y análisis* estadístico de los mismos.

Como plataforma base para el desarrollo se utilizará el simulador NS3 [18], concretamente la versión 3.26, que ofrece un conjunto de librerías para C++ y Python orientadas a facilitar la simulación de redes. Siguiendo con el guión propuesto, se comienza a continuación describiendo los problemas relacionados con la lectura del fichero de entrada.

4.1.1 Lectura de la topología

La simulación debe partir de un fichero en formato XML que describe una topología de red. Este documento contiene toda la información necesaria, de modo que a partir de él se debe automatizar la instalación de la red sobre el simulador.

La resolución del problema implica, además de la lectura del fichero, el diseño de una clase que almacene los resultados leídos y permita la interpretación sencilla de los mismos.

4.1.2 Despliegue de la topología

A partir del objeto generado en la lectura de la topología, se deben instalar los nodos que en él se describen sobre la simulación. La resolución de este problema implica realizar las siguientes tareas:

- *Elección de los modelos* de WiFi [24] a utilizar, que definen el comportamiento de la red.

- *Inicialización, definición y emparejamiento de los nodos*, que implica el posicionamiento según coordenadas, la asignación de tasa física a utilizar en la transmisión, la asignación de direcciones, etc.

Una vez completados lo mencionado anteriormente, se dispondrá de una simulación completamente funcional y únicamente faltaría la instalación de aplicaciones que cursen el tráfico de VoIP.

4.1.3 Caracterización del tráfico

Una vez desplegada la topología que se quiere evaluar, es necesario definir los flujos de tráfico que se cursarán en la red. En esta tarea se ha seguido de cerca la referencia [5], que ofrece un análisis muy completo del estudio de la capacidad de VoIP sobre WLAN.

En cuanto a los retos que supone la caracterización del tráfico, destacan los siguientes:

- *Elección del modelo* de tráfico a utilizar [25], puesto que existen una variedad de ellos ya implementados en NS3.
- *Elección del códec* a utilizar en las llamadas [26].
- *Definición* de pares de nodos entre los cuales se cursará *tráfico*.
- *Minimizar las irregularidades* que puedan reflejarse en el tráfico generado por motivos de simulación.

4.1.4 Recolección de flujos

En cuanto a los retos que se incluyen en la recolección y análisis de los flujos cursados, los principales aspectos a tener en cuenta son los siguientes:

- *Elección de los estadísticos* a medir.
- *Elección del módulo de NS3* a utilizar para la medición.

Realmente, la primera de ellas ya se resolvió parcialmente en la Sección 2.2, concretando que para la estimación de la MOS, los principales parámetros que se necesitan son el caudal, la tasa de pérdidas de paquetes, el retardo y la variación entre el retardo.

4.2 Solución desarrollada

Del mismo modo que en la primera fase, se aporta en la Tabla 4.1 el desglose de ficheros que forman parte de esta herramienta, así como una breve descripción de los problemas que tratan.

Tabla 4.1 Ficheros que componen la solución en NS3.

Fichero	Extensión	Cometido
simulador	*cc*	Contiene la función `main` del programa, responsable principal de la simulación
escenario	*h*	Contiene la declaración de las clases `Escenario`, `Dron` y `Usuario`; utilizadas para el almacenamiento de los datos leídos
funciones	*cc*/*h*	Contiene funciones de carácter general, utilizadas para la construcción de la simulación.
observador	*cc*/*h*	Contiene la clase `Observador`, encargada de las tareas de recolección de estadísticas.
mapped-rate-wifi-manager	*cc*/*h*	Contiene la clase `MappedRateWifiManager`, basada en `ConstantRateWifiManager`, utilizada para la selección de la tasa física a partir de una tabla.

A continuación se procede con la descripción de la resolución de cada una de los problemas implicados, manteniendo el esquema propuesto en el apartado anterior.

4.2.1 Lectura de la topología

La lectura se realiza a través de una función llamada `cargaEscenario` ubicada en el fichero *funciones.cc*. En él, se ha decidido utilizar un *parser* de contenido XML desarrollado en C llamado `libxml` [27]. La decisión radica en su sencilla integración en el entorno NS3.

En cuanto al algoritmo que se sigue, se trata de una función recursiva que itera en cada nodo del fichero XML [28] volcando los datos reconocidos en un objeto de la clase `Escenario`. Dicho algoritmo es, aunque laborioso, simple de comprender; por ello no se va a entrar en detalles sobre la implementación y se adjunta el código del mismo en el Apéndice B.

La clase `Escenario` contiene, además de los atributos globales al mismo, un vector de objetos de la clase `Dron`. Del mismo modo, esta última contiene un vector de objetos de la clase `Usuario`. La relación entre las clases puede verse reflejado en el diagrama de la Figura 4.1.

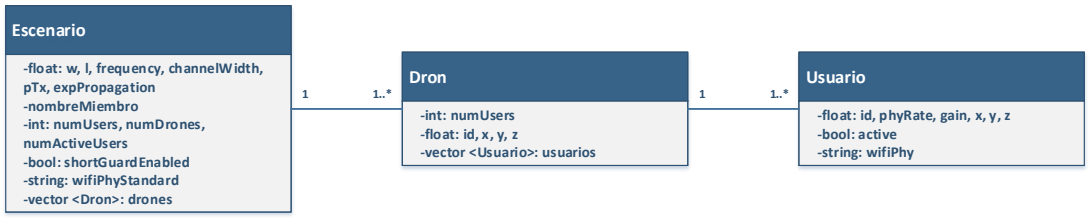


Figura 4.1 Diagrama de clases de las entidades que definen el escenario.

Una vez completado el objeto `Escenario`, es posible llevar a cabo el despliegue de la topología que en él se describe.

4.2.2 Despliegue de la topología

El despliegue de la topología sobre el simulador se puede dividir en varios pasos que se listan a continuación:

1. *Definición de la arquitectura* de la red.
2. *Definición de los modelos* de red a utilizar.
3. *Instalación de los nodos*.
4. *Asignación de direcciones* a los nodos.

A continuación se explican los detalles de cada una de las etapas anteriormente mencionadas.

Arquitectura de la red

Si bien es cierto que la topología que siguen los drones y usuarios ya está definida según el fichero de entrada, es necesario decidir entre qué entidades se cursarán las llamadas. Por ello se propone seguir la red descrita en el diagrama de la Figura 4.2.

Como se puede apreciar en la figura, cada dron será un punto de acceso WiFi al que se conectarán los usuarios definidos en el fichero de entrada. Además, cada punto de acceso se encuentra conectado mediante un enlace punto a punto a un equipo que actuará como servidor de las llamadas de VoIP.

Elección de modelos

Una vez decidida la arquitectura, es necesario concretar los modelos que regirán el comportamiento de la red. Para ello, NS3 proporciona la clase `Node` que representa a un nodo de la red. A partir de esta clase genérica se permite la definición completa del comportamiento de los nodos.

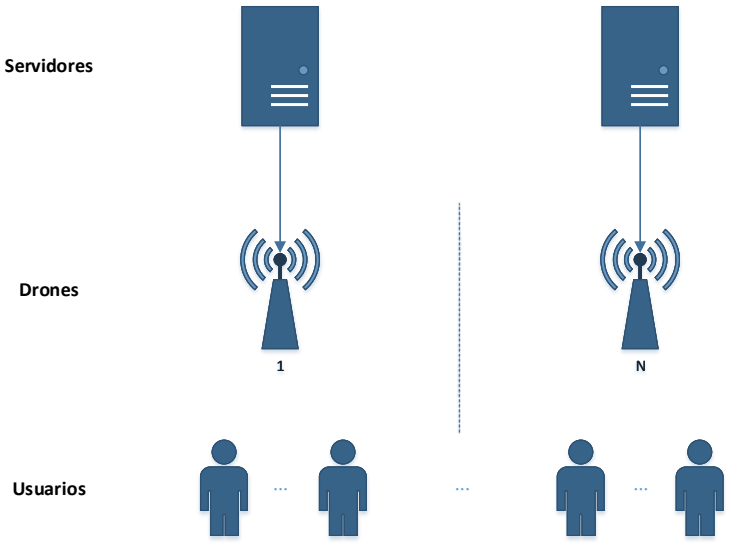


Figura 4.2 Diagrama de la red a simular.

Una de las entidades a definir por cada `Node` son los `NetDevice`, que modelan y representan al controlador de una interfaz de red. Dependiendo de la naturaleza de la interfaz se utilizan diferentes tipos de `NetDevice`. En nuestro caso, se utilizan los siguientes:

- Nodos `users`: Presentan un `WifiNetDevice`.
- Nodos `drones`: Tienen un `WifiNetDevice` y un `PointToPointNetDevice`.
- Nodos `servers`: Utilizan un único `PointToPointNetDevice`.

Para cada `NetDevice`, NS3 lo implementa siguiendo una división en capas lógicas [24]:

- La capa `PHY`, que modela el nivel físico.
- La capa `MAC low`, que representa los modelos del nivel MAC a bajo nivel.
- La capa `MAC high`, que representa los modelos MAC de alto nivel.

Además, cada una de las anteriores capas se descomponen en modelos más concretos, en Figura 4.3 se presenta la arquitectura de un `WifiNetDevice`.

En base a lo anterior, en la Tabla 4.2 se adjunta los modelos utilizados para cada tipo de nodo en la simulación.

Tabla 4.2 Resumen de modelos utilizados en la simulación.

Tipo de nodo	NetDevice	Phy	Mac	Channel
servers	PointToPointNetDevice	-	-	PointToPointChannel
drones	WifiNetDevice PointToPointNetDevice	SpectrumWifiPhy -	WifiMac -	SpectrumChannel PointToPointChannel
users	WifiNetDevice	SpectrumWifiPhy	WifiMac	SpectrumChannel

Además, para cada modelo particular existen una serie de parámetros modificables que permiten personalizar el comportamiento del nodo. A continuación, en la Tabla 4.3, se presentan los parámetros que requieren configuración en la simulación que se propone.

En la tabla anterior, se relaciona cada entidad con los parámetros que se han decidido configurar, así como el valor de estos si este es fijo para todos los nodos. Si, en cambio, depende de cada nodo, se ha marcado con un asterisco o se han mencionado ambas posibilidades.

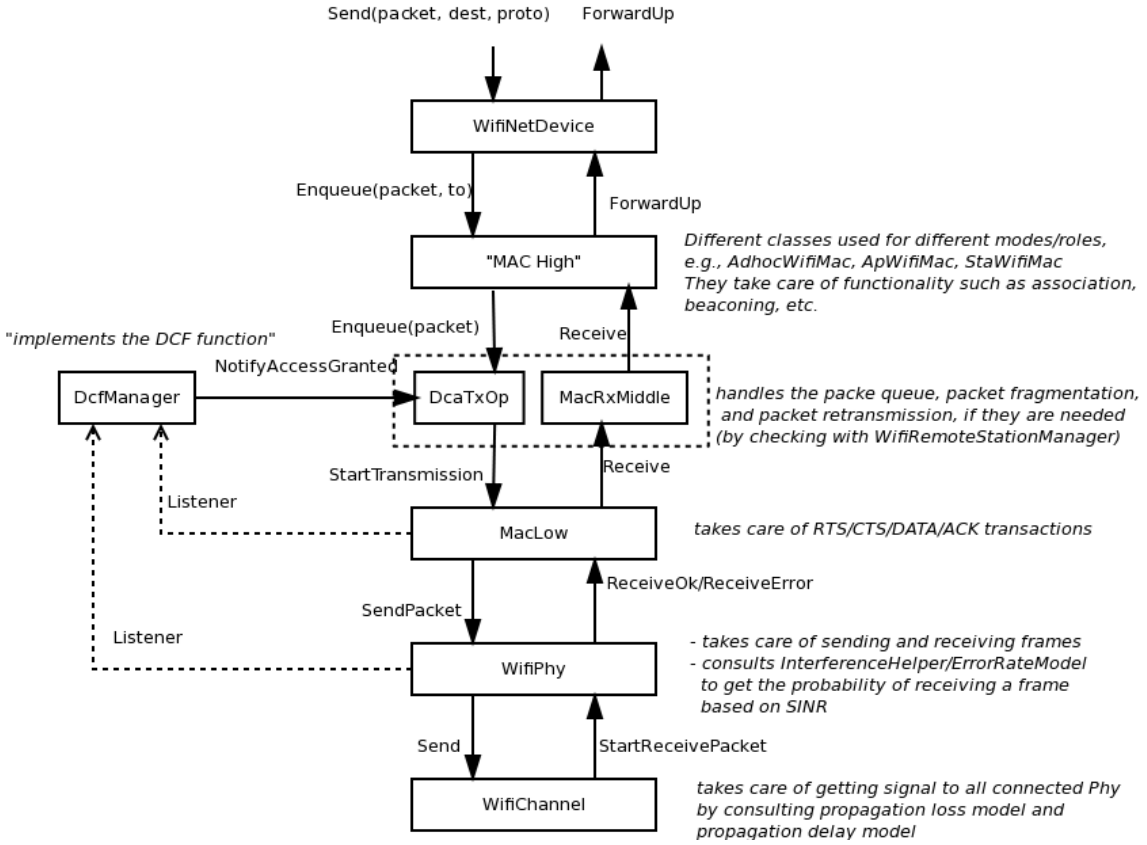


Figura 4.3 Arquitectura de un `WifiNetDevice`.

Tabla 4.3 Parámetros configurados por cada modelo.

Entidad	Parámetros	Valor
`PointToPointNetDevice`	`DataRate`	`1Gbps`
`WifiNetDevice`	`Standard` `RemoteStationManager`	* `(Mapped/Constant)RateWifiManager`

`SpectrumWifiPhy`	`ErrorRateModel`
	`ShortGuardEnabled`
	`TxPowerStart`
	`TxPowerEnd`
	`ChannelWidth`
	`RxNoiseFigure`
		`NistErrorRateModel`
		*
		*
		*
		*
		0

`WifiMac`	`Type` `MaxPacketNumber` (Queue)	`(Ap/Sta)WifiMac` *
`PointToPointChannel`	`Delay`	`0ms`

`SpectrumChannel`	`Channel`
	`PropagationLoss`
	`PropagationDelay`
		`MultiModelSpectrumChannel`
		`LogDistancePropagationLossModel`
		`ConstantSpeedPropagationDelayModel`

Todos los modelos utilizados salvo uno son implementados de manera nativa por NS3 y se han utilizado sin modificaciones. La excepción reside en el modelo `MappedRateWifiManager`.

`MappedRateWifiManager` es un tipo de `RemoteStationManager` desarrollado para este proyecto, cuya función es asignar una tasa física (MCS) para cada transmisión.

En el caso de los usuarios, basta con utilizar la clase `ConstantRateWifiManager` ya implementada en NS3, que permite establecer una tasa física constante y única para todas las transmisiones de un nodo. En cambio, para los drones, ha sido necesario desarrollar un `RemoteStationManager` que asigne una tasa

diferente dependiendo del destino de cada paquete.

Las relaciones entre dirección MAC destino y MCS son pasadas como parámetro en forma de tabla al asignar este `RemoteStationManager` al `WifiNetDevice`.

Declaración de los nodos

Una vez descrito el conjunto de modelos a utilizar en cada caso, es posible proceder con la instalación de los nodos en el simulador. El proceso seguido para recorrer los nodos almacenados en el objeto `Escenario` está descrito en la Figura 4.4 (31).

Puesto que los objetos se almacenan en vectores del tipo `std::vector`, es muy sencillo recorrerlos con un `iterator` e ir instalándolos uno a uno. Es importante comprender el orden en que se instalan los nodos, ya que así es posible compartir ciertos parámetros de configuración entre nodos de diferente tipo.

Por ejemplo, entre las redes desplegadas por los diferentes drones se utiliza un SSID (*Service Set Identifier*) diferente, y este se modifica antes de instalar todos los usuarios de un mismo dron.

En el algoritmo anterior se instalan las interfaces `WifiNetDevice`. Por el contrario, los `NetDevice` de las interfaces punto a punto no requieren configuración adicional y basta con incorporar las líneas del Código 4.1.

Código 4.1 Instalación de los `PointToPointNetDevice`.

```
for ( uint16_t cont = 0; cont < NumDrones; cont++){
  wiredDevices.push_back(p2p.Install(NodeContainer(drones.Get(cont), servers.
      Get(cont))));
}
```

Direccionamiento de los nodos

Una vez completada la instalación de los nodos, es necesario definir el direccionamiento de los mismos. En la Figura 4.5 se especifica tanto el direccionamiento como el nombrado de las SSID.

Una vez asignada la dirección a cada interfaz de red, se procede a relenar la tabla de encaminamiento de los nodos con ayuda del módulo `Ipv4GlobalRoutingHelper`. Sin embargo, este módulo no tiene un comportamiento adecuado en algunas topologías WiFi y es necesario completar las tablas generadas.

Por ello, a cada usuario se le ha configurado a mano su pasarela por defecto, siendo esta el dron con el que se encuentra emparejado. Para ello se ha utilizado el módulo `Ipv4StaticRoutingHelper` de NS3.

Por último, la tabla ARP de los nodos está vacía, para ello hay dos soluciones. O bien no rellenarla, de modo que en tiempo de simulación los equipos enviarán paquetes ARP para hacerlo; o, por lo contrario, rellenar estas tablas a mano.

Inicialmente se intentó mantener el aprendizaje clásico de los equipos, sin añadir entradas ARP a mano. Sin embargo, cuando el tamaño de la cola de transmisión del punto de acceso era unitario, el aprendizaje no se realizaba correctamente.

Por este motivo se propone el aprendizaje de las tablas antes de comenzar la simulación, mediante la función `PopulateArpCache` proporcionada en el fichero *funciones.cc/h*. En [29] se discute el uso de esta función y aunque presenta una serie de desventajas, no afectan al desempeño de esta simulación.

Una vez completado el diseño del direccionamiento es posible instanciar las aplicaciones que cursarán tráfico en la simulación.

4.2.3 Caracterización del tráfico

En la definición de los flujos a cursar se hará uso del módulo `Application` que implementa NS3, concretamente se usarán las aplicaciones `UdpClient` y `PacketSink`. La elección se debe al tipo de tráfico que se ha decidido cursar.

