Facultad de Ingeniería, Universidad de la República

Instituto de Computación

Aceleración de Radios Definidas por Software

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación

Autores
Gonzalo Arcos
Rodrigo Ferreri

Tutores
Pablo Ezzatti
Eduardo Grampin
Matías Richart

Montevideo, Uruguay
Año 2016

Resumen

Comúnmente, se asocia el concepto de radio únicamente a dispositivos capaces de sintonizar estaciones de difusión o broadcasting AM y FM. Sin embargo, la definición correcta incluye cualquier dispositivo que sea capaz de transmitir o recibir ondas electromagnéticas dentro de un rango de frecuencias determinado, y de realizar el procesamiento de la señal necesario para poder codificar o decodificar la información a partir de la señal. Bajo esta definición, tanto un celular, un enrutador inalámbrico, un dispositivo Bluetooth, como otros tantos, pueden ser clasificados como radios.

Las *Radios Definidas por Software* (*Software Defined Radios*, SDR), un tipo de sistema de radiocomunicación en donde todos o algunos de los componentes que realizan el procesamiento de la señal ejecutan en un procesador de propósito general, ofrecen un gran número de ventajas frente a las radios tradicionales implementadas en hardware, entre las cuales se encuentra la posibilidad de modificar el comportamiento de las mismas sin la necesidad de cambiar el hardware subyacente. Un sistema que implemente los componentes más básicos (por ejemplo, las antenas de radio) y que disponga de un procesador de propósito general puede, utilizando las técnicas de SDR, implementar funcionalidades tan diversas como los protocolos Wi-Fi, LTE o 3G, simplemente cambiando el software utilizado para realizar el procesamiento de la señal.

Si bien el concepto de SDR no es nuevo, la posibilidad de implementar en software protocolos con altas tasas de transferencia (como Wi-Fi) está pasando de ser un concepto teórico interesante a una posibilidad realizable debido principalmente a la gran mejora en la velocidad de los procesadores modernos. A la luz de este concepto es que se desarrolla este proyecto.

El presente trabajo realiza, en primer lugar, una extensa presentación de las bases teóricas que permiten entender el funcionamiento de un sistema de comunicación inalámbrica a bajo nivel.

En segundo lugar, presenta un estudio del estado del arte de las radios definidas por software, centrándose en particular en la última versión del estándar Wi-Fi y su aplicación en SDRs, así como también en la herramienta GNU Radio, cuya finalidad es permitir diseñar e implementar tanto SDRs como procesadores de señales digitales (*Digital Signal Processors*, DSPs) en software.

En tercer lugar, partiendo de una implementación preexistente de un transmisor y receptor Wi-Fi desarrollada en GNU Radio, y con el objetivo de evaluar el rendimiento y capacidad de transferencia de la misma en comparación con las tasas especificadas en el estándar, el trabajo fija el objetivo de lograr un aumento en el rendimiento original de la SDR, de forma de acercarlo al de una implementación realizada en hardware dedicado.

En esta línea es que se presentan distintas estrategias de optimización sobre los principales componentes, obteniendo como resultado final un incremento de entre dos y diez veces la tasa de transferencia original, dependiendo del esquema de modulación y la tasa de codificación elegidos.

Palabras clave: Códigos Convolucionales, GNU Radio, IEEE 802.11, Orthogonal Frequency Division Multiplexing (OFDM), Software-Defined Radios (SDR), Decodificador de Viterbi, Wi-Fi, gr-ieee802-11.

Índice general

Capítulo 1

Introducción

1.1. Motivación

Las *Radios Definidas por Software* (*Software Defined Radios*, SDR) son un tipo de sistema de radiocomunicación en donde todos o algunos de los componentes que realizan el procesamiento de la señal han sido implementados mediante software que ejecuta en un procesador de propósito general[1]. Ejemplos convencionales de este tipo de procesadores son los que se encuentran en las computadoras de escritorio o portátiles, así como en dispositivos móviles.

Las SDR han sido motivo de interés, tanto para el ámbito académico como para el sector de las telecomunicaciones, por más de veinte años. Cuando a principios de la década de los 90 fuera inventado el término en [1], hubo una revolución importante en la forma de pensar la ingeniería de radio que generó un nuevo abanico de posibilidades. Sin embargo, no ha sido sino hasta recientemente que las SDR han empezado a convertirse en una potente solución de uso amplio y accesible, y una importante alternativa a considerar frente a las radios convencionales.

Esta evolución se ha debido a varios factores, entre los cuales se encuentran los avances significativos en las tecnologías de procesamiento digital, conversores análogo-digital y digital-análogo, herramientas de software y hardware de radio, así como a la mejora en la tecnología de los microprocesadores.

Además de esto, que el procesamiento de la señal se realice a nivel de software y no mediante hardware dedicado presenta un conjunto importante de ventajas que la radio convencional no puede brindar. En primer lugar, permite un mayor nivel de abstracción para el diseñador, lo que redunda en una tarea de diseño e implementación más sencilla

[1] Se utilizará el acrónimo GPP (*General Purpose Processor*) para referirse a los procesadores de propósito general.

y que puede realizarse en menos tiempo. En segundo lugar, el mismo hardware (típicamente un GPP) puede reutilizarse con diferentes implementaciones de transceptores[2], siendo estos últimos sistemas capaces de transmitir y recibir información a través de un canal de comunicación utilizando un protocolo de comunicación determinado. En tercer lugar, el código puede ser modificado fácilmente, otorgando una gran flexibilidad en tiempo real que permite, por ejemplo, realizar actualizaciones sobre una implementación existente. Por último, el código fuente puede ser compartido a gran escala entre programadores, ingenieros e investigadores.

Las razones antes expuestas han motivado que las SDR se hayan vuelto, cada vez en mayor medida, una herramienta importante tanto para la educación como para la investigación y el desarrollo. En particular, destaca la capacidad de prototipado de nuevos algoritmos y paradigmas.

Los últimos avances en las tecnologías de SDR han fomentado la aparición tanto de plataformas de hardware SDR con suficiente poder computacional como para operar en un rango amplio de frecuencias y anchos de banda, como de entornos de desarrollo de software con funcionalidades de alto nivel sobre distintas plataformas, incluyendo un amplio conjunto de módulos y algoritmos y una curva de aprendizaje razonable.

Estos avances hacen que la plataforma (en su conjunto) sea idónea para el desarrollo de prototipos, pero aún están lejos de permitir su uso en implementaciones comerciales de estándares modernos, como lo son Wi-Fi, 3G y LTE, principalmente por presentar un rendimiento inferior al de una radio convencional implementada en hardware dedicado. Esta realidad motiva el presente trabajo, en donde se intenta avanzar en el desempeño de las implementaciones de SDR, de manera de acercar el uso comercial de dicha tecnología.

1.2. Objetivos

En primer lugar, se planteó el objetivo de realizar un estudio de las bases teóricas que permiten implementar comunicaciones inalámbricas a nivel físico. Dado que el proyecto se enmarca en el ámbito de la carrera Ingeniería en Computación, en la cual dichos temas no se estudian en profundidad, este objetivo cobró una importancia especial para lograr un entendimiento completo del temario a tratar.

En segundo lugar, se planteó el objetivo de realizar un estudio del estado del arte de las radios definidas por software, del estándar Wi-Fi[3] y de OFDM[4]. Esto incluyó

[2] Un transceptor es un dispositivo capaz de realizar tanto la transmisión como la recepción de una señal.

[3] Estándar IEEE 802.11.

[4] OFDM significa *Orthogonal Frequency Division Multiplexing*, Multiplexación por División de Frecuencias Ortogonales, que es un tipo de modulación que codifica la información de una manera especial. Dicho esquema de modulación se encuentra explicado con detalle en la Sección 2.3.

analizar con detalle una implementación preexistente de un transmisor y receptor Wi-Fi en GNU Radio, por lo que también fue necesario un estudio del funcionamiento de dicha herramienta.

Por último, el tercer objetivo planteado fue obtener alguna mejora en el desempeño de la implementación que se tomó como punto de partida. Esto incluyó realizar una serie de mediciones para determinar cuellos de botella, y un estudio de las posibilidades de mejora sobre los módulos que se identificaran como los más demandantes desde el punto de vista computacional, así como la implementación final de las mismas.

1.3. Estructura del documento

Se presenta a continuación la estructura del resto de este documento.

El Capítulo 2 introduce los conceptos teóricos que son necesarios para poder comprender el trabajo realizado. En particular, se hace una breve introducción a la comunicación analógica y digital, a los modelos para la comunicación sobre un canal físico y a los distintos esquemas de modulación. También se realiza un breve estudio sobre los códigos de corrección de errores, en especial los llamados códigos convolucionales. Se finaliza el capítulo con una descripción del esquema de modulación OFDM, incluyendo las ventajas que presenta sobre los esquemas tradicionales.