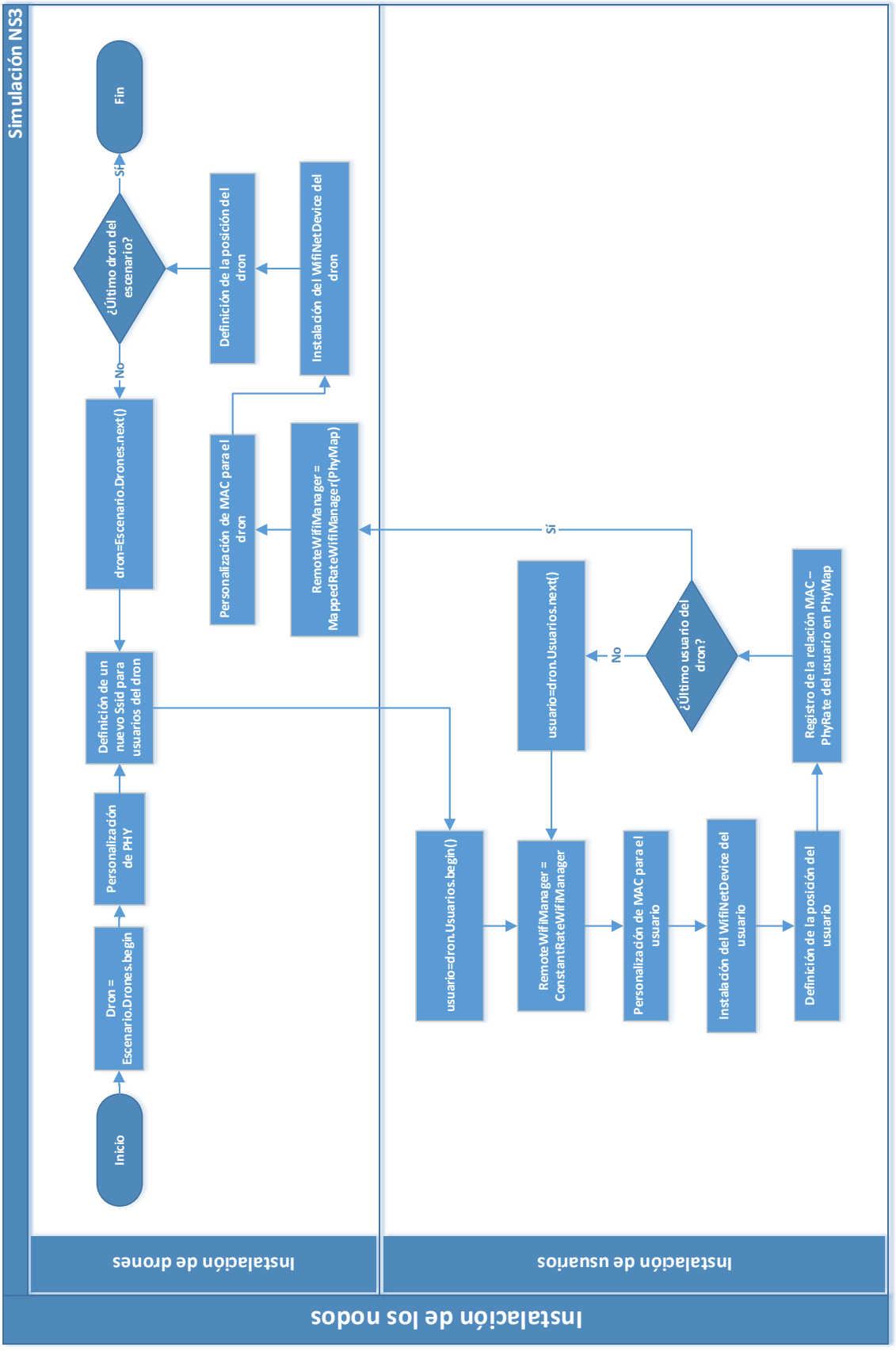


Figura 4.4 Diagrama de flujo de la instalación de los nodos en NS3.

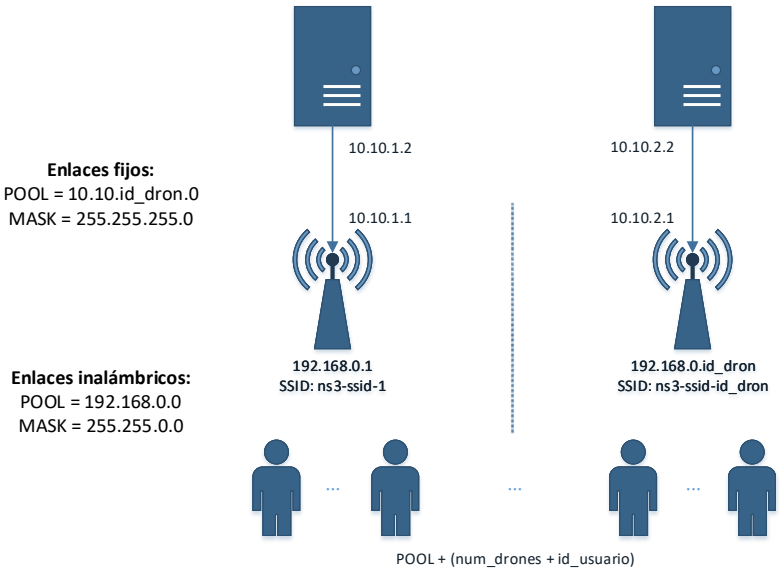


Figura 4.5 Asignación de direcciones en la red.

Concretamente, se va a utilizar el códec G.711, que toma una muestra de voz cada 125 microsegundos y utiliza una duración de trama de 20 milisegundos. De este modo, cada paquete a enviar contiene 160 bytes de datos. Además es necesario incluir los octetos correspondientes a la cabecera RTP (*Real-time Transport Protocol*), para ello se han considerado 12 bytes, de modo que el tamaño final de los datos a transmitir es de 172 octetos.

En adición, se ha decidido simular tráfico CBR (*Constant Bit Rate*) y bidireccional, puesto que se trata del caso más restrictivo. Esto desemboca en instalar aplicaciones tanto del tipo `UdpClient`, como `PacketSink` en todos los nodos usuarios y servidores.

La inicialización de todas las aplicaciones al mismo tiempo supone un problema, de modo que se ha seguido el orden que se indica en la Figura 4.6 en la planificación temporal de las mismas. El problema radica en que si se iniciaran todas las fuentes al mismo tiempo, el resultado de la simulación se vería adulterado por el encolado masivo inicial.

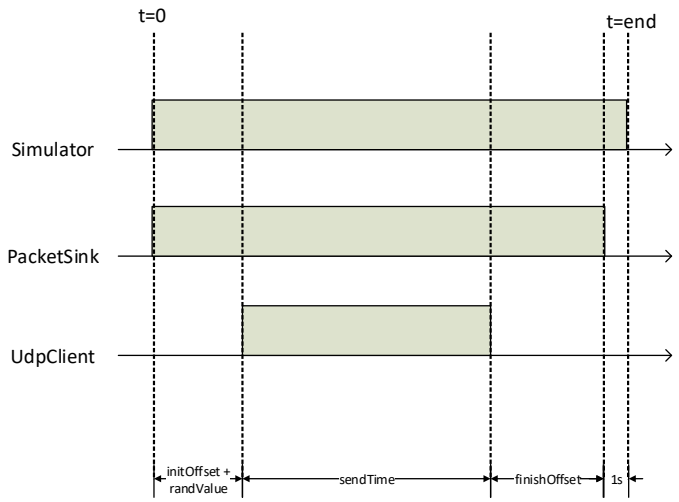


Figura 4.6 Planificación temporal de la simulación.

En la figura anterior, los valores `initOffset`, `sendTime` y `finishOffset` son fijos a lo largo de la simulación. Por contra, cada fuente `UdpClient` calcula un valor aleatorio `randValue` que solucionará los problemas anteriormente mencionados.

Una vez planificado el tráfico generado y consumido por las aplicaciones, la simulación es plenamente funcional, lo que sigue a continuación es la recolección de flujos y el análisis estadístico de los mismos.

4.2.4 Recolección de flujos

La recolección de flujos se programa haciendo uso del módulo `FlowMonitor` implementado por NS3. Por limpieza y estructuración del código, todo lo relativo a la obtención de estadísticas y parámetros se realiza a través de la clase `Observador` definida en *observador.cc/h*.

Gracias a la clase `Observador`, la recolección de flujos es totalmente transparente, siendo necesario únicamente la sucesión de las líneas recogidas en el Código 4.2.

Código 4.2 Uso de la clase `Observador`.

```
Observador * observador = new Observador(packetSize, sendTime);

... (Instalación del escenario) ...

// Instalo el monitor de flujos en todos los nodos
observador->installFlowMonitor();

... (Simulación) ...

// Obtención de parámetros con los que se ha simulado
WifiParameters params = observador->getWifiParameters(droneDevices.Get(0));

// Obtención del conjunto de estadísticas de los flujos cursados
Statistics stats = observador->getFlows();
```

Como se puede observar en el fragmento de código anterior, se han obtenido los parámetros en un `struct` del tipo `WifiParameters` y las estadísticas en un objeto de la clase `Statistics`. A continuación, en la Tabla 4.4, se adjunta el resumen de los datos almacenados en cada uno de ellos.

Tabla 4.4 Resumen de datos obtenibles mediante la clase `Observador`.

Entidad	Tipo	Parámetros
`WifiParameters`	`Struct`	`CWmin, CWmax, timeSlot, DIFS, EIFS, SIFS, SLRC, SSRC, GI, standard`
`Statistics`	Clase	`rate, per, delay, jitter`

Respecto a los datos obtenidos desde `Statistics`, se obtiene por cada campo una tabla de dos elementos, donde el primero almacena la media de los datos de bajada (servidor a usuario) y, el segundo almacena la media de los datos de subida (usuario a servidor).

Como se puede observar, los datos obtenibles son suficientes para estimar la MOS media percibida, si se quisiesen obtener datos particulares de cada usuario también sería posible mediante el módulo `FlowMonitor`.

4.3 Salidas y resultados

Con el fin de comprobar si la simulación es correcta, se propone el comportamiento de cada estándar de los implementados conforme crece el número de usuarios de la red.

Para favorecer la interpretación de los resultados se partirá de un caso homogéneo, es decir, todos los usuarios disfrutan de la misma tasa física. Además, se forzará la tasa física más baja posible, para acelerar la saturación de la capacidad de la red. Adicionalmente, se estudiará la influencia del tamaño de la cola de transmisión del punto de acceso.

4.3.1 Ejemplo de ejecución

Para obtener un intervalo de confianza suficiente se han requerido 30 iteraciones (29 grados de libertad). La automatización de la simulación completa se ha realizado mediante el *script* de *shell* que se adjunta a continuación.

Código 4.3 Script que automatiza la simulación propuesta.

```
#!/bin/bash
# simula.sh

SIM_TIME=10;
CUENTA=0;
TOTAL=$((30*2*20*4))

for ESTANDAR in "802.11b" "802.11g" "802.11n" "802.11ac"; do
  DIR_ENTRADA="scratch/TFM/entrada/$ESTANDAR";
  DIR_SALIDA="scratch/TFM/salida/$ESTANDAR";

  HEADER="Run;Standard;ChannelBandWidth(MHz);N;K;CWmin;CWmax;Ts(us);DIFS(us);
      SIFS(us);EIFS(us);SLRC;SSRC;GI(ns);BER;Throughput;PacketLoss;Delay;Jitter
      ";

  if [ ! -d $DIR_SALIDA ]; then
    mkdir $DIR_SALIDA
  fi

  echo $HEADER > $DIR_SALIDA/resultados.txt
  echo "" $DIR_SALIDA/log

  for NUMUSERS in {1..20}; do
    for QUEUESIZE in `echo 1 $NUMUSERS`; do
      for RNGRUN in {1..30}; do

        echo ""
        printf "E=$RNGRUN\tN=$NUMUSERS\tK=$QUEUESIZE\n"
        echo ""

        NS_GLOBAL_VALUE="RngRun=$RNGRUN" ./waf --run " TFM --inputFile=
            $DIR_ENTRADA/$NUMUSERS.xml --outputDir=$DIR_SALIDA/ --queueSize=
            $QUEUESIZE --simulationTime=$SIM_TIME" >> $DIR_SALIDA/log

        CUENTA=$((CUENTA+1));
        echo ""
        cat $DIR_SALIDA/salida.txt >> $DIR_SALIDA/resultados.txt
        echo "Terminada simulación $CUENTA/$TOTAL"

      done
    done
  done
done
```

El *script* anterior vuelca los resultados, previamente preparados para que se impriman con un formato en concreto, en un fichero. La interpretación de los resultados se hará mediante MATLAB pero no se entrará en detalles acerca de su implementación.

Una vez finalizada la simulación, los resultados que arroja se presentan en las gráficas contenidas en la Figura 4.7 y Figura 4.8.

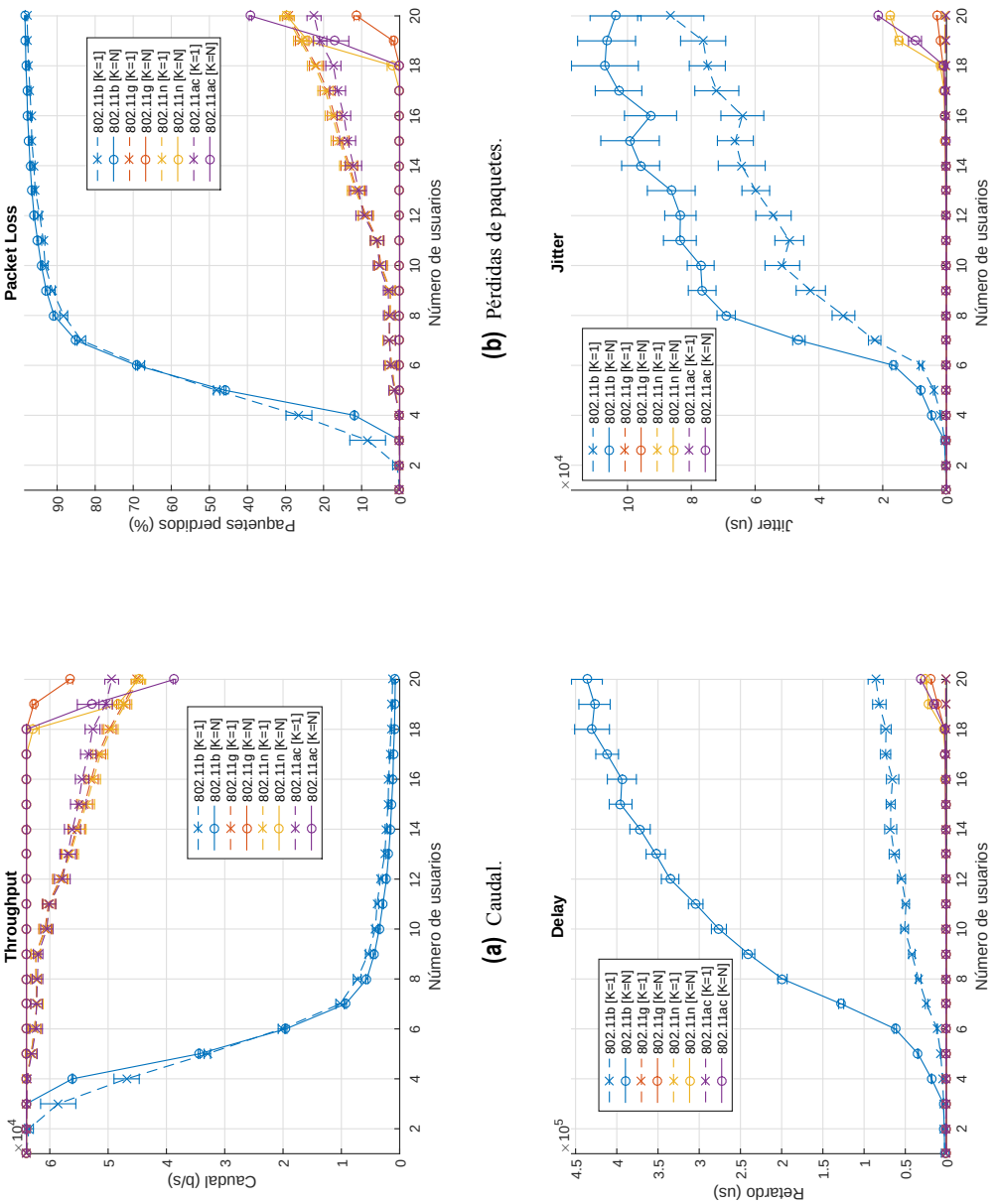
De las curvas obtenidas se obtiene como conclusión que el tráfico más desfavorable en el entorno simulado es el canal de bajada, fruto del encolado de los paquetes. A pesar de que en el canal de subida existen colisiones, su efecto es mucho menor al anteriormente mencionado.

4.3.2 Conclusiones y líneas futuras

La herramienta de simulación desarrollada permite desplegar la solución de la fase anterior a partir de su fichero de salida. Además, con muy pocas modificaciones es posible adaptarla en una herramienta de carácter más general, que estudie la capacidad de una topología WLAN bajo cualquier servicio.

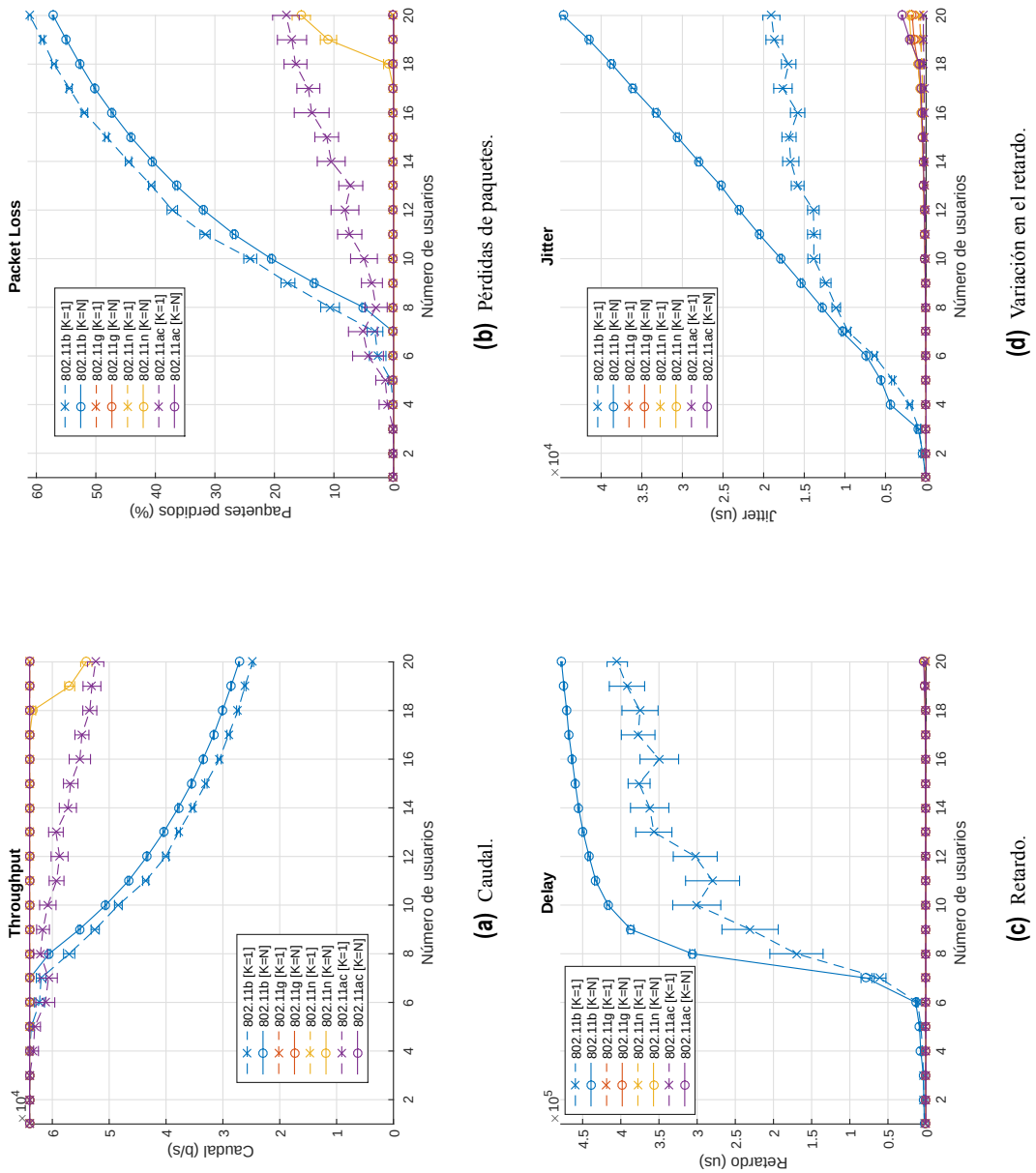
Como principal mejora que se propone destaca el traslado de la generación del intervalo de confianza al código del programa, evitando utilizar desde fuera un *script* de *shell* y, de este modo, se mejorará el tiempo requerido para completar la simulación. En el caso actual, se ha preferido utilizar este método puesto que favorece la división de la simulación entre diferentes equipos o incluso entre *cores* de un mismo equipo.