El Capítulo 3 describe el estado del arte de los temas más importantes tratados en este proyecto. En primer lugar, se presenta un resumen de las secciones del estándar Wi-Fi que son más relevantes para entender el trabajo. Luego se realiza un breve estudio sobre las tecnologías de radios definidas por software utilizadas en la actualidad, haciendo énfasis especial en la herramienta GNU Radio. También se realiza una breve mención de los dispositivos utilizados para transmitir señales en el aire cuando se está trabajando con una SDR.

El Capítulo 4 presenta en primer lugar un análisis de la implementación original del transmisor y receptor IEEE 802.11 a/g/p diseñado en GNU Radio y utilizado a lo largo del proyecto. Luego, se explican con detalle las técnicas que fueron aplicadas para identificar los módulos más costosos de la implementación original, y los cambios y mejoras que fueron realizados sobre ésta con el objetivo de incrementar su rendimiento.

El Capítulo 5 muestra el conjunto de pruebas llevadas a cabo para evaluar las propuestas realizadas en el capítulo anterior. Se presentan, en primer lugar, las pruebas realizadas sobre GNU Radio enteramente en software, y en segundo lugar, las pruebas de concepto que se realizaron con antenas físicas.

Las conclusiones obtenidas a partir del trabajo realizado se presentan en el Capítulo 6. Además, se resumen distintas líneas de investigación que es posible seguir a partir del presente trabajo.

Por último, el documento cuenta con tres anexos. El Anexo A explica cómo realizar una instalación completa de GNU Radio junto con todos los módulos que fueron utilizados para llevar a cabo este proyecto. En el Anexo B se explica cómo crear un bloque en la herramienta GNU Radio. El Anexo C contiene el Glosario de términos de uso común.

Capítulo 2

Conceptos Teóricos

2.1. Comunicación analógica y digital

El propósito final del uso de cualquier tecnología de comunicación siempre ha sido permitir a las personas enviar y recibir mensajes. Por esta razón, una cuestión muy importante a resolver es la de elegir cómo serán representados los mensajes al ser enviados. Para este propósito se distinguen dos formas: la *analógica* y la *digital*.

La representación digital asume que la información será codificada utilizando una representación discreta. En particular, se adoptará la convención de que los datos digitales serán representados utilizando dígitos binarios (*bits*). Por otra parte, la representación analógica utiliza señales análogas que se comportan de manera continua, representando la información mediante una función continua.

Ciertos datos se representan mejor de manera digital. Por ejemplo, los datos generados por una computadora se encuentran, de manera natural, codificados mediante secuencias de bits. Por otra parte, también se pueden encontrar datos cuya representación más natural es analógica, como lo son una imagen o un video. En general, cualquier dispositivo que obtenga datos de una fuente natural tendrá que hacer frente a magnitudes que adoptan valores en una escala continua.

Existen dos razones principales por las que utilizar una representación digital es mejor que una analógica:

1. Utilizar una abstracción digital permite la composición de módulos para construir sistemas grandes.

2. Utilizar una abstracción digital permite procesar datos utilizando algoritmos sofisticados, que eventualmente pueden lograr mejorar la calidad y eficiencia de los componentes de un sistema.

Sin embargo, la abstracción digital no es la manera natural de representar la información, debido a que, en su más bajo nivel, los enlaces de comunicación son analógicos. Esto introduce la siguiente interrogante: ¿Por qué no utilizar una representación analógica entonces?

La respuesta es que no existe un enlace de comunicaciones que esté libre de errores. Todos los enlaces sufren de perturbaciones, que provienen tanto del ruido como de ciertos tipos de distorsión[1]. Estas perturbaciones afectan la señal emitida, por lo que cada vez que hay una transmisión, el receptor no recibirá la señal de manera exacta, sino una versión perturbada. Más grave aún es el hecho de que estas perturbaciones se pueden acumular si se utilizan múltiples enlaces de comunicación, provocando que la salida de un enlace pueda ser un valor completamente diferente al que fue enviado por el transmisor.

Uno de los objetivos principales en este campo es el de garantizar que los valores de salida sean correctos más allá de los detalles intrínsecos de cada componente. Por lo tanto, es importante encontrar una manera de reducir o eliminar los errores que ocurren en cada etapa de procesamiento. Es por esto que se utiliza la abstracción digital.

Para poder distinguir entre señal y ruido, es necesario realizar un mapeo entre bits y señales utilizando un conjunto *discreto* de valores. La manera más simple de realizar esto es utilizando un esquema de mapeo *binario*, es decir, utilizar dos voltajes[2], V_0 y V_1, para representar los bits 0 y 1 respectivamente.

De esta manera, todas las señales recibidas cuyos voltajes sean valores cercanos a V_0 serán interpretadas como 0, mientras que las señales recibidas con valores cercanos a V_1 serán interpretadas como 1. Para lograr que el mapeo funcione de manera confiable, la estrategia más razonable consiste en separar V_0 y V_1 de tal manera que incluso una señal ruidosa sea interpretada de manera correcta.

La idea es intuitiva: se define el valor intermedio V_{th} como $\frac{V_0+V_1}{2}$ y todos los voltajes recibidos que sean menores o iguales a V_{th} se interpretan como el bit 0, mientras que todos los otros voltajes recibidos se interpretan como el bit 1.[3]

2.1.1. Transmisión de señales

Cada señal individual se puede pensar como una onda con voltaje fijo mantenida por algún período de tiempo. Por lo tanto, para enviar un bit 0 se transmitirá una señal

[1] Existen muchas razones por las cuales la salida de un enlace difiere de su entrada, tales como la tolerancia de sus componentes internos, factores del entorno (temperatura, voltaje suministrado) y factores externos (interferencia de otros transmisores cercanos, etc.). Todas estas fuentes pueden ser definidas de manera colectiva como *ruido* o *distorsión*.

[2] El voltaje es una magnitud física que cuantifica la diferencia de potencial eléctrico entre dos puntos.

[3] Extraído de [4].

con un voltaje constante de V_0 por un cierto periodo de tiempo, y para enviar un bit 1 se hará lo mismo con V_1. Estas señales de tiempo continuo se representan utilizando muestras discretas, en donde la *tasa de muestreo* se define como el número de muestras por segundo que se utiliza en el sistema. Tanto el transmisor como el receptor deben acordar la tasa de muestreo con anterioridad. El recíproco de la tasa de muestreo es el *intervalo de muestreo*.

Sobre un enlace de comunicaciones, el transmisor y el receptor deben acordar la tasa del reloj. De esta manera, eventos periódicos serán temporizados por un reloj. Cada nuevo bit es enviado cuando ocurre una transición en el reloj, y por cada bit se envían varias muestras, a una tasa regular. Por lo tanto, ¿cómo hace el receptor para recuperar los datos enviados, siendo que el transmisor solamente envía las muestras sin el reloj?

La idea es que el receptor infiera la presencia de un tic del reloj cada vez que hay una transición en el valor de las muestras recibidas. Luego, puesto que existe una tasa de muestreo acordada previamente, el receptor puede deducir la posición del resto de los tics del reloj, obteniendo la primera y última muestra para cada bit. Para lograr mayor robustez, es posible elegir la muestra del medio para determinar el bit del mensaje, o promediar todas las muestras recibidas.

Para que este enfoque funcione, existen dos problemas que deben ser resueltos:

1. ¿Cómo lidiar con las diferencias entre las frecuencias de los relojes del transmisor y del receptor?
2. ¿Cómo asegurar que existe una cantidad suficiente de transiciones entre 0s y 1s?

2.1.2. Recuperar el reloj y los datos

Tanto el transmisor como el receptor utilizan un reloj interno que corre a la tasa de muestreo para determinar cuándo generar o adquirir la siguiente muestra. Además, ambos guardan un índice de muestreo que contiene la posición de la muestra actual, así como contadores para almacenar la cantidad de muestras que hay en cada bit. El problema es que las frecuencias de los relojes utilizados por el transmisor y el receptor podrían no ser exactamente iguales. Esta pequeña diferencia se acumula con el tiempo, por lo que si el receptor utiliza una estrategia de muestreo estático como la descrita anteriormente, eventualmente ocurrirá que se estará realizando el muestreo exactamente en el punto de transición entre dos bits consecutivos. Además, la diferencia entre las frecuencias de reloj suele cambiar con el tiempo.

La solución consiste en hacer que el receptor adapte su temporización en base a las transiciones que detecta en las muestras recibidas. La transición debería ocurrir exactamente a la mitad entre dos puntos de muestreo consecutivos, por lo que si el

receptor determina que no existe una transición en ese punto, debería ajustar su índice de muestreo de forma apropiada.

Existen dos situaciones posibles:

1. La muestra recibida a la mitad de los puntos de muestreo es igual a la muestra actual. En este caso se está muestreando demasiado tarde, por lo que al tomar la próxima muestra, el índice de muestreo debería incrementarse en la cantidad de muestras por bit menos uno.

2. La muestra recibida a la mitad de los puntos de muestreo es distinta a la muestra actual. En este caso se está muestreando demasiado temprano, por lo que al tomar la próxima muestra, el índice de muestreo debería incrementarse en la cantidad de muestras por bit más uno.

Si no hay una transición en el punto medio, el índice de muestreo se incrementa en la cantidad de muestras por bit.

2.1.3. Codificación en línea

La codificación en línea fue desarrollada para resolver dos problemas. En primer lugar, por razones eléctricas, es deseable mantener un balance en el cable, por lo que en promedio el número de 0s debería ser igual al de 1s.

En segundo lugar, las transiciones entre bits recibidos indican el comienzo de un nuevo bit, y son útiles para sincronizar el proceso de muestreo en el receptor[4]. Por lo tanto, las transiciones frecuentes son deseables. Por otra parte, cada transición consume energía, por lo que sería deseable minimizar el número de transiciones, siendo consistente con los requisitos anteriores y con la necesidad de transmitir información.

Para resolver estos problemas, se puede utilizar un codificador en el transmisor para transformar los bits del mensaje a una secuencia que tenga las propiedades descritas anteriormente, y un decodificador en el receptor para recuperar el mensaje original.

2.1.4. Modelos para la comunicación sobre canales físicos

El primer paso para transmitir una secuencia de bits es convertir la misma en una señal digitalizada, discreta en el tiempo. La unidad de medida de esta señal ya no son los bits, sino las muestras.

Distintas señales pueden transmitir distinta cantidad de muestras por unidad de tiempo. A su vez, se puede elegir codificar cada bit para que sea representado por una

[4] Cuanto mejor sea la sincronización, más alta será la tasa de símbolos posible.

cantidad arbitraria de muestras en su correspondiente señal digital. Estas dos variables determinan la capacidad de transmisión (en bits por unidad de tiempo) que soporta determinada señal discreta.

A esta señal discreta y digital, producto de la codificación de una secuencia de bits, se le denomina *señal base (baseband signal)*. Si bien esta señal representa correctamente la información codificada, no es práctica para ser transmitida sobre un canal de comunicación físico por varias razones. Algunas de ellas son:

1. La señal base es una señal discreta. Típicamente, los canales de comunicación físicos transmiten señales continuas en el tiempo, por lo que esta señal debe convertirse a una señal análoga.

2. En la mayoría de los casos, el canal de comunicación sobre el cual se desea transmitir no soporta algunas de las propiedades que posee la señal generada.

3. Regulaciones gubernamentales no permiten transmitir en cualquier frecuencia.

Por estas razones, se estila utilizar una nueva señal, llamada *carrier* (portadora). La principal propiedad del carrier es que es una señal apta para ser transmitida a través del canal de comunicación físico en cuestión. Se dice que se *modula* esta última señal, cuando se perturban distintas propiedades de la misma con el objetivo de transmitir la información codificada en la señal base. Las propiedades que pueden perturbarse o modificarse son la amplitud, la frecuencia y la fase de la señal. Existen técnicas de modulación que modifican cada una de estas propiedades e incluso las combinan para lograr transmitir información en forma más rápida, pero a su vez, lo más robustamente posible; esto es, que la modulación sea tolerante a errores en la transmisión producto del ruido del entorno y demás factores.

Del lado del receptor, lo que se recibe es la señal carrier modulada, y se quiere recuperar la secuencia de bits original. Los pasos a seguir son los inversos a la transmisión, por lo que primero se debe obtener la señal base a partir de la señal modulada para luego decodificar cada bit. Al proceso de convertir la señal carrier modulada en la señal base se le llama *demodulación*.

El escenario ideal sería que la señal base demodulada en el receptor fuera completamente idéntica a la señal que se moduló del lado del transmisor. Esto en la realidad no se cumple por diversas razones, como por ejemplo el ruido sobre el canal de comunicación, o por propiedades físicas inherentes al canal que no permiten hacer una transición instantánea entre valores consecutivos que se estén modulando, resultando en una señal más relajada que al demodular afecta la señal base demodulada. De esta forma, la señal base se encuentra transformada en el receptor, y se puede abstraer este fenómeno obviando algunos detalles de lo que sucede en la realidad y modelar la transformación de la señal base de la forma que se describe en la Figura 2.1, donde:

- $X[n]$ es la señal base original que representa la secuencia de bits que se desea transmitir.
- $Y[n]$ es la señal base demodulada del lado del receptor.

En este modelo, la señal base sufre una transformación S al ser transmitida. Este comportamiento que presentan los canales de transmisión se puede modelar de forma bastante precisa utilizando modelos lineales e invariantes en el tiempo.

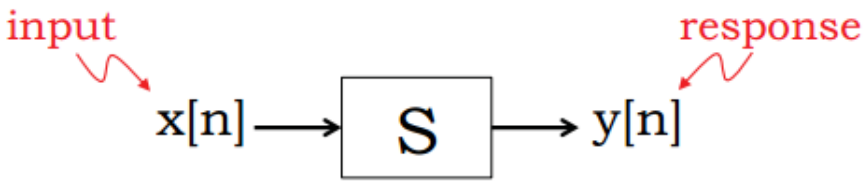

Figura 2.1: Transmisión de información del transmisor al receptor. Extraído de [4], Capítulo 10.

Que un sistema sea lineal significa que satisface el *principio de superposición*, que engloba las propiedades de proporcionalidad o escalado y aditividad. Que sea proporcional significa que cuando la entrada de un sistema es multiplicada por un factor, su salida también será multiplicada por el mismo factor. Por otro lado, que un sistema sea aditivo significa que si la entrada es el resultado de la suma de dos entradas, la salida será la resultante de la suma de las salidas que producirían cada una de esas entradas individualmente.

Por su parte, un sistema se dice invariante en el tiempo si dada una señal $x[t]$ cuya respuesta es $y[t]$, se puede mover esa señal en el tiempo hacia el futuro o hacia el pasado (sumando o restando un entero a t), y la respuesta en el instante t movido sigue siendo la misma que antes de realizar el desplazamiento.

A modo de resumen, muchas veces el canal de comunicación no soporta la transmisión de la señal base como se tiene, o es inviable transmitir determinada señal base por el tamaño de la antena que requeriría recibir esa señal, o también puede ser que no sea posible transmitir determinada señal porque viola aspectos regulatorios del gobierno. Por estas razones, hay que buscar otra forma de poder transmitir esa señal, a través de un canal de comunicación dado.

Un concepto muy relacionado con la modulación de una señal es el de multiplexado de frecuencias en un canal de comunicación. Esto significa hacer uso de un canal para transmitir múltiples señales al mismo tiempo y lograr que las mismas no interfieran unas con otras.

En el caso del aire, por ejemplo, en todo momento existen muchas señales compartiendo el canal de comunicación. El receptor no recibe una única señal del transmisor, sino una señal compuesta por señales generadas por distintos transmisores, entre ellos,

el que al receptor le interesa. Las dos principales razones por las cuales funciona el multiplexado de señales en un canal de comunicación como el aire son:

1. Cualquier señal base puede ser descompuesta en una sumatoria de sinusoides utilizando descomposición de Fourier. Para modular estas sinusoides, se multiplica la señal base al carrier. El resultado es una señal cuya frecuencia tiene una desviación acotada con respecto a la frecuencia original de la señal carrier.

2. Asumiendo que el ruido es despreciable, como el canal de comunicación cumple las propiedades de los modelos lineales e invariantes en el tiempo, se puede obtener de la señal recibida el valor de la señal en determinada frecuencia, aplicando filtros apropiados.

Esto está asegurado por la propiedad de linealidad, que garantiza que si una señal está formada por una combinación de sinusoides, la señal recibida por el receptor será también una combinación de sinusoides donde cada una es la señal recibida de cada sinusoide que conforma la señal origen.

Para poder conceptualizar mejor el proceso de modulación es necesario entender el espacio de señales. Así como en el plano uno puede describir cualquier vector como una combinación lineal de los vectores $(1, 0)$ y $(0, 1)$, lo mismo sucede en el espacio de señales. Este conjunto de elementos a partir del cual se puede construir todo el resto se llama conjunto básico o conjunto elemental. La propiedad principal que deben cumplir las funciones base para que puedan pertenecer a este conjunto es la de ortogonalidad entre ellas, que se define formalmente de la siguiente manera:

$$\int_{-\infty}^{+\infty} \phi_i(t)\phi_j(t)\,dt = \begin{cases} 1 & \text{i} = \text{j} \\ 0 & \text{i} \neq \text{j} \end{cases}, \tag{2.1}$$

donde tanto ϕ_i como ϕ_j son funciones del conjunto elemental. En el caso del espacio de señales, estas funciones son el seno y el coseno, que claramente cumplen el principio de ortogonalidad.