Por último, se plantea el estudio de los parámetros de WiFi a utilizar en la simulación, puesto que en la anterior se han mantenido, en muchos casos, los propuestos por defecto por NS3. Una buena referencia para abordar esta tarea es el estándar IEEE 802.11ax, además del artículo [5], que estudian esta problemática.



(a) Caudal. **(b)** Pérdidas de paquetes. **(c)** Retardo. **(d)** Variación en el retardo.

Figura 4.7 Resultados del tráfico de bajada por simulación.



(a) Caudal. **(b)** Pérdidas de paquetes. **(c)** Retardo. **(d)** Variación en el retardo.

Figura 4.8 Resultados del tráfico de subida por simulación.

5 Confirmación del modelo analítico

Llegados a este punto, haciendo uso de la herramienta de simulación desarrollada en la fase anterior, es posible la evaluación de modelos analíticos de WiFi para su futura incorporación en la primera fase.

Por lo tanto, se propone la confirmación de un modelo analítico de multiplexación homogénea en el que, como su propio nombre indica, todas las estaciones presentan la misma tasa física.

5.1 Descripción del problema

El algoritmo desarrollado para el posicionamiento inicial carece de métodos para estimar la calidad de servicio que percibirán los usuarios. Esto resulta en que, por ejemplo, a un punto de acceso se le puede asignar un número cualquiera de usuarios, sin tener en cuenta el agotamiento de la capacidad disponible.

Con el fin de atenuar el problema anterior, se propone el estudio de la validez de un modelo analítico de modo que, una vez confirmado, pueda ser añadido al algoritmo de la fase de posicionamiento para ofrecer resultados que tengan en cuenta la calidad de servicio.

Para ello, se propone el modelo analítico desarrollado en [30], que trata la estimación resultados de calidad de servicio en entornos con carga limitada (sin saturación).

En definitiva, se propone validar el modelo anterior para un caso homogéneo. Para ello se utilizará como referencia la simulación aportada en la Sección 4.3.

5.2 Modelo analítico y parámetros en la comparación

En [30] se propone un modelo matemático que trata de estimar el comportamiento de CSMA/CA en entornos sin saturación. El modelo teórico anterior parte de unos datos de tráfico y encolado supuestos, de modo que es posible obtener información interesante para estimar la calidad de servicio (principalmente el caudal).

En definitiva, este modelo puede considerarse a alto nivel como una función cuyos parámetros de entrada definirán un escenario WLAN y permitirá obtener información sobre el caudal y el retardo (principales responsables de la MOS).

En base a lo anterior, se propone estudiar el escenario homogéneo de la sección anterior en el modelo analítico con el fin de analizar diferencias en los resultados. El conjunto de parámetros utilizados, tanto en la simulación como en el modelo analítico, se encuentran expuestos en la Tabla 5.1.

Además de los parámetros anteriores, el modelo requiere como entrada la probabilidad de error de un paquete debido al ruido. Para ello se ha consultado los modelos de error que implementa NS3 y se han llegado a las siguientes conclusiones.

A pesar de que en la simulación se ha especificado utilizar el modelo `NistErrorRateModel` [31], si consulta se la documentación de este se observa que para el caso del estándar IEEE 802.11b, al utilizarse la

Tabla 5.1 Parámetros utilizados en la confirmación del modelo analítico.

Parámetro	Valor
Estándar	802.11b
Tasa	1 Mbps
Frecuencia	2.4 GHz
Ancho de banda de canal	22 MHz
CWmin	31
CWmax	1023
Time Slot	20 us
DIFS	50 us
SIFS	10 us
EIFS	364 us
SLRC	7
SSRC	7
Intervalo de guarda	800 ns

modulación DSSS (en lugar de OFDM que se utiliza para el resto de estándares estudiados), el modelo de error que se utiliza es el `DSSSErrorRateModel` [32].

A partir del modelo de error de la modulación DSSS detallada en términos teóricos en el artículo [33], se puede extraer la ecuación seguida para calcular la probabilidad de error de bit:

$$BER = 1/2 \cdot e^{-Eb/N_0}$$

La ecuación anterior requiere el cálculo de la relación de energía de bit a densidad de potencia de ruido, que se calcula como sigue a continuación:

$$Eb/N_0 = snr \cdot B/f$$

Donde *snr* representa el valor de la SNR en unidades naturales, *B* el ancho de banda del canal (en nuestro caso 22 MHz), y *f* representa la tasa física (1 Mbps). El valor de la SNR es conocida, puesto que indirectamente se trata de una de las variables de diseño de la topología.

A partir de la BER anterior es posible calcular la probabilidad de enviar correctamente un paquete:

$$P_{success} = (1 - BER)^{nBits}$$

El número de bits del paquete se calcula a continuación para el caso propuesto, teniendo en cuenta las cabeceras implicadas en la comunicación:

$$Bytes_{paquete} = 802.11\ (24) + LLC\ (8) + IP\ (20) + UDP\ (8) + RTP\ (12) + Datos\ (160) = 232\ bytes$$

Para el escenario propuesto, siguiendo los cálculos anteriores, la probabilidad de error de un paquete sería de $1.89582 \cdot 10^{-31}$. Sin embargo, por motivos de precisión en la simulación, se considera una probabilidad de error igual a 0. El motivo es que NS3 trata esta probabilidad como un `double` y, puesto que este solo almacena hasta 15 dígitos decimales, acaba considerando una probabilidad de error nula.

El resultado de esto es que no existen retransmisiones en la capa MAC y, en consecuencia, el tamaño de cola impuesto no afecta al resultado.

La conclusión que se puede obtener de este cálculo es que la probabilidad de error de paquete, por efectos del ruido, se va a considerar 0.

5.3 Resultados

Para contrastar los resultados obtenidos mediante ambos métodos, se ha utilizado MATLAB como plataforma. En la Figura 5.1 se adjuntan una serie de gráficas que muestran la comparativa entre el caudal, tasa de pérdidas y retardo obtenidos por ambos métodos.

En cuanto a la comparativa del caudal medio (Figura 5.1a), se observa que aunque el comportamiento para un número pequeño de nodos (entre 1 y 4) es similar, conforme aumenta el número de usuarios el resultado arrojado por el modelo se aleja de la curva de la simulación.

En VoIP, el aumento de las pérdidas de paquetes (Figura 5.1b) es muy abrupto [5] a partir de un número de nodos dado (en el caso actual, a partir de 3 nodos) y el modelo analítico utilizado es más conservador en la estimación de esta pérdida, arrojando resultados poco precisos.

Respecto al retardo medio percibido, en la Figura 5.1c se puede observar que los resultados están lejos de ser similares. Mientras que la curva de simulación asciende hasta aproximadamente 450 milisegundos cuando hay 20 estaciones, el modelo analítico apenas alcanza los 25 milisegundos. De hecho, este último presenta un comportamiento prácticamente lineal.

5.3.1 Discusión de los resultados

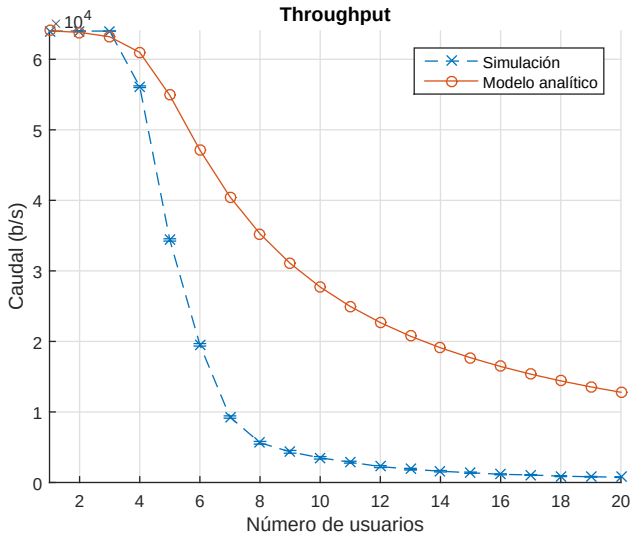
En los resultados anteriores se observa una clara diferencia entre ambos métodos lo cual indica que la investigación no finaliza aquí. Sin embargo, a pesar de las diferencias, las curvas de caudal y pérdidas de paquetes obtenidas por ambos métodos presentan un comportamiento en general similar (el rendimiento comienza a decaer a partir de tres nodos).

La discrepancia puede estar condicionada por varios motivos, a priori se contemplan las siguientes posibilidades:

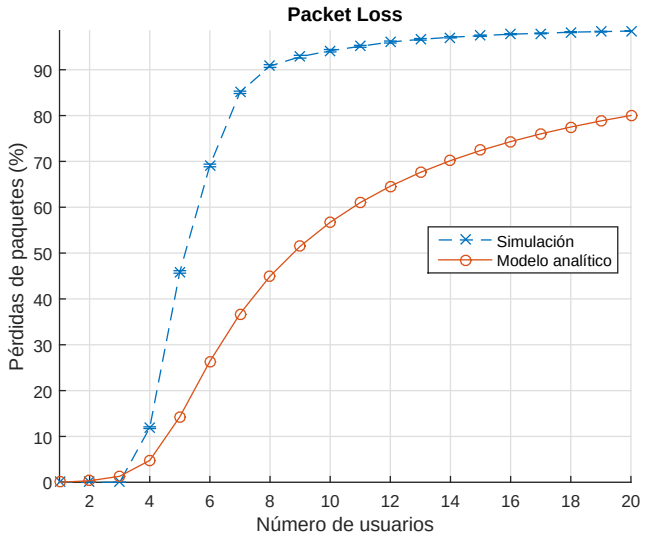
En primer lugar, es posible que se hayan cometido errores en la elección de los parámetros o modelos utilizados en la herramienta de simulación propuesta en la fase anterior. Del mismo modo, se contempla la posibilidad de haber utilizado una mala configuración en la puesta a punto del modelo analítico.

Por último, es posible que se haya cometido algún error en la extracción de parámetros de simulación utilizados (Tabla 5.1), de modo que el escenario que se está estudiando en ambos casos difieren en algún aspecto.

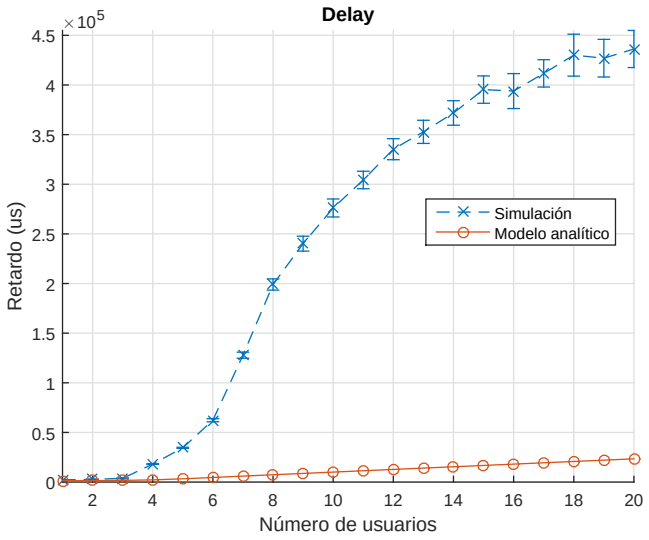
En cualquier caso, en este punto no es posible concluir en una decisión acerca del modelo analítico estudiado, de modo que se deja para futuros estudios el análisis de la causa que puede estar provocando las discrepancias, así como la decisión a tomar acerca del modelo propuesto.



(a) Caudal.



(b) Pérdidas de paquetes.



(c) Retardo.

Figura 5.1 Comparativa entre modelo analítico y simulación.

6 Conclusiones y líneas de avance

Las redes inalámbricas de área local suponen un recurso a tener en cuenta en el despliegue de servicios en tiempo real. En el ámbito del proyecto, se ha estudiado el procedimiento a seguir en el despliegue de un servicio de VoIP en tipo de evento particular, como puede ser un concierto o una zona accidentada.

Respecto al procedimiento descrito, se ha procurado desglosar el conjunto de problemas ofreciendo tres fases de diseño, que han dado lugar a; una herramienta para el posicionamiento inicial, un marco de simulación de la red propuesta y una serie de pautas a seguir en la validación de modelos analíticos útiles para la idea que se persigue.

En cuanto a las conclusiones adquiridas en la realización de este proyecto, destaca en primer lugar la importancia del estándar a elegir en el despliegue de la WLAN. El estándar 802.11b, al presentar una tasa física tan baja (máximo 11Mbps) dada por la modulación utilizada, se encuentra realmente limitada para abastecer de un servicio de VoIP a un conjunto de usuarios. Por contra, los estándares estudiados basados en OFDM (IEEE 802.11g/n/ac) amplían considerablemente la capacidad de la WLAN, mejorando la calidad de servicio.

En segundo lugar, se ha confirmado que la comunicación más restrictiva en cuanto a la calidad de servicio es aquella cursada en el canal de bajada, propiciada por el encolado de los paquetes. A pesar de que en el canal de subida existen colisiones, el efecto de estas es menor al anteriormente mencionado.

Por último, se ha comprobado el efecto del tamaño de la cola del punto de acceso, ofreciendo como resultado que en el caso del caudal, un menor tamaño de cola resulta en unas prestaciones más bajas; mientras que en el caso del retardo, un menor tamaño de cola desemboca en menores tiempos de espera en la cola y, en definitiva, menores valores de retardo.

Como línea de avance principal, se propone continuar con la investigación sobre el modelo analítico propuesto en este documento, con el fin de detectar los motivos por los que no se obtienen resultados similares. Una vez resuelto lo anterior, resulta interesante ampliar el caso homogéneo a uno heterogéneo, más fiel al tipo de eventos que se pretenden cubrir. De modo que una vez diseñado un modelo válido, se incorpore a la fase de posicionamiento inicial para dar lugar a soluciones más precisas.

Por último, una vez acotado el posicionamiento inicial de los drones, se propone la investigación de un algoritmo que actualice la posición de los mismos con el fin de mantener las restricciones de calidad impuestas, ya que los usuarios pueden cambiar de posición en cualquier momento.

Apéndice A
MATLAB

A.1 Definición de clases

A continuación se presentan las clases utilizadas en la herramienta MATLAB.

Código A.1 Escenario.m.

```
classdef Escenario < handle
    %ESCENARIO Permite colocar aleatoriamente personas a partir de una función
        densidad de probabilidad dada.

    properties
        Size
        Matrix
    end

    methods
        % Constructor
        function obj = Escenario(W,L,paso)
            obj.Size = [(L/paso)+1 (W/paso)+1];
            obj.Matrix = zeros([(L/paso)+1 (W/paso)+1]);
        end

        % Añade puntos según PDF
        function addPoints(obj, pdf, nPoints)
            if size(pdf) ~= size(obj.Matrix)
                error('No concuerdan las dimensiones')
            elseif sum(sum(pdf<0)) ~= 0
                error('La función densidad de probabilidad no es válida: contiene
                     elementos negativos')
            elseif abs(sum(sum(pdf))-1) > 1e-6
                error('La función densidad de probabilidad no es válida: la suma
                    de los elementos es %d y debe ser 1', sum(sum(pdf)))
            end
            pointsLeft = nPoints;
            factor = max(max(pdf));
            while pointsLeft > 0
                xIndex = randi([1 obj.Size(1)]);
                yIndex = randi([1 obj.Size(2)]);
                if obj.Matrix(xIndex, yIndex) == 1
```

```
                % No hacemos nada, ya hay un 1
            else
                if rand*factor < pdf(xIndex, yIndex)
                    obj.Matrix(xIndex, yIndex) = 1;
                    pointsLeft = pointsLeft - 1;
                end
            end
        end
    end

    % Devuelve resultados en formato coordenadas
    function indexMatrix = getCoordinates(obj)
        % Le doy la vuelta a los índices para tener la lógica que yo
        % quiero
        [yIndex, xIndex] = ind2sub(obj.Size, find(obj.Matrix>0)');
        indexMatrix = [xIndex - 1; yIndex - 1];
    end

    % Devuelve resultados en formato matriz
    function matrix = getMatrix(obj)
        matrix = obj.Matrix;
    end
  end
end
```