De esta manera surge una nueva forma de representar una señal. Se definen las señales I y Q, formadas a partir del conjunto elemental (el coseno y el seno respectivamente) y que combinadas representan cualquier señal del espacio de señales. Esta división no es solo conceptual. A nivel de hardware, las señales son generadas por separado y combinadas más adelante. Esto presenta varias ventajas, como por ejemplo un diseño de hardware simple. Tanto la energía de la señal como la fase se pueden representar a partir de estas señales base.

2.1.5. Diagramas de constelación

Los *diagramas de constelación* son una herramienta útil para visualizar gráficamente las componentes I y Q que forman la señal que se está analizando. El diagrama representa la señal en el plano complejo, siendo I el eje horizontal y Q el eje vertical. A medida que se muestrea la señal, se agrega un punto al diagrama por cada muestra, por lo que eventualmente pueden haber puntos superpuestos. La Figura 2.2 muestra un ejemplo de diagrama de constelación.

Los diagramas de constelación son siempre a nivel de señal base, o lo que es lo mismo, cuando la frecuencia del carrier es 0.

2.1.6. Tipos de modulación

Se presentan en esta sección los tipos de modulación más comunes que se utilizan en la comunicación de datos. La información sobre modulaciones fue extraída de [3].

- **Modulación por amplitud de la señal (AM o ASK)**: Este tipo de modulación se centra en modificar la amplitud de la señal a lo largo del tiempo para transmitir información.

  Como ejemplo de modulación por amplitud, se tiene OOK (*On-Off Keying*). Se trata del tipo de modulación más básica posible. En caso que se quiera transmitir un bit con valor 1, se mantiene la señal con una amplitud fija A, y cuando se quiere transmitir un bit cuyo valor es 0, se transmite la señal con amplitud 0.

- **Modulación por frecuencia de la señal (FM o FSK)**: Se modifica la frecuencia de la señal a lo largo de tiempo para transmitir información.

- **Modulación por fase de la señal (PSK)**: Se modifica la fase de la señal a lo largo de tiempo para transmitir información. Ejemplos de modulación por fase son:

  - *Binary Phase Shift Keying (BPSK)*: Solo se puede representar un bit por cada símbolo de la onda. Se modula utilizando como carrier únicamente al coseno, es decir, la componente Q del carrier tiene amplitud nula. Por esto mismo, su diagrama de constelación solo presentará puntos a lo largo del eje X de un plano cartesiano.
  - *Quadrature Phase Shift Keying (QPSK)*: Se pueden representar dos bits por símbolo. Es equivalente a BPSK, pero en este caso se utilizan ambas componentes del carrier, tanto en I como en Q. La gran ventaja que tiene sobre BPSK es que duplica la cantidad de información enviada por símbolo, sin hacer que el algoritmo sea más vulnerable al ruido, ya que la diferencia es que se utiliza una componente que es ortogonal a la primera, que estaba

siendo desperdiciada en BPSK. El diagrama de constelación para QPSK se muestra en la Figura 2.2.

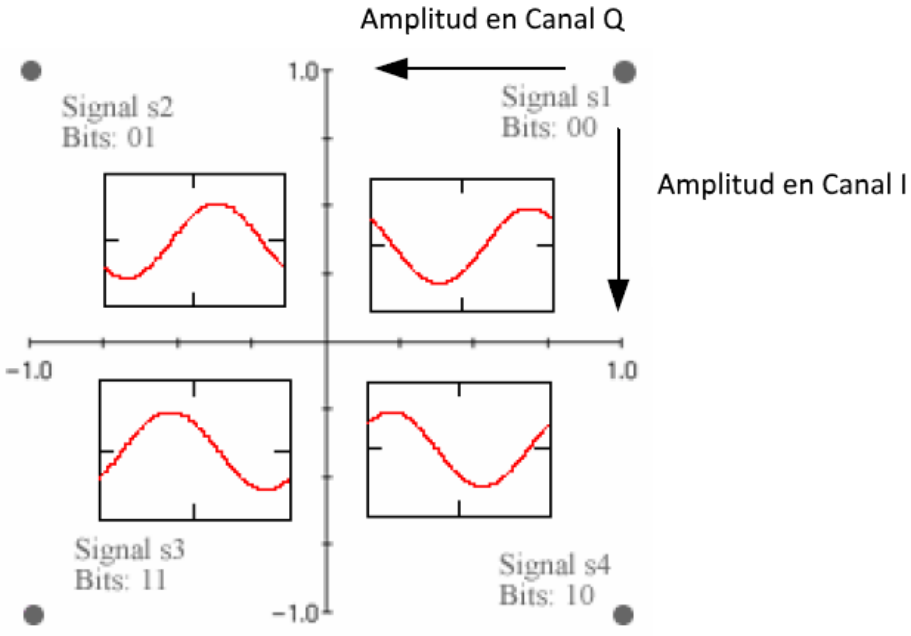

Figura 2.2: Diagrama de constelación de QPSK. Extraído de [3].

  - *8-PSK, 16-PSK...*: Se puede seguir dividiendo el espacio de los valores que puede tomar la fase. A diferencia del caso de QPSK, en estos casos el hecho de agregar más información sí hace que la modulación sea más susceptible a errores producidos por el ruido en el canal de comunicación.

- **Modulaciones mixtas**: Se combinan técnicas de modulación haciendo variar más de un parámetro de la sinusoide. Ejemplos de este tipo de modulación son 16QAM y 64QAM.

- **Modulaciones que no utilizan carriers**: Modulaciones que transmiten información directamente utilizando la señal base. Un ejemplo de este tipo es *Pulse Coded Modulation (PCM)*. No se ahondará en este tipo de modulaciones en este proyecto.

2.2. Códigos convolucionales

Dos de los objetivos principales que se persiguen al diseñar redes de comunicación digital son la confiabilidad y la eficacia del sistema en su conjunto. Para alcanzar estas metas, es necesario utilizar técnicas para lidiar con errores de bit, que son inevitables en cualquier canal de comunicación.

La idea clave a aplicar para alcanzar una comunicación confiable es la adición de *redundancia*, de forma tal de aumentar la probabilidad de que el mensaje original pueda ser reconstruido por el receptor a partir de datos que pueden estar posiblemente corruptos. El transmisor utiliza un *codificador* sobre los datos originales para producir bits codificados, que son enviados por el canal de comunicación. Al recibirlos, el receptor utiliza un *decodificador* para obtener la *mejor estimación posible* de los datos originales. Si el código utilizado es bueno, es posible tanto detectar como corregir errores para recuperar los datos originales.

La detección de errores permite que el receptor pueda determinar, con una probabilidad muy alta, si el mensaje fue decodificado correctamente o no. Para permitir una detección más potente, se suele utilizar un código de detección de errores que se encuentra separado del que se utiliza para corregir errores. Se suelen utilizar códigos que calculan un resumen (*digest*) de un mensaje completo, tales como funciones de hash o códigos de redundancia cíclica (CRC). Estos códigos toman un mensaje de tamaño n bits y producen una firma compacta de tamaño fijo (por ejemplo, 32 bits en el caso del esquema CRC-32 utilizado por el estándar Wi-Fi). A continuación, transmiten el mensaje junto con su firma. El receptor puede, luego de decodificar el mensaje para realizar corrección de errores, calcular la firma de los bits del mensaje y compararla con la firma recibida; si las firmas difieren, entonces el receptor puede afirmar que ha habido un error en uno o más bits. Si las firmas son iguales, puede afirmar con una probabilidad alta que el mensaje recibido es igual al enviado. En el caso de que las firmas sean buenas, esta probabilidad es muy alta.

Esta sección se centrará en los códigos de corrección de errores utilizados por el estándar Wi-Fi. Dichos códigos, llamados *códigos convolucionales*, son utilizados en una variedad amplia de sistemas inalámbricos, así como también en comunicaciones por satélite. La gran ventaja de esta clase de códigos es que existe una forma eficiente de decodificarlos para recuperar el mensaje que tuvo mayor probabilidad de haber sido enviado.

Los códigos convolucionales involucran el cálculo de *bits de paridad* sobre los bits del mensaje. Los cálculos de paridad consisten en utilizar funciones lineales[5] sobre los bits que conforman el mensaje, de forma tal de generar la redundancia a enviar. Sin embargo, en lugar de enviar bits de redundancia junto con los datos originales, el transmisor envía *solamente los bits de paridad*.

2.2.1. Construcción de los códigos convolucionales

El transmisor utiliza una *ventana deslizante* para calcular $r > 1$ bits de paridad combinando ciertos subconjuntos de bits de la ventana. Un ejemplo de esto se muestra

[5] Por esta razón, los códigos que involucran cálculos de paridad son *lineales*. Esto significa que cada bit de paridad es computado como la suma ponderada de uno o más bits del mensaje original. La mayoría de los códigos utilizados en la práctica son lineales.

en la Figura 2.3. Las operaciones de combinación utilizadas son sumas módulo 2 (es decir, equivalentes al *or-exclusivo*). La ventana deslizante se mueve de a un bit, lo que significa que distintas ventanas pueden superponerse. El tamaño de la ventana es importante, y se denota con la letra K (mayúscula). Cuanto más grande es el tamaño de la ventana, más son los bits de paridad influenciados por un bit de mensaje dado. Puesto que los bits de paridad son los únicos enviados por el canal, un tamaño de ventana grande generalmente implica una resistencia mayor a errores de bit. Sin embargo, un tamaño de ventana mayor también implica un tiempo de decodificación mayor.

Un código convolucional que produzca r bits de paridad por ventana y que deslice la misma hacia adelante un bit por vez tendrá una tasa efectiva de $1/r$. Cuanto más grande sea el valor de r, más resistencia a errores tendrá el código, pero también se dedicará un mayor ancho de banda para enviar datos redundantes. En la práctica, se intenta elegir valores de r y de tamaño de ventana lo más pequeño posibles, y que mantengan baja la probabilidad de errores de bit.

El proceso final es simple. El transmisor puede ver hasta K bits en un momento dado, y producir r bits de paridad según las funciones de cálculo, escogidas previamente, que operan sobre subconjuntos de los K bits. La Figura 2.3 muestra un ejemplo en donde $K = 3$ y $r = 2$ (la tasa del código es $1/r = 1/2$). El transmisor genera r bits y los envía de manera secuencial, desliza la ventana hacia la derecha un bit y luego repite el proceso.

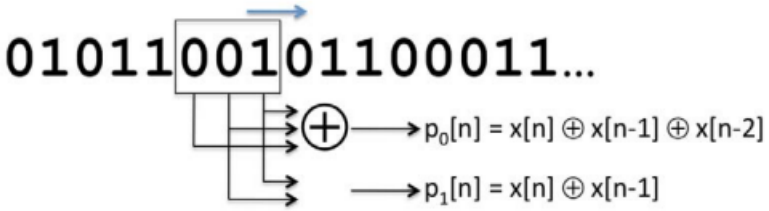

Figura 2.3: Ejemplo de un código convolucional con dos bits de paridad por cada bit de mensaje y un tamaño de ventana de tres. Extraído de [4], Capítulo 7.

Esto define un conjunto de *ecuaciones de paridad*, que determinan la forma en que son producidos los bits de paridad a partir de la secuencia de bits de mensaje. En general, es posible ver cada ecuación de paridad como producida por la combinación de los bits del mensaje y un polinomio generador g. En el ejemplo anterior, los coeficientes del polinomio generador son $(1, 1, 1)$ y $(1, 1, 0)$.

Denotando g_i al polinomio generador del bit de paridad p_i, es posible definir los bits de paridad $p_i[n]$ de la siguiente manera:

$$p_i[n] = (\sum_{j=0}^{k-1} g_i[j]x[n-j]) \tag{2.2}$$

La Ecuación 2.2 es una *convolución matemática* entre g y x (módulo 2); de ahí el nombre "código convolucional". El número de polinomios generadores es igual al número de bits de paridad generados, r, en cada ventana deslizante.

2.2.2. Codificador convolucional

Existen dos formas de ver al codificador convolucional que son útiles a la hora de implementar los procedimientos de codificación y decodificación. La primera es en términos de *registros de desplazamiento* (*shift*), puesto que es posible construir el mecanismo de codificación conectando distintos registros de desplazamiento. Esta vista es útil para implementar codificadores a nivel de hardware. La segunda es en términos de una *máquina de estados*, que se corresponde a la vista de un codificador como un conjunto de estados con transiciones bien definidas entre ellos. Esta vista es extremadamente útil para deducir cómo decodificar un conjunto de bits de paridad y reconstruir el mensaje original.

La Figura 2.4 muestra el mismo codificador de la Figura 2.3 en la forma de diagrama de bloques, utilizando registros de desplazamiento. Los valores $x[n-1]$ y $x[n-2]$ componen el *estado* del codificador. El diagrama muestra cómo se toma un bit a la vez ($x[n]$) y se generan dos bits de paridad de salida ($p_1[n]$ y $p_2[n]$). El cálculo se realiza utilizando tanto el bit de entrada como los que están guardados en el estado del codificador. Luego de que se producen r bits, el estado del codificador se desplaza un lugar a la derecha: $x[n]$ toma el lugar de $x[n-1]$ y $x[n-1]$ el de $x[n-2]$. Este diagrama se puede implementar directamente en hardware con registros de desplazamiento.

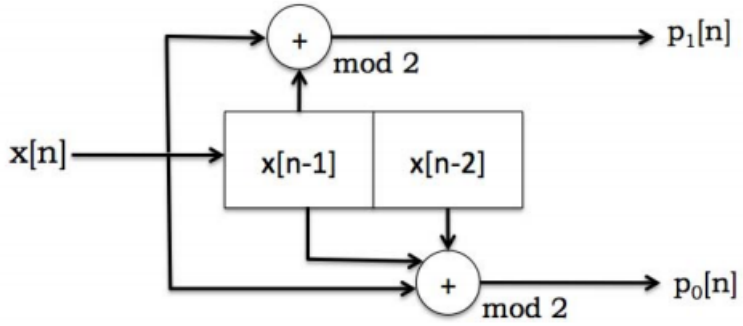

Figura 2.4: Vista del proceso de codificación convolucional utilizando registros de desplazamiento. Extraído de [4], Capítulo 7.

La otra forma útil de ver a los códigos convolucionales es como una máquina de estados, tal como se muestra en la Figura 2.5 (que continúa con el ejemplo de la Figura 2.3). Es importante destacar que la máquina de estados para un código convolucional es idéntica para todos los códigos que tengan un mismo tamaño de ventana K y que el número de estados es siempre 2^{K-1}. Solamente las etiquetas p_i cambian dependiendo

del número de generadores polinomiales y de los valores de sus coeficientes. Cada estado se etiqueta con $x[n-1]x[n-2]\dots x[n-K+1]$. Por su parte, cada arco se etiqueta con $x[n]/p_0p_1\dots$.

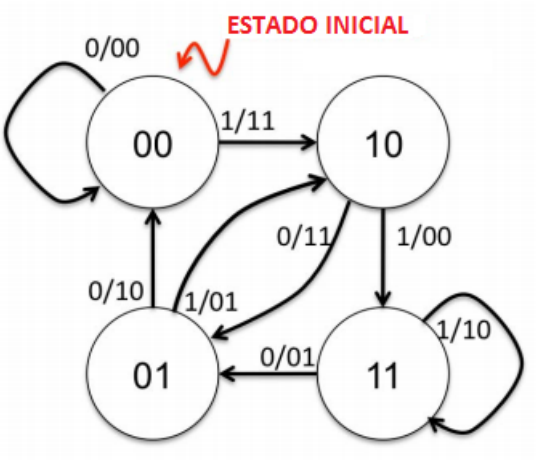

Figura 2.5: Vista del proceso de codificación convolucional mediante una máquina de estados. Extraído de [4], Capítulo 7.

La vista de máquina de estados ofrece una forma elegante de explicar qué es lo que hace el transmisor, y qué es lo que debe hacer el receptor para decodificar el mensaje. El transmisor comienza en el estado inicial y procesa el mensaje un bit a la vez. Por cada bit de mensaje, hace las transiciones de estado desde el estado actual al siguiente dependiendo del valor del bit de la entrada, y envía los bits de paridad que se encuentran en el arco correspondiente.

El receptor no tiene conocimiento directo de las transiciones de estado que llevó a cabo el transmisor al enviar el mensaje, sino que solamente conoce la secuencia de bits de paridad recibidos (posiblemente con errores). Su tarea es determinar la mejor secuencia de estados posible que pudo haber producido la secuencia de bits de paridad que recibió. Existen muchas formas de definir "mejor secuencia de estados posible", pero una que es especialmente atractiva es la de *secuencia de estados con mayor probabilidad de haber sido enviada por el transmisor*. Un decodificador que sea capaz de realizar esta tarea se llama *decodificador de máxima verosimilitud* (*maximum-likelihood decoder*), o decodificador ML, como se lo llamará de aquí en adelante.

Una propiedad muy útil que se cumple para todos los códigos convolucionales[6] es que el decodificador ML puede hallarse computando el código (válido) que tenga la menor *distancia de Hamming*[7] con el código recibido, y devolviendo el mensaje que

[6] Esta propiedad no solo se cumple para los códigos convolucionales, sino también para otra clase de códigos llamada *códigos de bloque*, independientemente de que el código sea lineal o no.