Código A.2 Constante.m.

```
classdef Constante
%CONSTANTE Constantes y datos fijos que tienen lugar en la simulación
%  Sobre el modo, CBW, IG, tabla de tasas, tabla de MCS...

  properties (Constant)
    %% Estandares WiFi tipicos
    WIFI = {'b','g','n','ac'};
    WIFI_b = 1;
    WIFI_g = 2;
    WIFI_n = 3;
    WIFI_ac = 4;

    %% Modos posibles contemplados
    WIFI_ESTANDAR = {'b@20MHz', 'g@20MHz', 'n@20MHz', 'n@40MHz', 'ac@20MHz', '
        ac@40MHz', 'ac@80MHz', 'ac@160MHz', 'none'};

    WIFI_b_20MHz = 1;
    WIFI_g_20MHz = 2;
    WIFI_n_20MHz = 3;
    WIFI_n_40MHz = 4;
    WIFI_ac_20MHz = 5;
    WIFI_ac_40MHz = 6;
    WIFI_ac_80MHz = 7;
    WIFI_ac_160MHz = 8;
    WIFI_ninguno = 9;

    %% Frecuencias WiFi contempladas
    MODO_24_GHZ = 2.4;
    MODO_5_GHZ = 5;
```

```
%% Anchos de banda de canal posibles
CBW_20_MHZ = 20;
CBW_40_MHZ = 40;
CBW_80_MHZ = 80;
CBW_160_MHZ = 160;

%% Intervalos de guarda
IG_CORTO = 400;
IG_LARGO = 800;

%% Definición de tasas según MCS:
% 802.11b
TASAS_b = [1 2 5.5 11];
% 802.11g
TASAS_g=[6 9 12 18 24 36 48 54];
% 802.11n y 802.11ac
% C_BW =  |  20   |   40   |   80    |   160
% GI [ns] = |800 | 400| 800 | 400| 800 | 400 | 800 | 400
TASAS_n_ac = ...
[
6.5, 7.2, 13.5, 15, 29.3, 32.5, 58.5, 65; % MCS = 0
13, 14.4, 27, 30, 58.5, 65, 117, 130;    % MCS = 1
19.5, 21.7, 40.5, 45, 87.8, 97.5, 175.5, 195; % MCS = 2
26, 28.9, 54, 60, 117, 130, 234, 260;    % MCS = 3
39, 43.3, 81, 90, 175.5, 195, 351, 390;  % MCS = 4
52, 57.8, 108, 120, 234, 260, 468, 520;  % MCS = 5
58.5, 65, 121.5, 135, 263.3, 292.5, 526.5, 585; % MCS = 6
65, 72.2, 135, 150, 292.5, 325, 585, 650; % MCS = 7
78, 86.7, 162, 180, 351, 390, 702, 780;  % MCS = 8
86.7, 96.3, 180, 200, 390, 433.3, 780, 866.7]; % MCS = 9

%% Definición de MCS
%      b  g  n  n  ac ac ac ac
% [MHz] 20 20 20 40 20 40 80 160
MCS = ...
[
-1 -1 -1 -1 -1 -1 -1 -1;    % SNR = 1
-1 0  0  -1 0  -1 -1 -1;    % SNR = 2
-1 0  0  -1 0  -1 -1 -1;    % SNR = 3
0  1  0  -1 0  -1 -1 -1;    % SNR = 4
0  2  1  0  1  0  -1 -1;    % SNR = 5
0  2  1  0  1  0  -1 -1;    % SNR = 6
1  2  1  0  1  0  -1 -1;    % SNR = 7
1  2  1  1  1  1  0  -1;    % SNR = 8
1  3  2  1  2  1  0  -1;    % SNR = 9
1  3  2  1  2  1  0  -1;    % SNR = 10
2  4  3  1  3  1  1  0;     % SNR = 11
2  4  3  2  3  2  1  0;     % SNR = 12
2  4  3  2  3  2  1  0;     % SNR = 13
2  4  3  3  3  3  1  1;     % SNR = 14
2  5  4  3  4  3  2  1;     % SNR = 15
3  5  4  3  4  3  2  1;     % SNR = 16
3  5  4  3  4  3  3  1;     % SNR = 17
3  6  5  4  5  4  3  2;     % SNR = 18
3  6  5  4  5  4  3  2;     % SNR = 19
3  7  6  4  6  4  3  3;     % SNR = 20
```

```
    3   7   6   5   6   5   4   3;      % SNR = 21
    3   7   6   5   6   5   4   3;      % SNR = 22
    3   7   6   6   6   6   4   3;      % SNR = 23
    3   7   6   6   6   6   5   4;      % SNR = 24
    3   7   7   6   7   6   5   4;      % SNR = 25
    3   7   7   6   7   6   6   5;      % SNR = 26
    3   7   7   7   7   7   6   5;      % SNR = 27
    3   7   7   7   7   7   6   5;      % SNR = 28
    3   7   7   7   8   7   6   6;      % SNR = 29
    3   7   7   7   8   7   6   6;      % SNR = 30
    3   7   7   7   9   7   7   6;      % SNR = 31
    3   7   7   7   9   8   7   6;      % SNR = 32
    3   7   7   7   9   8   7   6;      % SNR = 33
    3   7   7   7   9   9   7   7;      % SNR = 34
    3   7   7   7   9   9   8   7;      % SNR = 35
    3   7   7   7   9   9   8   7;      % SNR = 36
    3   7   7   7   9   9   9   7;      % SNR = 37
    3   7   7   7   9   9   9   8;      % SNR = 38
    3   7   7   7   9   9   9   8;      % SNR = 39
    3   7   7   7   9   9   9   9];     % SNR = 40

    %% Definición de rangos en la matriz MCS
    % Permite usar una máscara para buscar valores
    MCS_RANGO_24_GHZ = 1:4;
    MCS_RANGO_5_GHZ = 3:8;
    MCS_RANGO_CBW_20_MHZ = [1 2 3 5];
    MCS_RANGO_CBW_40_MHZ = [4 6];
    MCS_RANGO_CBW_80_MHZ = 7;
    MCS_RANGO_CBW_160_MHZ = 8;
    MCS_RANGO_b = 1;
    MCS_RANGO_g = 2;
    MCS_RANGO_n = 3:4;
    MCS_RANGO_ac = 5:8;

    %% Constantes para la escritura del fichero escenario.xml (formato NS3)
    PHY_b={'DsssRate1Mbps', 'DsssRate2Mbps', 'DsssRate5_5Mbps', 'DsssRate11Mbps
        '};
    PHY_g={'ErpOfdmRate6Mbps', 'ErpOfdmRate9Mbps', 'ErpOfdmRate12Mbps', '
        ErpOfdmRate18Mbps', 'ErpOfdmRate24Mbps', 'ErpOfdmRate36Mbps', '
        ErpOfdmRate48Mbps', 'ErpOfdmRate54Mbps'};
    PHY_n={'HtMcs0', 'HtMcs1', 'HtMcs2', 'HtMcs3', 'HtMcs4', 'HtMcs5', 'HtMcs6',
         'HtMcs7'}
    PHY_ac={'VhtMcs0', 'VhtMcs1', 'VhtMcs2', 'VhtMcs3', 'VhtMcs4', 'VhtMcs5', '
        VhtMcs6', 'VhtMcs7', 'VhtMcs8', 'VhtMcs9'}
  end

end
```

A.2 Definición de scripts

A continuación se presentan los scripts utilizados en la herramienta MATLAB.

Código A.3 configura.m.

```
% CONFIGURA Script que permite parametrizar la simulación

%% PARÁMETROS DEL TABLERO (USUARIOS):
% Largo (Eje Y)
L = 200;
% Ancho (Eje X)
W = 400;
% Precisión de la rejilla en metros (para los usuarios)
paso_real = 0.5;

%% PARÁMETROS DEL TABLERO (DRONES):
% Distancia entre drones en metros (definición de los puntos X-Y a probar)
xy_step = 50;
% Distancia entre drones en metros (definición de los puntos Z a probar)
z_step = 50;
min_altura = 10;
max_altura = 100;

%% PARÁMETROS DE SOLUCIÓN:
% Número de usuarios a repartir en el tablero
num_personas = 100;
% Número de usuarios que hablan
num_personas_activas = 100;
% Función objetivo a maximizar
f_FO = @(minimo_rx,media_rx) 10*minimo_rx+media_rx;
% Porcentaje de usuarios que deben cumplir requisitos de SNR en %
porcentaje_cubierto = 95;
% Número máximo de drones a probar
max_drones = 5;

%% PARÁMETROS DE RADIOPROPAGACIÓN:
% Altura de las antenas receptoras (usuarios) en metros
hr = 1.5;

% Directividad mediante dipolo corto
D_max_dB = 4.5;
D_dB = @(theta) 10*log10(10^(D_max_dB/10)*cos(deg2rad(theta)).^2);

% Función de pérdidas
exponente_propagacion = 3;
p_loss = @(f,d,g,ht,hr) -147.55 - g + 10*exponente_propagacion*log10(d) + 20*
    log10(f);

% Potencia en dBm transmitida por los drones
p_tx = 20;

% En dB, a la hora de elegir tasa física, elegimos como si llegasen menos
% dBs
margen = 3;

% SNR mínima que deberá recibir cada usuario (debe ser > margen)
snr_min = 10;

% Modo de TX:
% Constante.MODO_24_GHZ
% Constante.MODO_5_GHZ
modo = Constante.MODO_24_GHZ;
```

```
% Número de antenas del transmisor y receptor (en cuanto a flujos
% espaciales se refiere)
n_antenas = 1; % (sólo funcional 1 flujo espacial en código NS3, cuidado!)

% Intervalo de guarda (400ns/800ns) (solo aplica en 802.11n y 802.11ac):
% Constante.IG_CORTO
% Constante.IG_LARGO
intervalo_guarda = Constante.IG_CORTO;

% Estandar a utilizar -> b/g/n/ac
% Constante.WIFI_b -> 2.4 GHz
% Constante.WIFI_g -> 2.4 GHz
% Constante.WIFI_n -> 2.4/5GHz
% Constante.WIFI_ac -> 5GHz
estandar = Constante.WIFI_n;

% Ancho de banda de canal en MHz -> 20/40/80/160
% Constante.CBW_20_MHZ -> Solo para b/g/n/ac
% Constante.CBW_40_MHZ -> Solo para n/ac
% Constante.CBW_80_MHZ -> Solo para ac
% Constante.CBW_160_MHZ -> -> Solo para ac
CBW = Constante.CBW_20_MHZ;

% Ruido térmico (depende del ancho de banda del canal)
KTBF = -174+10*log10(CBW*10^6);
```

Código A.4 simula.m.

```
% SIMULA Script que simula el despliegue de drones y el cálculo de cobertura

%% Preparación del entorno
close all
clear

%% Carga de datos de entrada
configura;

%% Preparación del escenario
% Escala de la matriz de los usuarios
x_personas = 0:paso_real:W;
y_personas = 0:paso_real:L;

% Escala de la matriz de los drones
x_dron = 0:xy_step:W;                       % Diferentes índices de x a probar
     de los tx
y_dron = 0:xy_step:L;                       % Diferentes índices de y a probar
     de los tx
z_dron = min_altura:z_step:max_altura;      % Diferentes índices de z a probar
     de los tx

% Diferente número de drones a probar
n_drones = 1:max_drones;

switch modo
  case Constante.MODO_24_GHZ
```

```
    % Modo 2.4 GHz
    f = 2.4e9;
  case Constante.MODO_5_GHZ
    % Modo 5 GHz
    f = 5e9;
  otherwise
    error('Modo incorrecto, especifique un valor entre 1 (2.4GHz) y 2 (5GHz)');
end

% Variables para almacenar la solución
USUARIOS = zeros(num_personas, 10);
ESCENARIO = zeros(1, 7);

% Cargo la configuración en la matriz ESCENARIO:
ESCENARIO(1) = W;
ESCENARIO(2) = L;
ESCENARIO(3) = num_personas;
ESCENARIO(4) = num_personas_activas;
ESCENARIO(5) = modo;
ESCENARIO(6) = estandar;
ESCENARIO(7) = CBW;
ESCENARIO(8) = intervalo_guarda;
ESCENARIO(9) = p_tx;
ESCENARIO(10) = exponente_propagacion;

%% Elaboración del escenario
% Iniciamos el editor_escenario
% Supongamos W = L y separacion de paso_real en cuadricula
editor_escenario = Escenario(W, L, paso_real);

% Creamos la función masa de probabilidad de una distribución uniforme 2D
fu = ones([(L/paso_real)+1 (W/paso_real)+1]);
fu = fu/(((L/paso_real)+1)*((W/paso_real)+1));

% Añadimos los puntos al editor_escenario siguiendo la distribución generada
fprintf('\nDistribuyendo los usuarios en el tablero\n\n')
editor_escenario.addPoints(fu, num_personas);

% Obtenemos la matriz que utilizaremos para simular
escenario_matriz = editor_escenario.getMatrix;

% Representación del escenario generado
escenario_coordenadas = editor_escenario.getCoordinates;
figure(1000)
subplot(2,2,1)
dibuja_escenario(W, L, escenario_coordenadas, paso_real, x_dron, y_dron, hr);
drawnow

%% Búsqueda de una solución
% Buscamos una solución para cada número de drones, empezando por 1 y
% terminando en el número máximo de drones a probar
for n = n_drones

  fprintf('Simulando con %d drones\n', n)
  %% Buscamos una solución para el escenario dado y el número de drones dado
  [SOLUCIONES_POSIBLES, DRONES, SNR, EMPAREJAMIENTO, GANANCIA] = busca_solucion
      (escenario_matriz, n);
```

```
if ~isnan(DRONES)
  %% Si hay solución, la imprimimos por pantalla construyendo una tabla
  fprintf('\tEncontrada la mejor solución\n\n')

  Dron = (1:n)';
  x = DRONES(:,1);
  y = DRONES(:,2);
  z = DRONES(:,3);

  Tabla_drones = table(Dron, x, y, z);
  disp(Tabla_drones);

  %% Dibujamos la solución encontrada
  figure(1000)
  subplot(2,2,2)
  dibuja_solucion(W, L, escenario_coordenadas, DRONES, paso_real, x_dron,
      y_dron, hr);
  subplot(2,2,3)
  dibuja_snr(SNR, x_personas, y_personas, x_dron, y_dron);
  subplot(2,2,4)
  dibuja_emparejamiento(EMPAREJAMIENTO, x_personas, y_personas, x_dron,
      y_dron);
  drawnow

  %% Traspongo las coordenadas para que tenga la forma que yo quiero
  coord_personas = escenario_coordenadas';

  % Variable temporal, para trabajar con cadenas
  estandar_cell = cell(1,num_personas);

  %% Para cada persona, vamos a rellenar una fila de la matriz PERSONAS
  for usuario=1:num_personas
    % Posición normalizada de la persona que tratamos
    x = coord_personas(usuario,1);
    y = coord_personas(usuario,2);
    % Posición escalada de la persona que tratamos
    USUARIOS(usuario,1) = x*paso_real;
    USUARIOS(usuario,2) = y*paso_real;
    USUARIOS(usuario,3) = hr;
    % SNR de la persona
    USUARIOS(usuario,4) = SNR(y+1, x+1);
    % Dron emparejado con la persona (si es negativo, no cumple
    % especificación de SNR)
    USUARIOS(usuario,5) = EMPAREJAMIENTO(y+1, x+1);
    % Evaluamos la tasa y la tecnología utilizada
    [USUARIOS(usuario,6), USUARIOS(usuario, 8), USUARIOS(usuario,7), USUARIOS
        (usuario,10)] = comprueba_tasa(USUARIOS(usuario,4) - margen, modo,
        intervalo_guarda, n_antenas, CBW, estandar);
    estandar_cell{usuario} = Constante.WIFI_ESTANDAR{USUARIOS(usuario, 8)};
    USUARIOS(usuario,9) = GANANCIA(y+1, x+1);
  end

  %% Imprimimos por pantalla construyendo una tabla:
  Usuario = (1:num_personas)';
  x = USUARIOS(:,1);
  y = USUARIOS(:,2);
```

```
    z = USUARIOS(:,3);
    Cumple_SNR = USUARIOS(:,4)>snr_min;
    Dron_asignado = abs(USUARIOS(:,5));
    Tecnologia = estandar_cell';
    Frecuencia_GHz = USUARIOS(:,6)*(10^-9);
    Tasa_Mbps = USUARIOS(:,7);

    Tabla_usuarios = table(Usuario, x, y, z, Cumple_SNR, Dron_asignado,
        Tecnologia, Frecuencia_GHz, Tasa_Mbps);
    disp(Tabla_usuarios)

    USUARIOS_ACTIVOS = randperm(num_personas, num_personas_activas);

    %% Escribimos salida a un fichero XML
    escribe_salida(ESCENARIO, DRONES, USUARIOS, USUARIOS_ACTIVOS)

    %% No iteramos más, ya que hemos encontrado la mejor solución
    break;
  else
    % Si no hubiera solución...
    fprintf('\tNo se han encontrado soluciones\n\n');
  end
end
```

A.3 Definición de funciones

A continuación se presentan las funciones utilizadas en la herramienta MATLAB.

Código A.5 busca_solucion.m.

```
function [todas_soluciones, mejor_solucion, snr, emparejamiento, ganancia_dir]
    = busca_solucion(escenario_matriz, num_drones)

%% Capturo variables globales que necesito del fichero config.m
xy_step = evalin('base', 'xy_step');
z_step = evalin('base', 'z_step');
f_FO = evalin('base', 'f_FO');
L = evalin('base', 'L');
W = evalin('base', 'W');
x_dron = evalin('base', 'x_dron');
y_dron = evalin('base', 'y_dron');
z_dron = evalin('base', 'z_dron');
min_altura = evalin('base', 'min_altura');

%% Declaro algunas variables extra:
% Almacena la posición de los drones que estamos utilizando
% Se guarda como indice (número entero único) y se convertirá más
% adelante en un subíndice del tipo [x y z]
posicionamiento = zeros([1 num_drones]);

% Almacena todas las soluciones encontradas (en [x y z]):
todas_soluciones = zeros([num_drones, 3]);
snr = -inf;
```

```
emparejamiento = -inf;
ganancia_dir = -inf;

% Número total de posiciones a probar:
nPosiciones = length(x_dron) * length(y_dron) * length(z_dron);
max_iteraciones = nchoosek(nPosiciones, num_drones);
n_iteraciones = 0;
% Mejor solución encontrada (en [x y z]):
% Lo inicializo a inf*cero para que sea una matriz de NaN y poder utilizar
% función isNaN más adelante para comprobar si hay o no solución
mejor_solucion = inf*zeros([num_drones, 3]);
% Variables auxiliares:
num_soluciones = 0;
mejorFO = -inf;

% Llamamos a la función recursiva, le decimos el número de drones a
% estudiar, el número total de posiciones a probar, la posición inicial
% (1), y el drone inicial (1).
fprintf('\\tExisten %d posibilidades para %d drones...', max_iteraciones,
    num_drones);

nItera(num_drones, nPosiciones, 1, 1);

fprintf(': %d posibilidades probadas\n', n_iteraciones);

  %% Función recursiva
  function nItera(nDrones, nPosiciones, pos_ini, dron)
    for pos_act=pos_ini:nPosiciones
      posicionamiento(dron) = pos_act;
      if dron==nDrones
        calcula(posicionamiento, escenario_matriz);
      else
        nItera(nDrones, nPosiciones, pos_act + 1, dron + 1);
      end
    end
  end

  %% Función que estudia una posición dada
  function calcula(posicionamiento, escenario)

    n_iteraciones = n_iteraciones+1;

    [x,y,z] = ind2sub([length(x_dron), length(y_dron), length(z_dron)],
        posicionamiento');
    posicion_a_probar = [(x-1)*xy_step (y-1)*xy_step (z*z_step+min_altura)];

    [valida, Prx, snr_temp, emparejamiento_temp, g_temp] = comprueba_cobertura(
        escenario, posicion_a_probar);

    if (valida)
      num_soluciones=num_soluciones+1;
      todas_soluciones(:,:,num_soluciones)=posicion_a_probar;

      mediaPrx=mean(Prx(escenario==1));
      minPrx=min(Prx(escenario==1));

      FO = f_FO(minPrx,mediaPrx);
```

```
    if FO > mejorFO
      mejor_solucion=posicion_a_probar;
      mejorFO=FO;
      snr = snr_temp;
      emparejamiento = emparejamiento_temp;
      ganancia_dir = g_temp;

      figure(1000)
      subplot(2,2,2)
      scatter3(mejor_solucion(:,1), mejor_solucion(:,2),mejor_solucion(:,3), '
          x');
      axis([0 W 0 L])
      xlabel('Eje X [m]')
      ylabel('Eje Y [m]')
      zlabel('Altura [m]')
      set(gca,'xtick',x_dron)
      set(gca,'ytick',y_dron)
      title('Solución temporal')
      grid on

      drawnow
    end
  end
 end
end
```

Código A.6 comprueba_cobertura.m.

```
function [valido, Prx, SNR, dron_emparejado, G_dir] = comprueba_cobertura(
    escenario, posiciones_drones)

%% Capturamos los parámetros de configura.m
% No tengo claro si es mejor pasarlo como parámetro, pero son demasiados y
% he preferidos capturarlos directamente del workspace
hr = evalin('base', 'hr');
p_loss = evalin('base', 'p_loss');
p_tx = evalin('base', 'p_tx');
snr_min = evalin('base', 'snr_min');
f = evalin('base', 'f');
KTBF = evalin('base', 'KTBF');
x_personas = evalin('base', 'x_personas');
y_personas = evalin('base', 'y_personas');
num_personas = evalin('base', 'num_personas');
porcentaje_cubierto = evalin('base', 'porcentaje_cubierto');
D_dB = evalin('base', 'D_dB');

%% Procedimiento
% Matrices auxiliares para el cálculo de distancias
x_array = repmat(x_personas, [length(y_personas) 1]);
y_array = repmat(y_personas, [length(x_personas) 1])';

% Inicialización de la matriz de emparejamiento, inicialmente todos están
% emparejados al dron 1
dron_emparejado = ones([length(y_personas) length(x_personas)]);
G_dir = zeros([length(y_personas) length(x_personas)]);
```

```
distancia_final = ones([length(y_personas) length(x_personas)]);
angulo_final = ones([length(y_personas) length(x_personas)]);

% Cálculo de potencias recibidas
for i=1:length(posiciones_drones(:,1))
  % Calculo la matriz de distancias al dron i
  x_rel=(x_array-posiciones_drones(i,1));
  y_rel=(y_array-posiciones_drones(i,2));
  z_rel=(hr-posiciones_drones(i,3));

  xy_rel = x_rel.^2 + y_rel.^2;

  distancia = sqrt((xy_rel + (z_rel)^2));
  angulo = rad2deg(real(acos(abs(z_rel)./sqrt(xy_rel))));
  G_dir = D_dB(angulo);

  if i==1
    % Calculo la potencia recibida desde el dron 1
    Prx = p_tx-p_loss(f, distancia, G_dir, posiciones_drones(i,3), hr);
    angulo_final = angulo;
    distancia_final = distancia;
  else
    % Calculo la potencia recibida desde el dron i
    Prx_nuevo = p_tx-p_loss(f, distancia, G_dir, posiciones_drones(i,3), hr);
    % Si mejora la potencia recibida, emparejo con el nuevo dron
    zona_mejorada = Prx_nuevo>Prx;
    dron_emparejado(zona_mejorada)=i;
    % La potencia recibida será la máxima de entre los drones
    % desplegados
    Prx = max(Prx, Prx_nuevo);
    angulo_final(zona_mejorada) = angulo(zona_mejorada);
    distancia_final(zona_mejorada) = distancia(zona_mejorada);
  end
end

% Calculo SNR a partir de KTBF (en todas las posiciones)
SNR = Prx-KTBF;

if length(find(SNR(escenario==1)<snr_min))>(1-porcentaje_cubierto/100)*
    num_personas
  % No hacemos nada
  valido = 0;
else
  % Dibujamos posible solucion
  valido = 1;
  % Hago negativos los emparejamientos que no cumplen especificación de SNR
  dron_emparejado(SNR<snr_min)=dron_emparejado(SNR<snr_min)*-1;
end

end
```