[7] La *distancia de Hamming* entre dos códigos se define como el número de posiciones de bit en que dichos códigos difieren.

haya generado dicho código. Por lo tanto, la secuencia de bits de paridad con mayor probabilidad de haber sido enviada debe ser aquella que guarde la menor distancia de Hamming con la secuencia recibida.

Determinar el código válido más cercano a un código recibido es una tarea difícil de realizar en la práctica (especialmente para códigos convolucionales), puesto que la cantidad de códigos posibles es *infinita* (o al menos tan larga como se desee). Esto significa que el procedimiento de recorrer la lista completa de posibles mensajes transmitidos es intratable desde el punto de vista computacional, y hace necesario encontrar un mecanismo que permita recorrer el espacio de posibilidades y encontrar el mensaje válido (aquel con distancia de Hamming más pequeña) de manera rápida. La resolución de este problema, que es además el corazón del *decodificador de Viterbi*, utiliza una estructura derivada de la máquina de estados llamada *trellis*, que se describe a continuación.

La vista de máquina de estados permite conocer qué ocurre en cada instante de tiempo cuando el transmisor tiene un bit de mensaje para procesar, pero no permite conocer cómo el sistema evoluciona en el tiempo. La *trellis* es una estructura que hace explícita esta evolución; un ejemplo se muestra en la Figura 2.6. Cada columna de la trellis tiene el conjunto de estados; cada estado en una columna se conecta a dos estados en la siguiente columna (los mismos dos estados conectados en la máquina de estados). El arco superior de cada estado muestra qué se transmite si el bit actual es un 0, mientras que el arco inferior muestra qué se transmite ante un 1. La imagen muestra el recorrido en la trellis para el mensaje 101100.

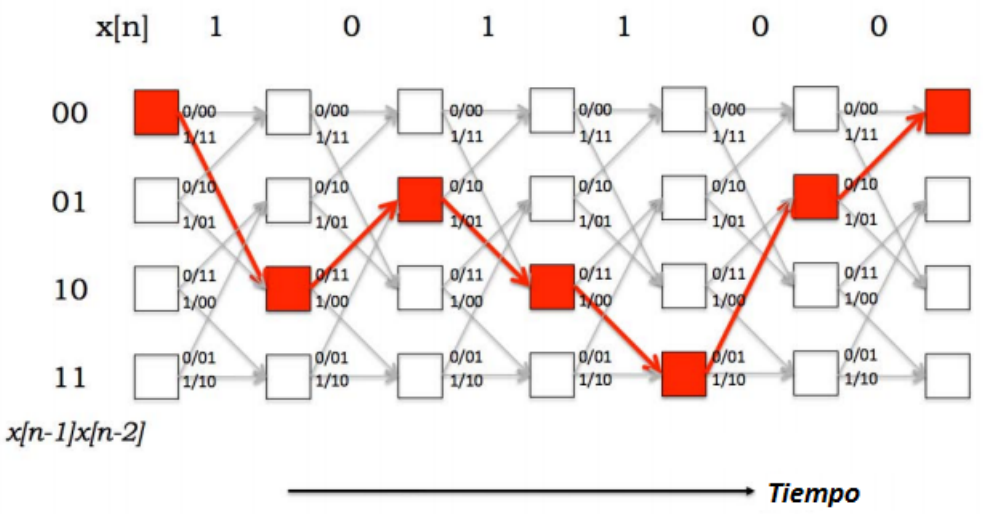

Figura 2.6: Recorrido en la trellis para el mensaje 101100. Extraído de [4], Capítulo 7.

Esta herramienta ofrece una nueva manera de pensar en la tarea del decodificador. El mismo obtiene una secuencia de bits de paridad, y debe determinar el mejor camino

a través de la trellis—es decir, la secuencia de estados que pueda explicar la cadena de bits de paridad recibida, posiblemente corrupta.

2.2.3. Decodificación de los códigos convolucionales

En esta sección se describe una manera elegante y eficiente de decodificar códigos convolucionales. Este método, que evita enumerar de manera explícita las 2^N posibles secuencias de N bits de paridad, fue inventado por A. Viterbi en 1967[8] y lleva su nombre.

Como se mencionó en la sección anterior, la trellis es una herramienta muy útil para entender el procedimiento de decodificación de códigos convolucionales. Suponiendo que se tiene la trellis completa para un código dado, si se recibe una secuencia de bits y no hay errores de paridad entonces debe existir un camino a través de los estados de la trellis que sea igual a la secuencia recibida. Ese camino (más precisamente, la concatenación de bits de paridad enviados según las aristas recorridas) se corresponde con los bits de paridad enviados. Partiendo de ahí, es fácil obtener el mensaje original puesto que el arco superior que nace de un nodo dado de la trellis se corresponde con un 0, y el inferior con un 1.

En presencia de errores de bit, es atractivo encontrar el mensaje con mayor chance de haber sido transmitido, puesto que se minimiza la probabilidad de que existan errores de bit al decodificar. Si existiera una forma de capturar los errores introducidos al ir de un estado al siguiente, entonces sería posible acumular dichos errores a lo largo de un camino y estimar el total de errores cometidos. El camino con la menor acumulación de errores es el buscado, y el mensaje transmitido puede ser determinado fácilmente al resolver la concatenación de estados explicada anteriormente.

El decodificador de Viterbi utiliza dos métricas para realizar esta tarea: la métrica de rama (*branch metric*, BM) y la métrica de camino (*path metric*, PM). La métrica de rama es una medida de la distancia de Hamming entre lo que fue transmitido y lo recibido, y es definido para cada arco de la trellis. La Figura 2.7 muestra un ejemplo donde se reciben los bits 00. Para cada transición de estado, el número en el arco muestra la métrica de rama para la transición. Dos son cero, correspondiendo a los únicos estados y transiciones donde la distancia de Hamming es cero. Las otras ramas que no son cero se corresponden a los casos que tienen errores de bit.

La métrica de camino es un valor asociado con un estado de la trellis que se corresponde con la distancia de Hamming entre la secuencia de bits de paridad recibida y el camino más probable desde el estado inicial al actual en la trellis, donde "más probable" se entiende como el camino con la menor distancia de Hamming entre el estado inicial y el actual, medida sobre todos los posibles caminos entre los dos estados. El camino

[8] Para más información, ver [7].

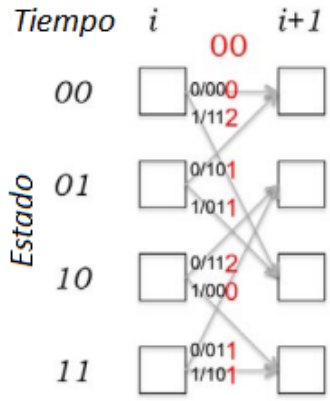

Figura 2.7: La métrica de rama, BM. En este ejemplo, el receptor recibe los bits de paridad 00. Extraído de [4], Capítulo 8.

con la distancia de Hamming más pequeña minimiza el total de errores de bit, y es el más probable cuando la tasa de errores de bit es baja.

La idea clave del algoritmo de Viterbi es que el receptor puede computar la métrica de camino (para un estado en un momento dado) incrementalmente, utilizando las métricas previamente computadas de estados anteriores en conjunto con las métricas de rama.

Computando la métrica de camino

Supóngase que el receptor tiene computadas las métricas de camino $PM[s, i]$ para cada estado s en el tiempo i. El valor de $PM[s, i]$ es el número total de errores de bit detectados al comparar los bits de paridad recibidos con el mensaje que tiene la mayor probabilidad de haber sido transmitido (si se consideran todos los mensajes que pudieron haber sido enviados por el transmisor hasta el tiempo i, empezando desde el estado 00).

De entre todos los estados posibles en el tiempo i, el estado más probable es aquel con la menor métrica de camino. Si hay más de un estado que cumpla esto, todos son posibilidades igualmente buenas.

Si el transmisor se encuentra en el estado s en el tiempo $i + 1$, entonces necesariamente *debe haber estado en uno de dos estados posibles en el tiempo* i. Estos dos estados previos, etiquetados α y β, son siempre los mismos para un estado dado. De hecho, solamente dependen de la longitud de ventana K (y no de las funciones de paridad). La Figura 2.7 muestra los estados previos de cada estado. Por ejemplo, para el estado 00, $\alpha = 00$ y $\beta = 01$, mientras que para el estado 01, $\alpha = 10$ y $\beta = 11$.

Cualquier secuencia que deje al transmisor en el estado s en el tiempo $i + 1$ debe haber dejado el transmisor en el estado α o β en el tiempo i. Por ejemplo, en la

Figura 2.7, para llegar al estado 01 en el tiempo $i+1$ debe cumplirse una de las dos propiedades que se listan a continuación:

1. El transmisor se encontraba en el estado 10 en el tiempo i, y el i-ésimo bit del mensaje fue un cero. Si ese es el caso, entonces el mensaje transmitido fue 11 y hubo dos errores de bit, porque se recibieron los bits 00. Entonces, la métrica de camino del nuevo estado, $PM[01, i+1]$, es igual a $PM[10, i] + 2$, puesto que el nuevo estado es 01, y la métrica de camino crece en dos porque hubo dos errores.

2. El transmisor se encontraba en el estado 11 en el tiempo i y el i-ésimo bit del mensaje fue un cero. Si ese es el caso, entonces el mensaje transmitido fue 01 y entonces hubo un error de bit, porque se recibieron los bits 00. La métrica de camino del nuevo estado, $PM[01, i+1]$ es igual a $PM[11, i] + 1$.

En resumen:

$$PM[s, i+1] = min(PM[\alpha, i] + BM[\alpha \rightarrow s], PM[\beta, i] + BM[\beta \rightarrow s]), \tag{2.3}$$

en donde α y β son los dos estados previos.

En el algoritmo de decodificación, es importante recordar qué arco se corresponde con el mínimo, puesto que es necesario recorrer este camino desde el estado final hasta el inicial manteniendo guardados los arcos que fueron recorridos, para luego finalmente revertir el orden de los bits para producir el mensaje más probable.

Selección del camino más probable

Se procede a describir cómo hace el decodificador para encontrar el camino más probable. Inicialmente, el estado 00 tiene un costo 0 y los otros $2^{K-1} - 1$ estados tienen un costo de ∞.

El bucle principal del algoritmo consiste en dos pasos principales: primero, calcular la métrica de rama para el próximo conjunto de bits de paridad, y segundo, computar la métrica de camino para la siguiente columna (de la trellis). El cálculo de la métrica de camino consiste en:

1. *Sumar* la métrica de rama del estado anterior.

2. *Comparar* las sumas para los caminos que llegan al nuevo estado. Existen solamente dos caminos posibles a comparar en cada nuevo estado debido a que solamente hay dos aristas incidentes desde la columna previa.

3. *Seleccionar* el camino con el valor más pequeño, rompiendo empates de manera arbitraria. Este camino es el que tiene menos errores.

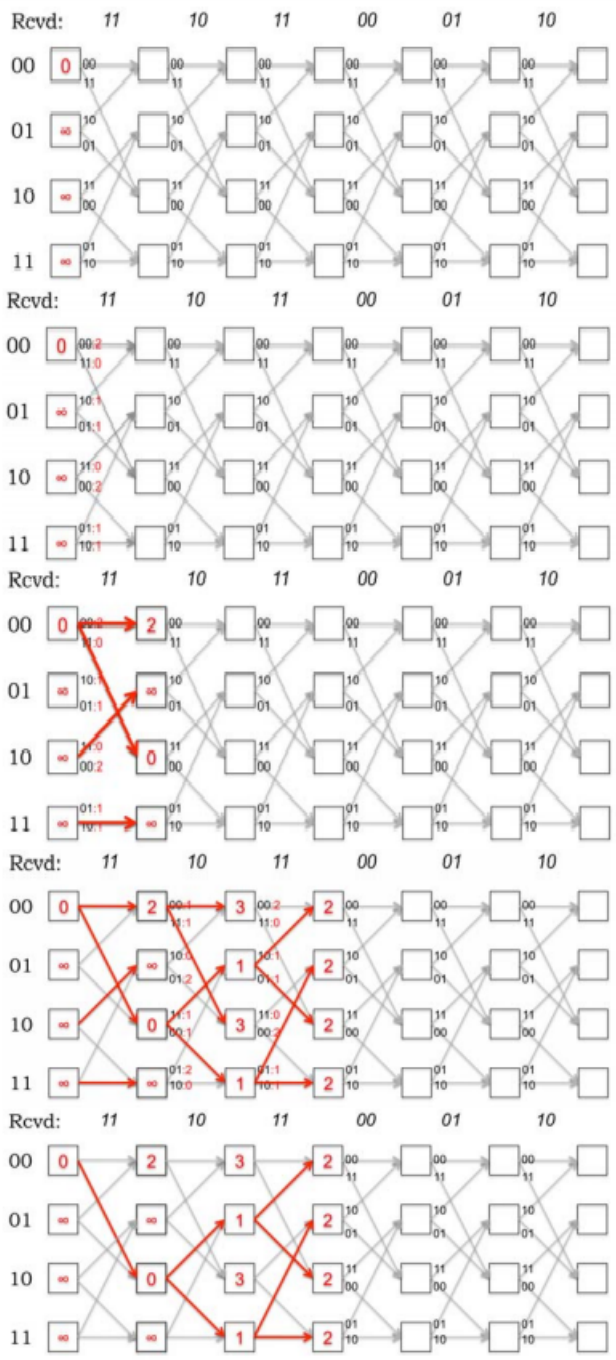

Figura 2.8: Funcionamiento del decodificador de Viterbi, para cuatro momentos en el tiempo. La última imagen muestra solamente los caminos sobrevivientes. Extraído de [4], Capítulo 8.

La Figura 2.8 muestra el funcionamiento del algoritmo de Viterbi. Este ejemplo describe cómo la secuencia de bits 11 10 11 00 01 10 es procesada por el receptor. Dentro de esta figura, el cuarto grafo (contando desde arriba) muestra todos los estados que

tienen la misma métrica de camino. En este punto, cualquiera de estos cuatro estados (con los caminos que llegan a ellos) son secuencias de bit con mayor probabilidad de haber sido transmitidas, todas con una distancia de Hamming de 2. Un camino sobreviviente es aquel con mayor probabilidad (un camino ML); hay muchos otros caminos que pueden ser eliminados puesto que no hay forma de que sean caminos ML. La razón por la cual el algoritmo de Viterbi es práctico es que el número de caminos sobrevivientes es muchísimo más pequeño que el número total de caminos en la trellis.

Otro punto importante es que el conocimiento futuro permite romper cualquier tipo de empate, causando que ciertos caminos, que habían sido considerados como de mayor probabilidad, sean eliminados a la luz de la nueva información.

2.2.4. Puncturing

Tal como se describió anteriormente, la tasa máxima que un código convolucional puede alcanzar es de $1/r$, donde r es el número de bits de paridad producidos por cada bit del mensaje original. Para obtener tasas más elevadas, existe una técnica llamada *puncturing*[9], en la cual el transmisor no envía todos los bits de paridad producidos, sino un subconjunto de los mismos. Este subconjunto debe haber sido acordado previamente entre transmisor y receptor.

Por ejemplo, en el caso de un código de tasa 1/2, es posible elegir 101 como patrón de puncturing para el primer flujo de bits de paridad, y 110 para el segundo. La notación significa que el codificador enviará el primer y tercer bit del primer flujo, pero no el segundo, y enviará el primer y el segundo bit del segundo flujo, pero no el tercero. Por lo tanto, mientras que el codificador hubiera enviado dos bits de paridad por cada bit de mensaje sin *puncturing*, ahora se enviarían cuatro bits de paridad (en lugar de seis) por cada tres bits de mensaje, dando lugar a una tasa de 3/4. Es decir, el transmisor emitiría $p_0[0]p_1[0]p_0[1]p_1[1]p_0[2]p_1[2]$ sin puncturing, y $p_0[0]p_1[0] - p_1[1]p_0[2]-$ con él (donde – indica la ausencia de bits).

En el decodificador, los bits de paridad ausentes no participan en el cálculo de las métricas de rama al utilizar *puncturing*. Más allá de esto, el procedimiento es el mismo.

2.3. Multiplexación por División de Frecuencias Ortogonales (OFDM)

Todas las formas de modulación presentadas hasta el momento tienen la particularidad de que utilizan una única señal como carrier. Esto significa que en cada momento existe un carrier con una fase y amplitud en particular. Este tipo de modulaciones son

[9] Puncturing podría traducirse como "punción" o "perforación".

Ejemplo	Tasa de símbolo (Msps)	Tiempo de símbolo (μsec)	Distancia de path (m)
Ciudad	0.1	10	3300
Campus universitario	1	1	330
Edificio	10	0.1	33
Habitación	100	0.01	3

Tabla 2.1: Tasa de datos vs. tamaño del área en la cual no hay multipath. Extraído de [2].

versátiles y simples de implementar, pero adolecen de limitaciones importantes al ser utilizadas en un enlace de tipo inalámbrico.