Código A.7 comprueba_tasa.m.

```
function [FRECUENCIA, ESTANDAR_INDEX, TASA, MCS] = comprueba_tasa( SNR_dB ,
    frec, GI, flujos_espaciales, CBW, estandar)
%COMPRUEBA_VELOCIDAD Calcula la tasa de sincronización y modo dada la SNR
```

```
%  Modo puede ser Constante.MODO_24_GHZ para 2.4GHz y Constante.MODO_5_GHZ
    para 5GHz
%  GI puede ser Constante.IG_CORTO y Constante.IG_LARGO

%% Truncamos la SNR
% La tabla sólo va de 1 a 40 dB
if (SNR_dB>40)
  SNR_entero = 40;
elseif (SNR_dB < 1)
  SNR_entero = 1; % Lo ponemos a 1 porque realmente con SNR = 1 la tasa es 0
      Mbit/s
else
  SNR_entero = floor(SNR_dB);
end

if SNR_entero > 0
  %% Truncamos las antenas si se pasan de lo posible
  switch frec
    case Constante.MODO_24_GHZ
    if (flujos_espaciales > 4)
      flujos_espaciales = 4;
    end
    FRECUENCIA = 2.4e9;
    rango_banda = Constante.MCS_RANGO_24_GHZ;
    case Constante.MODO_5_GHZ
    if (flujos_espaciales > 8)
      flujos_espaciales = 8;
    end
    FRECUENCIA = 5e9;
    rango_banda = Constante.MCS_RANGO_5_GHZ;
  end

  %% Elijo qué columnas de la tabla cumplen la CBW
  switch CBW
    case Constante.CBW_20_MHZ
      rango_cbw = Constante.MCS_RANGO_CBW_20_MHZ;
    case Constante.CBW_40_MHZ
      rango_cbw = Constante.MCS_RANGO_CBW_40_MHZ;
    case Constante.CBW_80_MHZ
      rango_cbw = Constante.MCS_RANGO_CBW_80_MHZ;
    case Constante.CBW_160_MHZ
      rango_cbw = Constante.MCS_RANGO_CBW_160_MHZ;
    otherwise
      error('CBW incorrecto')
  end

  %% Elijo qué columnas de la tabla cumplen el estándar
  switch estandar
    case Constante.WIFI_b
      rango_estandar = Constante.MCS_RANGO_b;
    case Constante.WIFI_g
      rango_estandar = Constante.MCS_RANGO_g;
    case Constante.WIFI_n
      rango_estandar = Constante.MCS_RANGO_n;
    case Constante.WIFI_ac
      rango_estandar = Constante.MCS_RANGO_ac;
  end
```

```
  %% Construimos tabla de tasas
  TASAS = zeros(size(Constante.MCS));
  for col=1:length(Constante.MCS(1,:))
    if col == 1
      matriz_tasas = Constante.TASAS_b;
    elseif col == 2
      matriz_tasas = Constante.TASAS_g;
    elseif col == 3
      matriz_tasas = flujos_espaciales*Constante.TASAS_n_ac(:, 2-(GI/Constante.
          IG_CORTO-1));
    elseif col == 4
      matriz_tasas = flujos_espaciales*Constante.TASAS_n_ac(:, 4-(GI/Constante.
          IG_CORTO-1));
    elseif col == 5
      matriz_tasas = flujos_espaciales*Constante.TASAS_n_ac(:, 2-(GI/Constante.
          IG_CORTO-1));
    elseif col == 6
      matriz_tasas = flujos_espaciales*Constante.TASAS_n_ac(:, 4-(GI/Constante.
          IG_CORTO-1));
    elseif col == 7
      matriz_tasas = flujos_espaciales*Constante.TASAS_n_ac(:, 6-(GI/Constante.
          IG_CORTO-1));
    elseif col == 8
      matriz_tasas = flujos_espaciales*Constante.TASAS_n_ac(:, 8-(GI/Constante.
          IG_CORTO-1));
    end
    columna = Constante.MCS(:,col)+1;
    columna_tasas = zeros(40, 1);
    for mcs_val=1:length(matriz_tasas)
      columna_tasas(columna==mcs_val) = matriz_tasas(mcs_val);
    end
    TASAS(:,col)=columna_tasas;
  end

  %% Aplico la máscara con las columnas posibles
  rango = intersect(rango_banda, rango_cbw);
  rango = intersect(rango, rango_estandar);
  mascara = zeros(1,8);
  mascara(rango)=1;
  mascara = repmat(mascara, [40, 1]);

  tabla_tasas = TASAS;
  tabla_tasas(~mascara) = -1;

  %% Calculo la tasa en la tabla de tasas
  [TASA, columna] = max(tabla_tasas(SNR_entero,:));
  MCS = Constante.MCS(SNR_entero, columna);
  index = find(TASA==tabla_tasas(SNR_entero,:), 1, 'last');

else
  %% SNR negativa, difícilmente va a tener conexión
  TASA = 0;
  MCS = 0;
  index = Constante.WIFI_ninguno;
end
```

```
ESTANDAR_INDEX = index;

end
```

A.4 Relación entre SNR y MCS

A continuación se adjuntan las tablas utilizadas para relacionar la SNR y MCS [34, 35].

Tabla A.1 Relación SNR y MCS en 802.11b.

IEEE 802.11b	20MHz
MCS	Min SNR (dB)
0	4
1	7
2	11
3	16

Tabla A.2 Relación SNR y MCS en 802.11b.

IEEE 802.11g	20MHz
MCS	Min SNR (dB)
0	2
1	4
2	5
3	9
4	11
5	15
6	18
7	20

Tabla A.3 Relación SNR y MCS en 802.11n.

			20MHz				40MHz			
HT MCS Index	**Modulation**	**Coding**	**Data Rate (Mbps)** GI = 800ns	GI = 400ns	**Min. SNR (dBm)**	**Receive Sensitivity (RSSI)**	**Data Rate (Mbps)** GI = 800ns	GI = 400ns	**Min. SNR (dBm)**	**Receive Sensitivity (RSSI)**
1 Spatial Stream										
0	BPSK	1/2	6,5	7,2	2	-82	13,5	15	5	-79
1	QPSK	1/2	13	14,4	5	-79	27	30	8	-76
2	QPSK	3/4	19,5	21,7	9	-77	40,5	45	12	-74
3	16-QAM	1/2	26	28,9	11	-74	54	60	14	-71
4	16-QAM	3/4	39	43,3	15	-70	81	90	18	-67
5	64-QAM	2/3	52	57,8	18	-66	108	120	21	-63
6	64-QAM	3/4	58,5	65	20	-65	121,5	135	23	-62
7	64-QAM	5/6	65	72,2	25	-64	135	150	28	-61
2 Spatial Streams										
8	BPSK	1/2	13	14,4	2	-82	27	30	5	-79
9	QPSK	1/2	26	28,9	5	-79	54	60	8	-76
10	QPSK	3/4	39	43,3	9	-77	81	90	12	-74
11	16-QAM	1/2	52	57,8	11	-74	108	120	14	-71
12	16-QAM	3/4	78	86,7	15	-70	162	180	18	-67
13	64-QAM	2/3	104	115,6	18	-66	216	240	21	-63
14	64-QAM	3/4	117	130,3	20	-65	243	270	23	-62
15	64-QAM	5/6	130	144,4	25	-64	270	300	28	-61
3 Spatial Streams										
16	BPSK	1/2	19,5	21,7	2	-82	40,5	45	5	-79
17	QPSK	1/2	39	43,3	5	-79	81	90	8	-76
18	QPSK	3/4	58,5	65	9	-77	121,5	135	12	-74
19	16-QAM	1/2	78	86,7	11	-74	162	180	14	-71
20	16-QAM	3/4	117	130	15	-70	243	270	18	-67
21	64-QAM	2/3	156	173,3	18	-66	324	360	21	-63
22	64-QAM	3/4	175,5	195	20	-65	364,5	405	23	-62
23	64-QAM	5/6	195	216,7	25	-64	405	450	28	-61
4 Spatial Streams										
24	BPSK	1/2	26	28,9	2	-82	54	60	5	-79
25	QPSK	1/2	52	57,8	5	-79	108	120	8	-76
26	QPSK	3/4	78	86,7	9	-77	162	180	12	-74
27	16-QAM	1/2	104	115,6	11	-74	216	240	14	-71
28	16-QAM	3/4	156	173,3	15	-70	324	360	18	-67
29	64-QAM	2/3	208	231,1	18	-66	432	480	21	-63
30	64-QAM	3/4	234	260	20	-65	486	540	23	-62
31	64-QAM	5/6	260	288,9	25	-64	540	600	28	-61

Tabla A.4 Relación SNR y MCS en 802.11ac (20 y 40 MHz).

VHT MCS Index	Modulation	Coding	20MHz				40MHz			
			Data Rate (Mbps)		Min. SNR (dB)	Receive Sensitivity (RSSI)	Data Rate (Mbps)		Min. SNR (dB)	Receive Sensitivity (RSSI)
			GI = 800ns	GI = 400ns			GI = 800ns	GI = 400ns		
1 Spatial Stream										
0	BPSK	1/2	6,5	7,2	2	-82	13,5	15	5	-79
1	QPSK	1/2	13	14,4	5	-79	27	30	8	-76
2	QPSK	3/4	19,5	21,7	9	-77	40,5	45	12	-74
3	16-QAM	1/2	26	28,9	11	-74	54	60	14	-71
4	16-QAM	3/4	39	43,3	15	-70	81	90	18	-67
5	64-QAM	2/3	52	57,8	18	-66	108	120	21	-63
6	64-QAM	3/4	58,5	65	20	-65	121,5	135	23	-62
7	64-QAM	5/6	65	72,2	25	-64	135	150	28	-61
8	256-QAM	3/4	78	86,7	29	-59	162	180	32	-56
9	256-QAM	5/6			31	-57	180	200	34	-54
2 Spatial Stream										
0	BPSK	1/2	13	14,4	2	-82	27	30	5	-79
1	QPSK	1/2	26	28,9	5	-79	54	60	8	-76
2	QPSK	3/4	39	43,3	9	-77	81	90	12	-74
3	16-QAM	1/2	52	57,8	11	-74	108	120	14	-71
4	16-QAM	3/4	78	86,7	15	-70	162	180	18	-67
5	64-QAM	2/3	104	115,6	18	-66	216	240	21	-63
6	64-QAM	3/4	117	130,3	20	-65	243	270	23	-62
7	64-QAM	5/6	130	144,4	25	-64	270	300	28	-61
8	256-QAM	3/4	156	173,3	29	-59	324	360	32	-56
9	256-QAM	5/6			31	-57	360	400	34	-54
3 Spatial Stream										
0	BPSK	1/2	19,5	21,7	2	-82	40,5	45	5	-79
1	QPSK	1/2	39	43,3	5	-79	81	90	8	-76
2	QPSK	3/4	58,5	65	9	-77	121,5	135	12	-74
3	16-QAM	1/2	78	86,7	11	-74	162	180	14	-71
4	16-QAM	3/4	117	130	15	-70	243	270	18	-67
5	64-QAM	2/3	156	173,3	18	-66	324	360	21	-63
6	64-QAM	3/4	175,5	195	20	-65	364,5	405	23	-62
7	64-QAM	5/6	195	216,7	25	-64	405	450	28	-61
8	256-QAM	3/4	234	260	29	-59	486	540	32	-56
9	256-QAM	5/6	260	288,9	31	-57	540	600	34	-54
4 Spatial Stream										
0	BPSK	1/2	26	28,9	2	-82	54	60	5	-79
1	QPSK	1/2	52	57,8	5	-79	108	120	8	-76
2	QPSK	3/4	78	86,7	9	-77	162	180	12	-74
3	16-QAM	1/2	104	115,6	11	-74	216	240	14	-71
4	16-QAM	3/4	156	173,3	15	-70	324	360	18	-67
5	64-QAM	2/3	208	231,1	18	-66	432	480	21	-63
6	64-QAM	3/4	234	260	20	-65	486	540	23	-62
7	64-QAM	5/6	260	288,9	25	-64	540	600	28	-61
8	256-QAM	3/4	312	346,7	29	-59	648	720	32	-56
9	256-QAM	5/6			31	-57	720	800	34	-54

Tabla A.5 Relación SNR y MCS en 802.11ac (80 y 160 MHz) [Parte 1].

VHT MCS Index	Modulation	Coding	80MHz				160MHz			
			Data Rate (Mbps)		Min. SNR (dB)	Receive Sensitivity (RSSI)	Data Rate (Mbps)		Min. SNR (dB)	Receive Sensitivity (RSSI)
			GI = 800ns	GI = 400ns			GI = 800ns	GI = 400ns		
1 Spatial Stream										
0	BPSK	1/2	29,3	32,5	8	-76	58,5	65	11	-73
1	QPSK	1/2	58,5	65	11	-73	117	130	14	-70
2	QPSK	3/4	87,8	97,5	15	-71	175,5	195	18	-68
3	16-QAM	1/2	117	130	17	-68	234	260	20	-65
4	16-QAM	3/4	175,5	195	21	-64	351	390	24	-61
5	64-QAM	2/3	234	260	24	-60	468	520	27	-57
6	64-QAM	3/4	263,3	292,5	26	-59	526,5	585	29	-56
7	64-QAM	5/6	292,5	325	31	-58	585	650	34	-55
8	256-QAM	3/4	351	390	35	-53	702	780	38	-50
9	256-QAM	5/6	390	433,3	37	-51	780	866,7	40	-48
2 Spatial Stream										
0	BPSK	1/2	58,5	65	8	-76	117	130	11	-73
1	QPSK	1/2	117	130	11	-73	234	260	14	-70
2	QPSK	3/4	175,5	195	15	-71	351	390	18	-68
3	16-QAM	1/2	234	260	17	-68	468	520	20	-65
4	16-QAM	3/4	351	390	21	-64	702	780	24	-61
5	64-QAM	2/3	468	520	24	-60	936	1040	27	-57
6	64-QAM	3/4	526,5	585	26	-59	1053	1170	29	-56
7	64-QAM	5/6	585	650	31	-58	1170	1300	34	-55
8	256-QAM	3/4	702	780	35	-53	1404	1560	38	-50
9	256-QAM	5/6	780	866,7	37	-51	1560	1733,3	40	-48
3 Spatial Stream										
0	BPSK	1/2	87,8	97,5	8	-76	175,5	195	11	-73
1	QPSK	1/2	175,5	195	11	-73	351	390	14	-70
2	QPSK	3/4	263,3	292,5	15	-71	526,5	585	18	-68
3	16-QAM	1/2	351	390	17	-68	702	780	20	-65
4	16-QAM	3/4	526,5	585	21	-64	1053	1170	24	-61
5	64-QAM	2/3	702	780	24	-60	1404	1560	27	-57
6	64-QAM	3/4			26	-59	1579,5	1755	29	-56
7	64-QAM	5/6	877,5	975	31	-58	1755	1950	34	-55
8	256-QAM	3/4	1053	1170	35	-53	2106	2340	38	-50
9	256-QAM	5/6	1170	1300	37	-51			40	-48
4 Spatial Stream										
0	BPSK	1/2	117	130	8	-76	234	260	11	-73
1	QPSK	1/2	234	260	11	-73	468	520	14	-70
2	QPSK	3/4	351	390	15	-71	702	780	18	-68
3	16-QAM	1/2	468	520	17	-68	936	1040	20	-65
4	16-QAM	3/4	702	780	21	-64	1404	1560	24	-61
5	64-QAM	2/3	936	1040	24	-60	1872	2080	27	-57
6	64-QAM	3/4	1053	1170	26	-59	2106	2340	29	-56
7	64-QAM	5/6	1170	1300	31	-58	2340	2600	34	-55
8	256-QAM	3/4	1404	1560	35	-53	2808	3120	38	-50
9	256-QAM	5/6	1560	1733,3	37	-51	3120	3466,7	40	-48

Tabla A.6 Relación SNR y MCS en 802.11ac (80 y 160 MHz) [Parte 2].

VHT MCS Index	Modulation	Coding	80MHz				160MHz			
			Data Rate (Mbps)		Min. SNR (dB)	Receive Sensitivity (RSSI)	Data Rate (Mbps)		Min. SNR (dB)	Receive Sensitivity (RSSI)
			GI = 800ns	GI = 400ns			GI = 800ns	GI = 400ns		
5 Spatial Stream										
0	BPSK	1/2	146,3	162,5	8	-76	292,5	325	11	-73
1	QPSK	1/2	292,5	325	11	-73	585	650	14	-70
2	QPSK	3/4	438,8	487,5	15	-71	877,5	975	18	-68
3	16-QAM	1/2	585	650	17	-68	1170	1300	20	-65
4	16-QAM	3/4	877,5	975	21	-64	1755	1950	24	-61
5	64-QAM	2/3	1170	1300	24	-60	2340	2600	27	-57
6	64-QAM	3/4	1316,3	1462,5	26	-59	2632,5	2925	29	-56
7	64-QAM	5/6	1462,5	1625	31	-58	2925	3250	34	-55
8	256-QAM	3/4	1755	1950	35	-53	3510	3900	38	-50
9	256-QAM	5/6	1950	2166,7	37	-51	3900	4333,3	40	-48
6 Spatial Stream										
0	BPSK	1/2	175,5	195	8	-76	351	390	11	-73
1	QPSK	1/2	351	390	11	-73	702	780	14	-70
2	QPSK	3/4	526,5	585	15	-71	1053	1170	18	-68
3	16-QAM	1/2	702	780	17	-68	1404	1560	20	-65
4	16-QAM	3/4	1053	1170	21	-64	2106	2340	24	-61
5	64-QAM	2/3	1404	1560	24	-60	2808	3120	27	-57
6	64-QAM	3/4	1579,5	1755	26	-59	3159	3510	29	-56
7	64-QAM	5/6	1755	1950	31	-58	3510	3900	34	-55
8	256-QAM	3/4	2106	2340	35	-53	4212	4680	38	-50
9	256-QAM	5/6			37	-51	4680	5200	40	-48
7 Spatial Stream										
0	BPSK	1/2	204,8	227,5	8	-76	409,5	455	11	-73
1	QPSK	1/2	409,5	455	11	-73	819	910	14	-70
2	QPSK	3/4	614,3	682,5	15	-71	1228,5	1365	18	-68
3	16-QAM	1/2	819	910	17	-68	1638	1820	20	-65
4	16-QAM	3/4	1228,5	1365	21	-64	2457	2730	24	-61
5	64-QAM	2/3	1638	1820	24	-60	3276	3640	27	-57
6	64-QAM	3/4			26	-59	3685,5	4095	29	-56
7	64-QAM	5/6	2047,5	2275	31	-58	4095	4550	34	-55
8	256-QAM	3/4	2457	2730	35	-53	4914	5460	38	-50
9	256-QAM	5/6	2730	3033,3	37	-51	5460	6066,7	40	-48
8 Spatial Stream										
0	BPSK	1/2	234	260	8	-76	468	520	11	-73
1	QPSK	1/2	468	520	11	-73	936	1040	14	-70
2	QPSK	3/4	702	780	15	-71	1404	1560	18	-68
3	16-QAM	1/2	936	1040	17	-68	1872	2080	20	-65
4	16-QAM	3/4	1404	1560	21	-64	2808	3120	24	-61
5	64-QAM	2/3	1872	2080	24	-60	3744	4160	27	-57
6	64-QAM	3/4	2106	2340	26	-59	4212	4680	29	-56
7	64-QAM	5/6	2340	2600	31	-58	4680	5200	34	-55
8	256-QAM	3/4	2808	3120	35	-53	5616	6240	38	-50
9	256-QAM	5/6	3120	3466,7	37	-51	6240	6933,3	40	-48

Apéndice B
NS3

B.1 Código de la solución

Se presentan a continuación los códigos utilizados en la herramienta de simulación.

Código B.1 simulador.cc.

```
// Para NS3:
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/spectrum-module.h"
#include "ns3/average.h"
#include "ns3/gnuplot.h"

// Para lectura de ficheros:
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <stdio.h>
#include <stdlib.h>

// Propios
#include "escenario.h"
#include "funciones.h"
#include "mapped-rate-wifi-manager.h"
#include "observador.h"

// Para NS3:
using namespace ns3;

// Para lectura de ficheros:
using namespace std;

NS_LOG_COMPONENT_DEFINE("Simulador");
```

```
// Main
int main (int argc, char ** argv){

  // Setup inicial del entorno de NS3
  NS_LOG_INFO("");
  GlobalValue::Bind("ChecksumEnabled", BooleanValue(true));
  Time::SetResolution (Time::NS);
  RngSeedManager::SetSeed (1);

  /* Declaración de variables del escenario */

  Escenario escenario;
  // Objeto de la clase Escenario, definida en escenario.h
  float W;
  // Ancho del escenario
  float L;
  // Largo del escenario
  float P_tx;
  // Potencia de transmisión en dBm
  float Exp_prop;
  // Exponente de propagación del medio
  float Frequency;
  // Frecuencia en GHz
  float ChannelWidth;
  // Ancho de banda del canal en MHz
  uint32_t NumDrones;
  // Número de drones a desplegar
  uint32_t NumUsers;
  // Número de usuarios a considerar
  uint32_t NumActiveUsers;
  // Número de usuarios activos
  bool ShortGuardEnabled;
  // True si IG corto activo, false en caso contrario (procede para 802.11n y
      802.11ac)
  std::string WifiPhyStandard;
  // Estándar 802.11 a utilizar, en "formato" NS3
  std::vector <uint16_t> activeUsers;
  // Vector de usuarios activos (identificadores)

  /* Declaración de variables de NS3 */

  // Contenedores, en cierto modo:
  NodeContainer drones, users, servers;
  // NodeContainers declarados; separados por drones, usuarios y servidores.
  NetDeviceContainer droneDevices, userDevices;
  // NetDevices declarados; Interfaces Wireless
  std::vector <NetDeviceContainer> wiredDevices;
  //Interfaces Wired (Elemento 0 corresponde a dron, 1 a servidor)
  Ipv4InterfaceContainer droneNodeInterface, userNodeInterface;
  // Interfaces de red;   Interfaces Wireless
  std::vector <Ipv4InterfaceContainer> wiredNodeInterface;
  //Interfaces Wired (análogo a NetDevices)
  ApplicationContainer apps_sink, apps_source;
  // Contenedores de aplicaciones; separados por aplicaciones sumidero y
      fuentes.
```

```
// Helpers:
PointToPointHelper p2p;
// Para instalar conexiones entre drones y su servidor asociado.
WifiHelper wifi;
// Para instalar conexiones Wifi entre usuarios y drones.
WifiMacHelper mac;
// Para instalar la capa MAC en Wifi.
SpectrumWifiPhyHelper phy;
// Para instalar la capa PHY en Wifi.
SpectrumChannelHelper channel;
// Para instalar el canal físico en Wifi.

MobilityHelper mobility;
// Para instalar la posición de los nodos.
InternetStackHelper stack;
// Para instalar la pila Internet en nodos.
Ipv4AddressHelper address;
// Para asignar direcciones a; interfaces Wireless.
std::vector <Ipv4AddressHelper> wiredAddress;
//interfaces Wired (Elemento 0 corresponde a interfaz del dron, 1 a la del
    servidor).
PacketSinkHelper sink ("ns3::UdpSocketFactory", Ipv4Address::GetAny ());
// Para instalar sumideros UDP.
UdpClientHelper udpSource (Ipv4Address::GetAny ());

// Punteros inteligentes:
Ptr<ListPositionAllocator> positionAlloc;
// Útil para definir posición.
Ptr <Mapa> mapa = CreateObject<Mapa>();
// Objeto de la clase Mapa definida en mapped-rate-wifi-manager.h, nos
    permite enviarle el mapeo Dir. Mac Destino - Modulación al
    MappedRateWifiManager.

/* Parámetros de la simulación */

// Parametrizables mediante línea de comandos:
uint32_t wifiQueue = 100;
bool mustTrace = true;

// Fijos:
uint16_t port = 9;
// Puerto para recibir tráfico.
uint16_t sendTime = 4;
// Tiempo de simulación, en segundos.
uint16_t rtpHeaderSize = 12;
uint16_t packetSize = 160;
// Tamaño (MTU) de paquetes.
double initOffset = 0.5;
// Tiempo de espera antes de empezar a transmitir (le damos tiempo a los Sink
     de iniciarse).
double finishOffset = 1.5;
// Tiempo de espera antes de finalizar la simulación (debe ser mayor que 1s)

/* Declaración de variables adicionales (contadores, auxiliares...) */

uint16_t actualChannel = 1;
uint16_t numChannels = 0;
```

```
uint16_t channelFrequency [14] = {2412, 2417, 2422, 2427, 2432, 2437, 2442,
    2447, 2452, 2457, 2462, 2467, 2472, 2484};
uint16_t spaceBetweenUsedChannels = 0;

bool onlyDownload = false;
bool allResults = false;

// Relacionados con la lectura del fichero:
xmlDocPtr doc;
xmlNodePtr root_element;

string file = "scratch/TFM/entrada/802.11n/1.xml";
string output = "scratch/TFM/salida/";

// StringStream que nos permite diseñar cadenas para utilizarlo como
    parámetro en ciertos métodos:
stringstream ss;

// NetDeviceContainer temporal que nos ayuda a reordenar los usuarios en el
    NetDeviceContainer original.
NetDeviceContainer userDevicesOrderedById;

// Captura de trazas Ascii:
//AsciiTraceHelper ascii;

// Objetos de la clase AddressValue, nos ayudará a definir orígenes y
    destinos de las aplicaciones:
AddressValue localAddress = AddressValue();
AddressValue remoteAddress = AddressValue();

// Enteros para hacer referencia a un usuario (ID) o el índice de un dron (
    ID_dron - 1):
uint16_t id_usuario;
uint16_t index_dron;

// Dejamos a punto una variable aleatoria uniforme para obtener valores
    cuando lo necesitemos (entre 0 y 1):
Ptr <UniformRandomVariable> uniformRandomVariable = CreateObject <
    UniformRandomVariable> ();
uniformRandomVariable->SetAttribute("Max", DoubleValue(1.0));
uniformRandomVariable->SetAttribute("Min", DoubleValue(0));

// Dejamos preparado un Double para lo que lo necesitemos más adelante:
double tempValue = 0;

/* Lectura de parámetros de entrada */

// Parámetros recogidos por cmd

CommandLine cmd;
cmd.AddValue ("queueSize", "Tamaño de cola del AP", wifiQueue);
cmd.AddValue ("inputFile", "Ruta al fichero de entrada", file);
cmd.AddValue ("outputDir", "Ruta al directorio de salida (acabado en /)",
    output);
cmd.AddValue ("simulationTime", "Tiempo de simulación", sendTime);
cmd.AddValue ("noUpload", "Sólo simular tráfico descendente", onlyDownload);
cmd.AddValue ("showAllResults", "Mostrar resultados por flujo", allResults);
```

```
cmd.Parse (argc,argv);

/* Lectura del fichero */

doc = xmlReadFile(file.c_str(), NULL, 0);
if (doc == NULL) {
NS_LOG_ERROR("Error abriendo el fichero.");
}

root_element = xmlDocGetRootElement(doc);

// Llamamos a una función recursiva declarada en funciones.h, a partir de un
    elemento raíz y un objeto Escenario, lo va leyendo y construyendo:
escenario = cargaEscenario(root_element, escenario);

// Liberamos:
xmlFreeDoc(doc);

/* Almacenamiento de parámetros del escenario */

Observador * observador = new Observador(packetSize, sendTime);

// Almacenamos el contenido del escenario en variables para tenerlas más a
    mano:
W = escenario.w;
L = escenario.l;
Frequency = escenario.frequency;
ChannelWidth = escenario.channelWidth;
NumDrones = escenario.numDrones;
NumUsers = escenario.numUsers;
NumActiveUsers = escenario.numActiveUsers;
ShortGuardEnabled = escenario.shortGuardEnabled;
WifiPhyStandard = escenario.wifiPhyStandard;
P_tx = escenario.pTx;
Exp_prop = escenario.expPropagation;

// Trazas de LOG:
NS_LOG_INFO(">> Encontrado escenario declarado en escenario.xml:");
NS_LOG_INFO(">>>> Dimensiones: " << W << "x" << L);
NS_LOG_INFO(">>>> Frecuencia: " << Frequency << " GHz");
NS_LOG_INFO(">>>> Estándar: " << WifiPhyStandard);
NS_LOG_INFO(">>>> Ancho de banda de canal: " << ChannelWidth << " MHz");
NS_LOG_INFO(">>>> Intervalo de guarda corto (true o false): " <<
    ShortGuardEnabled);
NS_LOG_INFO(">>>> " << NumDrones << " drones");
NS_LOG_INFO(">>>> " << NumUsers << " usuarios");
NS_LOG_INFO(">>>> " << NumActiveUsers << " usuarios activos");
NS_LOG_INFO("");

/* Aplicación de algunos parámetros por defecto (comunes a todos los nodos)
    */

// Fijamos por defecto el exponente de propagación del modelo según lo leído:
Config::SetDefault ("ns3::LogDistancePropagationLossModel::Exponent",
    DoubleValue (Exp_prop));
```

```
// Cambiamos el valor por defecto de la pérdida referencial a 1 metro si es
    2.4GHz, si no (se asume 5GHz) el valor por defecto ya es correcto.
if (Frequency == (float) 2.4){
  Config::SetDefault ("ns3::LogDistancePropagationLossModel::ReferenceLoss",
      DoubleValue (40.046));
}

// Solución para bug reconocido (Para SpectrumWifiPhy):
Config::SetDefault ("ns3::WifiPhy::CcaMode1Threshold", DoubleValue (-62.0));

/* Inicialización de algunos parámetros generales del escenario */

// Inicialización de nodos:
drones.Create(NumDrones);
users.Create(NumUsers);
servers.Create(NumDrones);

// Inicialización de direcciones:
// Las de WIFI pertenecerán todas a la misma subred
address.SetBase ("192.168.0.0", "255.255.0.0");

// Los enlaces punto a punto (entre dron y servidor), vendrán dadas por la
    dirección 10.10.num_dron.1 si es dron y 10.10.num_dron.2 el servidor
    asociado
for ( uint16_t cont = 0; cont < NumDrones; cont++){
  wiredAddress.push_back(Ipv4AddressHelper());
  ss.clear();
  ss.str("");
  ss << "10.10." << cont+1 << ".0";
  wiredAddress[cont].SetBase(Ipv4Address (ss.str().c_str()), "255.255.255.0")
      ;
}

// Inicialización de los enlaces punto a punto:
// Lo sobredimensiono adrede
p2p.SetDeviceAttribute ("DataRate", StringValue("1Gbps"));
p2p.SetChannelAttribute ("Delay", StringValue("0ms"));

// Inicialización del canal
channel.SetChannel("ns3::MultiModelSpectrumChannel");
channel.AddPropagationLoss("ns3::LogDistancePropagationLossModel");
channel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel");

// Inicialización de la capa física
phy.SetErrorRateModel ("ns3::NistErrorRateModel");
phy.Set ("ShortGuardEnabled", BooleanValue (ShortGuardEnabled));
phy.Set ("TxPowerStart", DoubleValue (P_tx));
phy.Set ("TxPowerEnd", DoubleValue (P_tx));
//phy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11_RADIO);

// Inicialización del estándar
if (!WifiPhyStandard.compare("WIFI_PHY_STANDARD_80211b")){
  wifi.SetStandard ( WIFI_PHY_STANDARD_80211b );
  numChannels = 14;
}else if(!WifiPhyStandard.compare("WIFI_PHY_STANDARD_80211g")){
  wifi.SetStandard ( WIFI_PHY_STANDARD_80211g );
  numChannels = 13;
```

```
}else if(!WifiPhyStandard.compare("WIFI_PHY_STANDARD_80211n_2_4GHz")){
  wifi.SetStandard ( WIFI_PHY_STANDARD_80211n_2_4GHZ );
  numChannels = 13;
}else if(!WifiPhyStandard.compare("WIFI_PHY_STANDARD_80211n_5GHz")){
  wifi.SetStandard ( WIFI_PHY_STANDARD_80211n_5GHZ );
}else if(!WifiPhyStandard.compare("WIFI_PHY_STANDARD_80211ac")){
  wifi.SetStandard ( WIFI_PHY_STANDARD_80211ac );
}else{
  NS_LOG_ERROR("ERROR: Estándar irreconocido");
  // Error
}

// Inicialización del modelo de movilidad
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

/* Inicialización de los parámetros individuales de cada nodo */

NS_LOG_INFO(">> Comienza la instalación de los nodos");

// Defino un canal para todos
if (Frequency == (float) 2.4){
  phy.SetChannel(channel.Create());
  spaceBetweenUsedChannels = numChannels/NumDrones;
  actualChannel = spaceBetweenUsedChannels/2;
}

// Para cada drone
for (vector<Dron>::iterator dron_t=escenario.drones.begin(); dron_t !=
    escenario.drones.end(); ++dron_t){

  if (Frequency == (float) 2.4){
  phy.Set ("Frequency", UintegerValue(channelFrequency[actualChannel]));
  actualChannel+=spaceBetweenUsedChannels;
  }else{
  phy.SetChannel(channel.Create());
  }
  // Defino su ssid
  Ssid ssid;
  ss.clear();
  ss.str("");
  ss << "ns3-ssid-" << (*dron_t).id;
  ssid = Ssid (ss.str());

  // Para cada usuario
  for (vector<Usuario>::iterator usuario_t=(*dron_t).usuarios.begin();
      usuario_t != (*dron_t).usuarios.end(); ++usuario_t){

    // Si el usuario es uno de los activos, lo encolo:
    if (usuario_t->active){
    activeUsers.push_back(usuario_t->id);
    }

    // Defino su PhyMode
    wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode",
        StringValue ((*usuario_t).wifiPhy), "ControlMode", StringValue ((*
       usuario_t).wifiPhy));
```

```
    // Defino su capa mac
    mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid), "ActiveProbing",
         BooleanValue (false));

    // Instalo el NetDevice y lo añado a la lista de NetDevices
        correspondiente
    userDevices.Add(wifi.Install (phy, mac, users.Get((*usuario_t).id-1)));

    // Añado la relación Mac - PhyMode para el RemoteStationManager del dron
        asignado
    mapa->phyMap[DynamicCast<WifiNetDevice>(users.Get((*usuario_t).id-1)->
        GetDevice(0))->GetMac()->GetAddress()] = (*usuario_t).wifiPhy;

    // Defino su posición
    positionAlloc = CreateObject<ListPositionAllocator> ();
    positionAlloc->Add (Vector ((*usuario_t).x, (*usuario_t).y, (*usuario_t).
        z)) ;

    // Instalo su posición
    mobility.SetPositionAllocator (positionAlloc);
    mobility.Install (users.Get((*usuario_t).id-1));

    // Defino su ganancia de recepción y transmisión
    DynamicCast<WifiNetDevice>(users.Get((*usuario_t).id-1)->GetDevice(0))->
        GetPhy()->SetTxGain((*usuario_t).gain);
    DynamicCast<WifiNetDevice>(users.Get((*usuario_t).id-1)->GetDevice(0))->
        GetPhy()->SetRxGain((*usuario_t).gain);

    NS_LOG_INFO(">>>> Instalado usuario " << (*usuario_t).id << " con NodeID:
         " << (users.Get((*usuario_t).id-1))->GetId());
  }

  // Defino su gestor de la tasa física
  wifi.SetRemoteStationManager ("ns3::MappedRateWifiManager", "Map",
      PointerValue(mapa));

  // Defino su capa mac
  mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid),"BeaconGeneration",
       BooleanValue (true), "BeaconInterval", TimeValue (Seconds (2.5)));

  // Instalo el NetDevice y lo añado a la lista de NetDevices correspondiente
  droneDevices.Add(wifi.Install (phy, mac, drones.Get((*dron_t).id-1)));

  // Defino su posición
  positionAlloc = CreateObject<ListPositionAllocator> ();
  positionAlloc->Add (Vector ((*dron_t).x, (*dron_t).y, (*dron_t).z)) ;

  // Instalo su posición
  mobility.SetPositionAllocator (positionAlloc);
  mobility.Install (drones.Get((*dron_t).id-1));

  NS_LOG_INFO(">>>> Instalado dron " << (*dron_t).id << " con NodeID: " << (
      drones.Get((*dron_t).id-1))->GetId());
}

for ( uint16_t cont = 0; cont < NumDrones; cont++){
```

```
  wiredDevices.push_back(p2p.Install(NodeContainer(drones.Get(cont), servers.
      Get(cont))));
}

/* Configuramos parámetros comunes a todos los nodos */
Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Phy/ChannelWidth",
     UintegerValue (ChannelWidth));
Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Phy/RxNoiseFigure"
    , DoubleValue (0));

// Todas las colas al valor de wifiQueue, dependiendo del estándar usa una u
    otra (pero sólo una!)
Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/$ns3::
    ApWifiMac/DcaTxop/Queue/MaxPacketNumber", UintegerValue(wifiQueue));
Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/$ns3::
    ApWifiMac/BE_EdcaTxopN/Queue/MaxPacketNumber", UintegerValue(wifiQueue));
Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/$ns3::
    ApWifiMac/BK_EdcaTxopN/Queue/MaxPacketNumber", UintegerValue(wifiQueue));
Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/$ns3::
    ApWifiMac/VO_EdcaTxopN/Queue/MaxPacketNumber", UintegerValue(wifiQueue));
Config::Set ("/NodeList/*/DeviceList/*/$ns3::WifiNetDevice/Mac/$ns3::
    ApWifiMac/VI_EdcaTxopN/Queue/MaxPacketNumber", UintegerValue(wifiQueue));

/* Configuramos TCP/IP y asignamos direcciones */

// Instalamos pila TCP/IP a todos los nodos
stack.Install(drones);
stack.Install(users);
stack.Install(servers);

// Asignamos direcciones de la parte wired
  for ( uint16_t cont = 0; cont < NumDrones; cont++){
  wiredNodeInterface.push_back(wiredAddress[cont].Assign (wiredDevices[cont])
      );
}

// Asignamos las direcciones IP por orden de ID... primero drones
droneNodeInterface = address.Assign (droneDevices);

// Reordeno los NetDevice de los users para que vayan en orden de ID
for (uint32_t cont = 0; cont < users.GetN(); cont++){
  userDevicesOrderedById.Add(users.Get(cont)->GetDevice(0));
}
userDevices = userDevicesOrderedById;
userDevicesOrderedById = ns3::NetDeviceContainer();

// Asigno las direcciones a los users desde mismo pool.
userNodeInterface = address.Assign (userDevices);

NS_LOG_INFO("");
NS_LOG_INFO(">> Resumen de la instalación:");
NS_LOG_INFO(">>>> Finalizado con " << droneDevices.GetN() << " drones
    instalados");
NS_LOG_INFO(">>>> Finalizado con " << userDevices.GetN() << " usuarios
    instalados");
NS_LOG_INFO("");
```

```
/* Desplegamos las aplicaciones */

// Habilito la captura de trazas de los drones (pcap) y todos (ascii)
/*
ss.clear();
ss.str("");
ss << output << "trazaAscii.tr";
phy.EnableAsciiAll (ascii.CreateFileStream (ss.str()));
*/
if (mustTrace){
  ss.clear();
  ss.str("");
  ss << output << "trazaWifi";
  phy.EnablePcapAll(ss.str());

  ss.clear();
  ss.str("");
  ss << output << "trazaP2P";
  p2p.EnablePcapAll(ss.str());
}
NS_LOG_INFO (">> Comienza la instalación de las aplicaciones:");

udpSource.SetAttribute ("MaxPackets", UintegerValue(4294967295u));
udpSource.SetAttribute ("Interval", TimeValue (Time ("20ms")));
udpSource.SetAttribute ("PacketSize", UintegerValue(UintegerValue(packetSize
    + rtpHeaderSize)));
udpSource.SetAttribute ("RemotePort", UintegerValue(port));

if (!onlyDownload){
  for ( uint16_t cont = 0; cont < droneDevices.GetN(); cont++){
    localAddress = AddressValue(InetSocketAddress (wiredNodeInterface[cont].
        GetAddress(1), port));
    sink.SetAttribute("Local", localAddress);
    apps_sink.Add(sink.Install(servers.Get(cont)));
  }
}

for ( uint16_t cont = 0; cont < NumActiveUsers; cont++){

  id_usuario = activeUsers[cont];

  // Busco el índice del dron con el que debe comunicarse:
  index_dron = buscaDron(droneDevices, userDevices, id_usuario);

  // Preparo el sink del usuario:
  localAddress = AddressValue(InetSocketAddress (userNodeInterface.GetAddress
      (id_usuario-1), port));
  sink.SetAttribute("Local", localAddress);

  apps_sink.Add(sink.Install(users.Get(id_usuario-1)));

  if (!onlyDownload){
  // Preparo uplink:
  tempValue = uniformRandomVariable->GetValue();

  remoteAddress = AddressValue(InetSocketAddress (wiredNodeInterface[
      index_dron].GetAddress(1), port));
```

```
  udpSource.SetAttribute ("RemoteAddress", remoteAddress);
  udpSource.SetAttribute ("StartTime", TimeValue(Seconds(initOffset +
      tempValue)));
  udpSource.SetAttribute ("StopTime", TimeValue(Seconds(sendTime + initOffset
       + tempValue)));

  apps_source.Add(udpSource.Install (users.Get(id_usuario-1)));
  }
  // Preparo downlink:
  tempValue = uniformRandomVariable->GetValue();

  remoteAddress = AddressValue(InetSocketAddress (userNodeInterface.
      GetAddress (id_usuario-1), port));
  udpSource.SetAttribute ("RemoteAddress", remoteAddress);
  udpSource.SetAttribute ("StartTime", TimeValue(Seconds(initOffset +
      tempValue)));
  udpSource.SetAttribute ("StopTime", TimeValue(Seconds(sendTime + initOffset
       + tempValue)));

  apps_source.Add(udpSource.Install (servers.Get(index_dron)));

  NS_LOG_INFO(">>>> Hablan el usuario " << id_usuario << " (" <<
      userNodeInterface.GetAddress (id_usuario-1) << ") y el servidor (" <<
      wiredNodeInterface[index_dron].GetAddress(1) << ") a través del dron "
      << index_dron + 1 );
}

// Preparo el lanzamiento de las aplicaciones sumidero
apps_sink.Start (Seconds (0));
apps_sink.Stop (Seconds (sendTime + finishOffset));

NS_LOG_INFO("");

Simulator::Stop (Seconds (sendTime + finishOffset + 1));

// Instalo el monitor de flujos en todos los nodos
observador->installFlowMonitor();

// Relleno las tablas de encaminamiento

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

// Por algún motivo que desconozco, no se añaden las entradas correctamente,
    falta la "salida" del usuario al dron
Ipv4StaticRoutingHelper ipv4RoutingHelper;
Ptr<Ipv4StaticRouting> staticRouting;

for (uint16_t cont = 0; cont < users.GetN(); cont++){
  index_dron = buscaDron(droneDevices, userDevices, cont+1);
  staticRouting = ipv4RoutingHelper.GetStaticRouting(users.Get(cont)->
      GetObject<Ipv4>());
  staticRouting->SetDefaultRoute(droneNodeInterface.GetAddress(index_dron),1)
      ;
}

WifiParameters params = observador->getWifiParameters(droneDevices.Get(0));
```

```
NS_LOG_INFO(">> Parámetros de capa MAC a utilizar: ");
NS_LOG_INFO(">>>> CWmin: " << params.CWmin);
NS_LOG_INFO(">>>> CWmax: " << params.CWmax);
NS_LOG_INFO(">>>> Time slot: " << params.timeSlot.GetMicroSeconds() << "us");
NS_LOG_INFO(">>>> DIFS: " << params.DIFS.GetMicroSeconds() << "us");
NS_LOG_INFO(">>>> SIFS: " << params.SIFS.GetMicroSeconds() << "us");
NS_LOG_INFO(">>>> EIFS: " << params.EIFS.GetMicroSeconds() << "us");
NS_LOG_INFO(">>>> SLRC: " << params.SLRC);
NS_LOG_INFO(">>>> SSRC: " << params.SSRC);
NS_LOG_INFO(">>>> GI: " << params.GI.GetNanoSeconds() << "ns");

NS_LOG_INFO("");

NS_LOG_UNCOND(">> Comienza la simulación");

PopulateArpCache();

Simulator::Run ();

NS_LOG_INFO("");

NS_LOG_UNCOND(">> Finaliza la simulación");

NS_LOG_UNCOND("");
NS_LOG_UNCOND(">> Resultados:");
NS_LOG_UNCOND("");

Statistics stats = observador->getFlows();
NS_LOG_UNCOND("\t\t" << "Rate (b/s)\t" << "P.L. (%)\t" << "Delay (us)\t" << "
    Jitter (us)");
if (!onlyDownload){
  NS_LOG_UNCOND("[UPLOAD]\t" << stats.rate[UPLOAD].Avg() <<"\t\t" << stats.
      per[UPLOAD].Avg() << "\t\t" << stats.delay[UPLOAD].Avg() << "\t\t" <<
      stats.jitter[UPLOAD].Avg());
}

NS_LOG_UNCOND("[DOWNLOAD]\t" << stats.rate[DOWNLOAD].Avg() <<"\t\t" << stats.
    per[DOWNLOAD].Avg() << "\t\t" << stats.delay[DOWNLOAD].Avg() << "\t\t" <<
     stats.jitter[DOWNLOAD].Avg());
NS_LOG_UNCOND("");

ss.clear();
ss.str("");
ss << output << "salida.txt";

Ptr<OutputStreamWrapper> outputStreamWrapper = Create<OutputStreamWrapper>(ss
    .str(),std::ios::out);
std::ostream *stream = outputStreamWrapper->GetStream ();
*stream << RngSeedManager::GetRun() << ";"
<< WifiPhyStandard << ";"
<< ChannelWidth << ";"
<< NumActiveUsers << ";"
<< wifiQueue << ";"
<< params.CWmin << ";"
<< params.CWmax << ";"
<< params.timeSlot.GetMicroSeconds() << ";"
<< params.DIFS.GetMicroSeconds() << ";"
```

```
  << params.SIFS.GetMicroSeconds() << ";"
  << params.EIFS.GetMicroSeconds() << ";"
  << params.SLRC << ";"
  << params.SSRC << ";"
  << params.GI.GetNanoSeconds() << ";"
  << stats.rate[DOWNLOAD].Avg() << ";"
  << stats.per[DOWNLOAD].Avg() << ";"
  << stats.delay[DOWNLOAD].Avg() << ";"
  << stats.jitter[DOWNLOAD].Avg() << ";"
  << stats.rate[UPLOAD].Avg() << ";"
  << stats.per[UPLOAD].Avg() << ";"
  << stats.delay[UPLOAD].Avg() << ";"
  << stats.jitter[UPLOAD].Avg() << std::endl;

  Simulator::Destroy ();
}
```

Código B.2 observador.cc.

```
//
// observador.cc
//
//
// Created by Vicente Jesús Mayor Gallego on 23/5/17.
//
//

#include "ns3/flow-monitor-module.h"
#include "mapped-rate-wifi-manager.h"
#include "observador.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("Observador");

Observador::Observador(uint16_t packetSize, uint16_t sendTime){
    NS_LOG_FUNCTION(this);
    m_packetSize = packetSize;
    m_sendTime = sendTime;
}

void Observador::installFlowMonitor(){
    NS_LOG_FUNCTION(this);
    monitor = flowMonitor.InstallAll();
}

Statistics Observador::getFlows(){
    NS_LOG_FUNCTION(this);
    prepareClassifier();
    prepareStats();

    uint16_t modo = 0;

    for (FlowMonitor::FlowStatsContainerI it = statsContainer.begin(); it !=
        statsContainer.end(); ++it){
```

```
      // Si el origen pertenece a la subred wifi, es tráfico de modo; en caso
          contrario, es de bajada
      modo = ((((ipv4Classifier->FindFlow(it->first)).sourceAddress).
          CombineMask("255.255.0.0")).IsEqual(Ipv4Address("192.168.0.0"))) ?
          UPLOAD : DOWNLOAD;

      NS_LOG_INFO(">> (" << (ipv4Classifier->FindFlow(it->first)).
          sourceAddress << ") -> (" << (ipv4Classifier->FindFlow(it->first)).
          destinationAddress << ")");
      NS_LOG_INFO(">>>> Tasa (carga útil): " << (it->second.rxPackets*
          m_packetSize*8)/(m_sendTime) << " b/s");

      stats.rate[modo].Update((it->second.rxPackets*m_packetSize*8)/(
          m_sendTime));

      if (it->second.rxPackets != 0){
          stats.per[modo].Update(100*(it->second.txPackets-it->second.
              rxPackets)/(it->second.txPackets));
          stats.delay[modo].Update(((it->second.delaySum)/(it->second.
              rxPackets)).GetMicroSeconds());
          stats.jitter[modo].Update(((it->second.jitterSum)/(it->second.
              rxPackets)).GetMicroSeconds());

          NS_LOG_INFO(">>>> PER: " << 100*(it->second.txPackets-it->second.
              rxPackets)/(it->second.txPackets) << " %");
          NS_LOG_INFO(">>>> Retardo (media): " << (it->second.delaySum)/(it->
              second.rxPackets));
    if (it->second.rxPackets != 1){
      NS_LOG_INFO(">>>> Jitter (media): " << (it->second.jitterSum)/(it->
          second.rxPackets-1));
          }
    NS_LOG_INFO("");

      }else{

          NS_LOG_INFO(">>>> PER: 100 %");
          NS_LOG_INFO("");

          stats.per[modo].Update(100);
      }

  }

  NS_LOG_INFO(">> Estadísticas medias de la red (subida): ");
  NS_LOG_INFO(">>>> Tasa: " << stats.rate[UPLOAD].Avg() << " b/s");
  NS_LOG_INFO(">>>> PER: " << stats.per[UPLOAD].Avg() << " %");
  NS_LOG_INFO(">>>> Retardo (media): " << stats.delay[UPLOAD].Avg() << "us");
  NS_LOG_INFO(">>>> Jitter (media): " << stats.jitter[UPLOAD].Avg() << "us");
  NS_LOG_INFO("");

  NS_LOG_INFO(">> Estadísticas medias de la red (bajada): ");
  NS_LOG_INFO(">>>> Tasa: " << stats.rate[DOWNLOAD].Avg() << " b/s");
  NS_LOG_INFO(">>>> PER: " << stats.per[DOWNLOAD].Avg() << " %");
  NS_LOG_INFO(">>>> Retardo (media): " << stats.delay[DOWNLOAD].Avg() << "us"
      );
  NS_LOG_INFO(">>>> Jitter (media): " << stats.jitter[DOWNLOAD].Avg() << "us"
      );
```

```
    NS_LOG_INFO("");

    return stats;

}

void Observador::prepareClassifier(){
    NS_LOG_FUNCTION(this);
    ipv4Classifier = DynamicCast<Ipv4FlowClassifier>(flowMonitor.GetClassifier
        ());
}

void Observador::prepareStats(){
    NS_LOG_FUNCTION(this);
    statsContainer = monitor->GetFlowStats();
}

WifiParameters Observador::getWifiParameters(Ptr <NetDevice> apDevice){
    NS_LOG_FUNCTION(this);

    PointerValue ptr;

    Ptr <WifiMac> mac = DynamicCast<WifiNetDevice>(apDevice)->GetMac();
    Ptr <WifiPhy> phy = DynamicCast<WifiNetDevice>(apDevice)->GetPhy();
    Ptr <WifiRemoteStationManager> apWifiRemoteStationManager = mac->
        GetWifiRemoteStationManager();

    mac->GetAttribute("DcaTxop", ptr);
    Ptr<DcaTxop> dca = ptr.Get<DcaTxop>();
    NS_ASSERT (dca != NULL);

    parameters.CWmin = dca->GetMinCw();
    parameters.CWmax = dca->GetMaxCw();
    parameters.SIFS = mac->GetSifs();
    parameters.timeSlot = mac->GetSlot();
    parameters.DIFS = parameters.SIFS + 2*parameters.timeSlot;
    parameters.EIFS = mac->GetEifsNoDifs()+parameters.DIFS;
    parameters.SLRC = apWifiRemoteStationManager->GetMaxSlrc();
    parameters.SSRC = apWifiRemoteStationManager->GetMaxSsrc();

    parameters.standard = phy->GetStandard();

    if ((parameters.standard == WIFI_PHY_STANDARD_80211n_2_4GHZ) || (parameters.
        standard == WIFI_PHY_STANDARD_80211n_5GHZ) || (parameters.standard ==
        WIFI_PHY_STANDARD_80211ac)){
        if (DynamicCast<WifiNetDevice>(apDevice)->GetPhy()->GetGuardInterval()){
            parameters.GI = Time(NanoSeconds(400));
        }else{
            parameters.GI = Time(NanoSeconds(800));
        }
    }else{
        parameters.GI = Time(NanoSeconds(800));
    }

    return parameters;
}
```

Código B.3 observador.h.

```
//
// observador.h
//
//
// Created by Vicente Jesús Mayor Gallego on 23/5/17.
//
//

#ifndef observador_h
#define observador_h

#define UPLOAD 1
#define DOWNLOAD 0
#define MAXMODES 2

#include "ns3/node.h"
#include "ns3/core-module.h"
#include "ns3/wifi-module.h"
#include "ns3/packet.h"
#include "ns3/average.h"

using namespace ns3;

class Statistics{

public:
    Average <double> rate [MAXMODES];
    // Almacenamos tasa (subida y bajada)
    Average <double> per [MAXMODES];
    // Almacenamos per (subida y bajada)
    Average <uint64_t> delay [MAXMODES];
    // Almacenamos delay (subida y bajada)
    Average <uint64_t> jitter [MAXMODES];
    // Almacenamos jitter (subida y bajada)
private:

};

struct WifiParameters{
    uint32_t CWmin;
    uint32_t CWmax;
    Time timeSlot;
    Time DIFS;
    Time SIFS;
    Time EIFS;
    uint32_t SLRC;
    uint32_t SSRC;
    Time GI;
    WifiPhyStandard standard;
};

class Observador{

public:
```

```
    // Metodos publicos
    Observador(uint16_t packetSize, uint16_t sendTime);
    void installFlowMonitor();
    Statistics getFlows();
    WifiParameters getWifiParameters(Ptr <NetDevice> apDevice);

private:
    // Metodos privados
    void prepareClassifier();
    void prepareStats();

    // Declaracion de variables
    FlowMonitorHelper flowMonitor;
    // Para registrar estadísticas de los flujos de tráfico.
    Ptr <FlowMonitor> monitor;
    // Útil para recopilar estadísticas.
    Ptr <Ipv4FlowClassifier> ipv4Classifier;
    // Útil para clasificar estadísticas.
    FlowMonitor::FlowStatsContainer statsContainer;
    // Contenedor de estadísticas recopiladas por FlowStats

    // Almacenamiento de estadísticas:
    Statistics stats;
    WifiParameters parameters;

    uint16_t m_packetSize;
    // Tamaño (útil) de paquetes.
    uint16_t m_sendTime;
    // Tiempo de simulación, en segundos.

};

#endif /* observador_h */
```

Código B.4 mapped-rate-wifi-manager.cc.

```
 // Modificación de constant-rate-wifi-manager para que clone la tasa del
     objetivo

#include "mapped-rate-wifi-manager.h"
#include "ns3/assert.h"
#include "ns3/log.h"
#include "ns3/pointer.h"

#define Min(a,b) ((a < b) ? a : b)

namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("MappedRateWifiManager");

NS_OBJECT_ENSURE_REGISTERED (MappedRateWifiManager);

TypeId
MappedRateWifiManager::GetTypeId (void)
{
  static TypeId tid = TypeId ("ns3::MappedRateWifiManager")
```

```
    .SetParent<WifiRemoteStationManager> ()
    .SetGroupName ("Wifi")
    .AddConstructor<MappedRateWifiManager> ()
  .AddAttribute ("Map", "Mapeo entre Dirección Mac Destino - Modulación",
         PointerValue(),
         MakePointerAccessor (&MappedRateWifiManager::mapa),
         MakePointerChecker<Mapa>())
  ;
  return tid;
}

MappedRateWifiManager::MappedRateWifiManager ()
{
    NS_LOG_FUNCTION (this);
}

MappedRateWifiManager::~MappedRateWifiManager ()
{
    NS_LOG_FUNCTION (this);
}

WifiRemoteStation *
MappedRateWifiManager::DoCreateStation (void) const
{
  NS_LOG_FUNCTION (this);
  WifiRemoteStation *station = new WifiRemoteStation ();
  return station;
}

void
MappedRateWifiManager::DoReportRxOk (WifiRemoteStation *station,
                                     double rxSnr, WifiMode txMode)
{
  NS_LOG_FUNCTION (this << station << rxSnr << txMode);
}

void
MappedRateWifiManager::DoReportRtsFailed (WifiRemoteStation *station)
{
  NS_LOG_FUNCTION (this << station);
}

void
MappedRateWifiManager::DoReportDataFailed (WifiRemoteStation *station)
{
  NS_LOG_FUNCTION (this << station);
}

void
MappedRateWifiManager::DoReportRtsOk (WifiRemoteStation *st,
                                      double ctsSnr, WifiMode ctsMode, double
                                          rtsSnr)
{
  NS_LOG_FUNCTION (this << st << ctsSnr << ctsMode << rtsSnr);
}

void
```

```
MappedRateWifiManager::DoReportDataOk (WifiRemoteStation *st,
                                       double ackSnr, WifiMode ackMode, double
                                           dataSnr)
{
    NS_LOG_FUNCTION (this << st << ackSnr << ackMode << dataSnr);
}

void
MappedRateWifiManager::DoReportFinalRtsFailed (WifiRemoteStation *station)
{
  NS_LOG_FUNCTION (this << station);
}

void
MappedRateWifiManager::DoReportFinalDataFailed (WifiRemoteStation *station)
{
  NS_LOG_FUNCTION (this << station);
}

WifiTxVector
MappedRateWifiManager::DoGetDataTxVector (WifiRemoteStation *st)
{
    NS_LOG_FUNCTION (this << st);
    /* Modificación para que copie la modulación del objetivo */
    m_dataMode = WifiMode(mapa->phyMap[st->m_state->m_address]);

    return WifiTxVector (m_dataMode, GetDefaultTxPowerLevel (),
        GetLongRetryCount (st), GetShortGuardInterval (st), Min(
        GetNumberOfTransmitAntennas (), GetNumberOfSupportedRxAntennas (st)), 0,
         GetChannelWidth (st), GetAggregation (st), false);
}

WifiTxVector
MappedRateWifiManager::DoGetRtsTxVector (WifiRemoteStation *st)
{
  NS_LOG_FUNCTION (this << st);
  /* Modificación para que copie la modulación del objetivo */
  m_ctlMode = WifiMode(mapa->phyMap[st->m_state->m_address]);
  return WifiTxVector (m_ctlMode, GetDefaultTxPowerLevel (), GetShortRetryCount
       (st), GetShortGuardInterval (st), 1, 0, GetChannelWidth (st),
      GetAggregation (st), false);
}

bool
MappedRateWifiManager::IsLowLatency (void) const
{
  NS_LOG_FUNCTION (this);
  return true;
}

} //namespace ns3
```

Código B.5 mapped-rate-wifi-manager.h.

```
#ifndef MAPPED_RATE_WIFI_MANAGER_H
#define MAPPED_RATE_WIFI_MANAGER_H
```

```
#include <stdint.h>
#include <map>
#include "ns3/wifi-remote-station-manager.h"
#include "ns3/average.h"
#include "ns3/nist-error-rate-model.h"

namespace ns3 {

/**
 * \ingroup wifi
 * \brief use constant rates for data and RTS transmissions
 *
 * This class uses always the same transmission rate for every
 * packet sent.
 */
class Mapa : public Object
{
public:
    std::map <ns3::Mac48Address, std::string> phyMap;
private:

};

class MappedRateWifiManager : public WifiRemoteStationManager
{
public:
    static TypeId GetTypeId (void);
    MappedRateWifiManager ();
    virtual ~MappedRateWifiManager ();
private:
    //overriden from base class
    virtual WifiRemoteStation* DoCreateStation (void) const;
    virtual void DoReportRxOk (WifiRemoteStation *station,
                           double rxSnr, WifiMode txMode);
    virtual void DoReportRtsFailed (WifiRemoteStation *station);
    virtual void DoReportDataFailed (WifiRemoteStation *station);
    virtual void DoReportRtsOk (WifiRemoteStation *station,
                            double ctsSnr, WifiMode ctsMode, double rtsSnr);
    virtual void DoReportDataOk (WifiRemoteStation *station,
                             double ackSnr, WifiMode ackMode, double dataSnr);
    virtual void DoReportFinalRtsFailed (WifiRemoteStation *station);
    virtual void DoReportFinalDataFailed (WifiRemoteStation *station);
    virtual WifiTxVector DoGetDataTxVector (WifiRemoteStation *station);
    virtual WifiTxVector DoGetRtsTxVector (WifiRemoteStation *station);
    virtual bool IsLowLatency (void) const;

    WifiMode m_dataMode; //!< Wifi mode for unicast DATA frames
    WifiMode m_ctlMode; //!< Wifi mode for RTS frames

    Ptr <Mapa> mapa;
};

} //namespace ns3

#endif /* MAPPED_RATE_WIFI_MANAGER_H */
```

Código B.6 funciones.cc.

```
// Para NS3:
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/wifi-module.h"
#include "ns3/internet-module.h"

// Para lectura de ficheros:
#include <list>
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <stdio.h>
#include <stdlib.h>
#include "escenario.h"
#include "funciones.h"

// Para NS3:
using namespace ns3;

NS_LOG_COMPONENT_DEFINE("Funciones");

// Lee el fichero y lo guarda en un objeto Escenario
Escenario cargaEscenario(xmlNode * a_node, Escenario escenario)
{
    xmlNode *cur_node = NULL;
    for (cur_node = a_node; cur_node; cur_node = cur_node->next) {
        if (cur_node->type == XML_ELEMENT_NODE) {
      Dron dron;
      Usuario usuario;
      int elemento_actual = -1;
      if (!strcmp((char *) (cur_node->name), "escenario")){
        elemento_actual = 1;
      }else if (!strcmp((char *) (cur_node->name), "dron")) {
        elemento_actual = 2;
        escenario.numDrones++;
      }else if (!strcmp((char *) (cur_node->name), "usuario")) {
        elemento_actual = 3;
      }else{
        elemento_actual = -1;
        // error en el XML
      }
           //printf("Encontrado elemento: %s\n", cur_node->name);
      xmlAttr* attribute = cur_node->properties;
      while(attribute)
      {
        xmlChar* value = xmlNodeListGetString(cur_node->doc, attribute->children
            , 1);
        switch (elemento_actual){
            case 1:
              // Es el escenario raiz
              if (!strcmp((char *) attribute->name, "W")){
                escenario.w = atof( (char *) value);
              }else if(!strcmp((char *) attribute->name, "L")){
                escenario.l = atof( (char *) value);
              }else if(!strcmp((char *) attribute->name, "ChannelWidth")){
                escenario.channelWidth = atof( (char *) value);
```

```
            }else if(!strcmp((char *) attribute->name, "Frequency")){
              escenario.frequency = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "Ptx")){
              escenario.pTx = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "ExpPropagation")){
              escenario.expPropagation = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "NumActiveUsers")){
              escenario.numActiveUsers = atoi( (char *) value);
            }else if(!strcmp((char *) attribute->name, "NumUsers")){
              escenario.numUsers = atoi( (char *) value);
            }else if(!strcmp((char *) attribute->name, "ShortGuardEnabled")){
              escenario.shortGuardEnabled = !strcmp("true", (char *) value);
            }else if(!strcmp((char *) attribute->name, "WifiPhyStandard")){
              escenario.wifiPhyStandard = (char *) value;
            }else{
              // error en el XML
              NS_LOG_ERROR("Error en el XML");
            }
            break;
          case 2:
            // Es un dron
            if (!strcmp((char *) attribute->name, "id")){
              dron.id = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "x")){
              dron.x = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "y")){
              dron.y = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "z")){
              dron.z = atof( (char *) value);
            }else{
              NS_LOG_ERROR("Error en el XML");
            }
            break;
          case 3:
            // Es un usuario

            if (!strcmp((char *) attribute->name, "id")){
              usuario.id = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "x")){
              usuario.x = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "y")){
              usuario.y = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "z")){
              usuario.z = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "PhyRate")){
              usuario.phyRate = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "Gain")){
              usuario.gain = atof( (char *) value);
            }else if(!strcmp((char *) attribute->name, "WifiPhy")){
              usuario.wifiPhy = (char *) value;
            }else if(!strcmp((char *) attribute->name, "Active")){
              usuario.active = !strcmp("true", (char *) value);
            }else{
              NS_LOG_ERROR("Error en el XML");
            }
            break;
          default:
```

```
            NS_LOG_ERROR("Error en el XML");
            break;
          }
          xmlFree(value);
          attribute = attribute->next;
      }

      switch (elemento_actual){
        case 1:
          // nada
          break;
        case 2:
          escenario.drones.push_back(dron);
          break;
        case 3:
          escenario.drones.back().usuarios.push_back(usuario);
          break;
        default:
          NS_LOG_ERROR("Error en el XML");
          break;
      }
        }
        escenario = cargaEscenario(cur_node->children, escenario);
    }
  return escenario;
}

uint16_t buscaDron (ns3::NetDeviceContainer droneDevices, ns3::
    NetDeviceContainer userDevices, uint16_t id_user){
  uint16_t index_dron;

  for (uint16_t cont = 0; cont < droneDevices.GetN(); cont++){
    if (DynamicCast<WifiNetDevice>(droneDevices.Get(cont))->GetMac()->GetSsid().
        IsEqual(DynamicCast<WifiNetDevice>(userDevices.Get(id_user-1))->GetMac
        ()->GetSsid())){
      index_dron = cont;
    }
  }

  return index_dron;
}

void PopulateArpCache ()
{
  Ptr<Packet> dummy = Create<Packet> ();
  Ptr<ArpCache> arp = CreateObject<ArpCache> ();
  arp->SetAliveTimeout (Seconds (3600 * 24 * 365));
  for (NodeList::Iterator i = NodeList::Begin (); i != NodeList::End ();
++i)
    {
      Ptr<Ipv4L3Protocol> ip = (*i)->GetObject<Ipv4L3Protocol> ();
      NS_ASSERT (ip != 0);
      ObjectVectorValue interfaces;
      ip->GetAttribute ("InterfaceList", interfaces);
      for (ObjectVectorValue::Iterator j = interfaces.Begin (); j !=
interfaces.End (); j++)
        {
```

```
        Ptr<Ipv4Interface> ipIface =
j->second->GetObject<Ipv4Interface> ();
        NS_ASSERT (ipIface != 0);
        Ptr<NetDevice> device = ipIface->GetDevice ();
        NS_ASSERT (device != 0);
        Mac48Address addr = Mac48Address::ConvertFrom
(device->GetAddress ());
        for (uint32_t k = 0; k < ipIface->GetNAddresses (); k ++)
          {
            Ipv4Address ipAddr = ipIface->GetAddress (k).GetLocal ();
            if (ipAddr == Ipv4Address::GetLoopback ())
              {
                continue;
              }
            Ipv4Header ipHeader;
            ArpCache::Entry *entry = arp->Add (ipAddr);
            entry->MarkWaitReply (ns3::ArpCache::Ipv4PayloadHeaderPair(dummy,
                ipHeader));
            entry->MarkAlive (addr);
            entry->ClearPendingPacket();
            entry->MarkPermanent ();
          }
      }
   }
  for (NodeList::Iterator i = NodeList::Begin (); i != NodeList::End ();
++i)
   {
     Ptr<Ipv4L3Protocol> ip = (*i)->GetObject<Ipv4L3Protocol> ();
     NS_ASSERT (ip != 0);
     ObjectVectorValue interfaces;
     ip->GetAttribute ("InterfaceList", interfaces);
     for (ObjectVectorValue::Iterator j = interfaces.Begin (); j !=
interfaces.End (); j ++)
       {
         Ptr<Ipv4Interface> ipIface =
j->second->GetObject<Ipv4Interface> ();
         ipIface->SetAttribute ("ArpCache", PointerValue (arp));
       }
   }
}
```

Código B.7 funciones.h.

```
#ifndef FUNCIONES_H
#define FUNCIONES_H

// Funciones declaradas
Escenario cargaEscenario(xmlNode * a_node, Escenario escenario);
uint16_t buscaDron (ns3::NetDeviceContainer droneDevices, ns3::
    NetDeviceContainer userDevices, uint16_t id_user);
void PopulateArpCache ();
#endif
```

Código B.8 escenario.h.

```
#ifndef ESCENARIO_H
#define ESCENARIO_H

// Clases declaradas
class Usuario {
  public:
  float id, phyRate, gain, x, y, z;
  bool active;
  std::string wifiPhy;
};

class Dron {
  public:
  uint32_t numUsers = 0;
  float id, x, y, z;
  std::vector <Usuario> usuarios;
};

class Escenario {
  public:
  float w, l, frequency, channelWidth, pTx, expPropagation;
  uint32_t numUsers, numDrones = 0, numActiveUsers;
  bool shortGuardEnabled;
  std::string wifiPhyStandard;
  std::vector <Dron> drones;
};

#endif
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] R. M. Gray, ‘‘The 1974 origins of voip,’’ *IEEE Signal Processing Magazine*, vol. 22, no. 4, pp. 87–90, 2005.

[2] M. A. R. Siddique and J. Kamruzzaman, ‘‘Voip call capacity over wireless mesh networks,’’ in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*. IEEE, 2008, pp. 1–6.

[3] D. B. Green and A. Obaidat, ‘‘An accurate line of sight propagation performance model for ad-hoc 802.11 wireless lan (wlan) devices,’’ in *Communications, 2002. ICC 2002. IEEE International Conference on*, vol. 5. IEEE, 2002, pp. 3424–3428.

[4] L. Liechty, E. Reifsnider, and G. Durgin, ‘‘Developing the best 2.4 ghz propagation model from active network measurements.’’ in *VTC Fall*, 2007, pp. 894–896.

[5] S. Shin and H. Schulzrinne, ‘‘Measurement and analysis of the voip capacity in ieee 802.11 wlan,’’ *IEEE transactions on mobile computing*, vol. 8, no. 9, pp. 1265–1279, 2009.

[6] A. Takahashi, H. Yoshino, and N. Kitawaki, ‘‘Perceptual qos assessment technologies for voip,’’ *IEEE Communications Magazine*, vol. 42, no. 7, pp. 28–34, 2004.

[7] N. Unuth, ‘‘Mean opinion score (mos)-a measure of voice quality,’’ *Retrieved May*, vol. 5, p. 2011, 2010.

[8] P. Serrano, A. Garcia-Saavedra, M. Hollick, and A. Banchs, ‘‘On the energy efficiency of ieee 802.11 wlans,’’ in *Wireless Conference (EW), 2010 European*. IEEE, 2010, pp. 932–939.

[9] E. Cocker, F. Ghazzi, and U. Speidel, ‘‘Quality trend measurement of long-distance voip communication via estimated mean opinion score,’’ in *Information, Communications and Signal Processing (ICICS), 2015 10th International Conference on*. IEEE, 2015, pp. 1–5.

[10] A. Bouhlel, V. Guillet, G. El Zein, and G. Zaharia, ‘‘Simulation analysis of wireless channel effect on ieee 802.11 n physical layer,’’ in *Vehicular Technology Conference (VTC Spring), 2012 IEEE 75th*. IEEE, 2012, pp. 1–5.

[11] A. Fotouhi, M. Ding, and M. Hassan, ‘‘Dynamic base station repositioning to improve spectral efficiency of drone small cells,’’ *arXiv preprint arXiv:1704.01244*, 2017.

[12] P. T. Schultz, J. R. Barfield, T. M. Willis, R. A. Sartini *et al.*, ‘‘System and method for providing extension of network coverage,’’ Jul. 19 2016, uS Patent 9,398,467.

[13] D. Reina, J. M. L. Coca, M. Askalani, S. Toral, F. Barrero, E. Asimakopoulou, S. Sotiriadis, and N. Bessis, ‘‘A survey on ad hoc networks for disaster scenarios,’’ in *Intelligent Networking and Collaborative Systems (INCoS), 2014 International Conference on*. IEEE, 2014, pp. 433–438.

[14] D. Reina, S. L. Toral, F. Barrero, N. Bessis, and E. Asimakopoulou, ‘‘Evaluation of ad hoc networks in disaster scenarios,’’ in *Intelligent Networking and Collaborative Systems (INCoS), 2011 Third International Conference on*. IEEE, 2011, pp. 759–764.

[15] X. Li, D. Guo, H. Yin, and G. Wei, ‘‘Drone-assisted public safety wireless broadband network,’’ in *Wireless Communications and Networking Conference Workshops (WCNCW), 2015 IEEE*. IEEE, 2015, pp. 323–328.

[16] Matlab - el lenguaje del cálculo técnico. Mathworks. [Online]. Available: https://es.mathworks.com/products/matlab.html

[17] P.800.1 : mean opinion score (mos) terminology. ITU. [Online]. Available: https://www.itu.int/rec/T-REC-P.800.1

[18] ns-3 network simulator. [Online]. Available: https://www.nsnam.org/

[19] Anonymous functions - matlab & simulink - mathworks españa. [Online]. Available: https://es.mathworks.com/help/matlab/matlab_prog/anonymous-functions.html

[20] Enumeration of combinations - matlab combnk - mathworks españa. [Online]. Available: https://es.mathworks.com/help/stats/combnk.html

[21] Subscripts from linear index - matlab ind2sub - mathworks españa. MATLAB. [Online]. Available: https://es.mathworks.com/help/matlab/ref/ind2sub.html

[22] F. Altiparmak, M. Gen, L. Lin, and T. Paksoy, ‘‘A genetic algorithm approach for multi-objective optimization of supply chain networks,’’ *Computers & industrial engineering*, vol. 51, no. 1, pp. 196–215, 2006.

[23] P. Bevelacqua. The short dipole antenna. [Online]. Available: http://www.antenna-theory.com/antennas/shortdipole.php

[24] Wi-fi module — model library. [Online]. Available: https://www.nsnam.org/docs/models/html/wifi.html

[25] ns-3: Applications. [Online]. Available: https://www.nsnam.org/doxygen/group__applications.html

[26] A. J. Estepa, R. Estepa, J. Vozmediano, and P. Carrillo, ‘‘Dynamic voip codec selection on smartphones,’’ *Network Protocols and Algorithms*, vol. 6, no. 2, pp. 22–37, 2014.

[27] The xml c parser and toolkit of gnome. [Online]. Available: http://xmlsoft.org/

[28] Libxml2 set of examples. [Online]. Available: http://xmlsoft.org/examples/index.html

[29] Bug 187 – need ’perfectárp. [Online]. Available: https://www.nsnam.org/bugzilla/show_bug.cgi?id=187

[30] D. Malone, K. Duffy, and D. Leith, ‘‘Modeling the 802.11 distributed coordination function in nonsaturated heterogeneous conditions,’’ *IEEE/ACM Transactions on networking*, vol. 15, no. 1, pp. 159–172, 2007.

[31] ns-3: ns3::nisterrorratemodel class reference. [Online]. Available: https://www.nsnam.org/doxygen/classns3_1_1_nist_error_rate_model.html

[32] ns-3: ns3::dssserrorratemodel class reference. [Online]. Available: https://www.nsnam.org/doxygen/classns3_1_1_dsss_error_rate_model.html

[33] G. Pei and T. Henderson, ‘‘Validation of ns-3 802.11 b phy model,’’ *Online: http://www. nsnam. org/~ pei/80211b. pdf*, 2009.

[34] Wi-fi snr to mcs data rate mapping reference — revolution wi-fi. [Online]. Available: http://www.revolutionwifi.net/revolutionwifi/2014/09/wi-fi-snr-to-mcs-data-rate-mapping.html

[35] Mcs index, modulation and coding index 11n and 11ac. [Online]. Available: http://mcsindex.com/