Normalmente, en una comunicación típica entre dos antenas (una transmisora y otra receptora) existen diferentes tipos de obstáculos que pueden reflejar la señal emitida, provocando que ésta recorra un camino distinto al de la línea recta que une las dos antenas. Esto significa que, en general, la señal emitida por el transmisor recorrerá diversos caminos antes de llegar al receptor. Esto es conocido como *multipath*. Debido a que el tiempo que requieren dos señales que van por dos caminos alternativos es generalmente distinto, el receptor tendrá un serio problema al sumar las señales recibidas si el retraso es comparable al tiempo de símbolo (*symbol time*) y si la potencia de las señales no es muy distinta. Este fenómeno, conocido como *interferencia entre símbolos* (*intersymbol interference*), puede provocar que la señal recibida en el receptor sea completamente distinta a la señal transmitida.

El tipo de distorsión generada por el multipath es un factor a tener en cuenta cuando el retraso de propagación es comparable con los tiempos de símbolo. La Tabla 2.1 muestra la distancia que, para una cierta tasa de datos, puede ser cubierta sin que el multipath sea un problema grave. Es interesante observar que es posible transmitir a una tasa de 10 Msps en un edificio de tamaño razonable utilizando las modulaciones presentadas hasta ahora, pero para obtener tasas más altas se hace necesario un enfoque diferente.

Una técnica interesante para resolver el problema del multipath permitiendo tasas de datos más altas sobre áreas más grandes es la utilización de un tipo de modulación especializado, *Orthogonal Frequency-Division Modulation* (OFDM). OFDM es utilizado en los estándares 802.11a y 802.11g, razón por la cual se incluye en esta sección.

OFDM utiliza tres métodos básicos para resolver el problema del multipath. El primero de ellos es el paralelismo: enviar una señal de alta velocidad dividiéndola en múltiples señales de baja velocidad que son enviadas en paralelo. Los canales paralelos surgen de utilizar señales carrier con diversas frecuencias, una para cada canal, tal como se muestra en la Figura 2.9.

Si este arreglo es implementado de la forma tradicional, existen dos problemas. En primer lugar, el costo del equipamiento se incrementa de manera lineal con el número

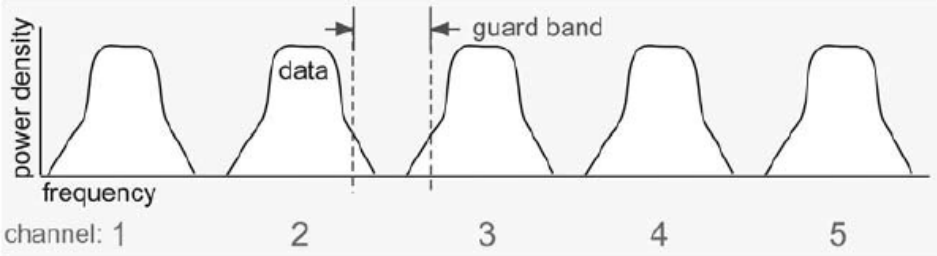

Figura 2.9: Multiplexado convencional de división de frecuencia para subcarriers. Extraído de [2].

de canales paralelos o *subcarriers*; en segundo lugar, parte del espectro asignado se desperdicia debido a la necesidad de que existan bandas de guarda (*guard bands*) entre los subcarriers para permitir que el receptor pueda filtrar la señal del subcarrier correcto.

Para resolver este problema se aplican los otros dos métodos que se encuentran en el corazón de OFDM. El primero de ellos consiste en explotar la ortogonalidad de las diferentes frecuencias de una manera más sutil. Tal como se explicó anteriormente, es conocido que diferentes frecuencias son ortogonales al ser integradas sobre un largo periodo de tiempo. Sin embargo, si el intervalo de tiempo contiene un número entero de ciclos de cada frecuencia, las dos son exactamente ortogonales.

$$\int_0^T \cos(2\pi[\frac{n}{T}]t)\cos(2\pi[\frac{m}{T}])\,dt = 0 \text{ para } m \neq n \tag{2.4}$$

En la Ecuación 2.4, ambos cosenos tienen, para el tiempo T, $\frac{nT}{T} = n$ y $\frac{mT}{T} = m$ ciclos completos de oscilación. Si T es elegido como el tiempo de símbolo, se pueden enviar símbolos consistentes en una amplitud y una fase para cada frecuencia en paralelo a las demás frecuencias que estén espaciadas por enteros y las frecuencias continuarán siendo ortogonales entre sí, siempre y cuando el tiempo de integración sea el correcto. El resultado de este esquema es la eliminación de las bandas de guarda, permitiendo una utilización mucho más eficiente del espectro asignado.

De esta manera, utilizando subcarriers que estén lo suficientemente cerca, se hace posible dividir una secuencia única de símbolos en varias secuencias paralelas de símbolos cuya tasa es más lenta. Es posible alcanzar un mejor resultado si se reduce levemente la tasa de datos, utilizando el tiempo de símbolo como un intervalo de guarda para mejorar la resistencia contra la interferencia entre símbolos.

Si se puede conocer exactamente dónde se encuentra el límite multipath, el intervalo de guarda podría ser implementado apagando la señal en el momento adecuado. Sin embargo, cualquier error al determinar el borde del intervalo causaría una integración sobre un intervalo parcial de símbolos, por lo que la ortogonalidad se perdería, produciéndose interferencia entre los símbolos paralelos. Para esto es que se utiliza una

extensión cíclica del símbolo original. La porción de cada subcarrier que ocurre durante el período de tiempo correspondiente al intervalo de guarda simplemente se repite al final del intervalo base. Puesto que todos los subcarriers son periódicos en el tiempo T, esto es el equivalente a realizar la misma operación sobre el símbolo final, que es la suma de todos los subcarriers. El símbolo resultante preserva la ortogonalidad de sus subcarriers sobre el intervalo T incluso si el inicio del intervalo se encuentra apenas fuera de lugar, puesto que la porción que se quita del principio es agregada hacia el final. Esto se ilustra en la Figura 2.10.

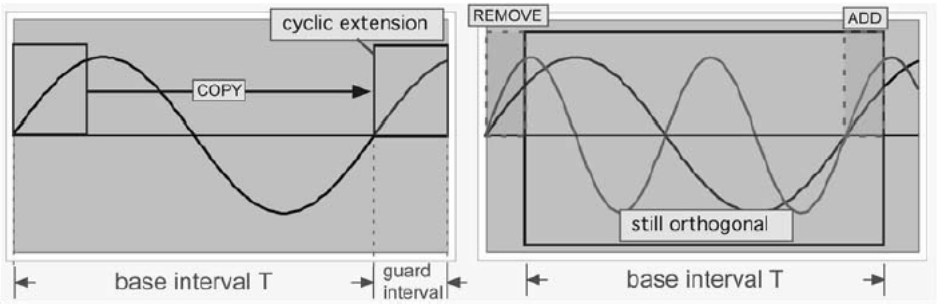

Figura 2.10: La extensión cíclica de un símbolo en OFDM hace que la integración no cambie, incluso ante la existencia de pequeñas desviaciones de tiempo. Extraído de [2].

2.3.1. Transformada rápida de Fourier

El uso del paralelismo y de un intervalo de guarda utilizando una extensión cíclica de los símbolos permite transmitir un grupo de símbolos sin distorsión a una tasa de datos alta aún con la existencia de distorsión multipath. Solo una radio es requerida para transmitir y recibir las señales. Sin embargo, dicha radio aún tiene un desafío importante: para obtener una mejora sustancial sobre la resistencia multipath, es necesario utilizar una gran cantidad de subcarriers. Para extraer cada subcarrier, es necesario integrar el producto de los símbolos recibidos y la frecuencia deseada, y el número de puntos de integración a utilizar está dado por la frecuencia más alta del subcarrier a la que se transmite. Para N subcarriers, parecería que fueran necesarias N integraciones con N puntos cada una, es decir, N^2 multiplicaciones y sumas. Para un N grande, esto se hace prohibitivo.

Afortunadamente, existe un último artilugio que permite resolver este problema: la *transformada rápida de Fourier (Fast Fourier Transform, FFT)*. Los algoritmos FFT pueden realizar la integración deseada en aproximadamente $N \log(N)$ operaciones. Para un N grande, esto representa una mejora sustancial con respecto a N^2: para 802.11a, el problema es reducido a 400 operaciones en lugar de 4000.

Para entender cómo funciona la FFT, es necesario examinar con más detalle el problema de extraer una aproximación a la transformada de Fourier de una señal muestreada en el dominio del tiempo a intervalos discretos. La operación a realizar se

muestra de manera esquemática en la Figura 2.11 donde se desea convertir N muestras ($N = 16$) en el tiempo a un estimado de la transformada de Fourier de la señal en los puntos $k\delta$, en donde $\delta = \frac{1}{N\tau}, -\frac{N}{2} \leq k \leq \frac{N}{2}$.

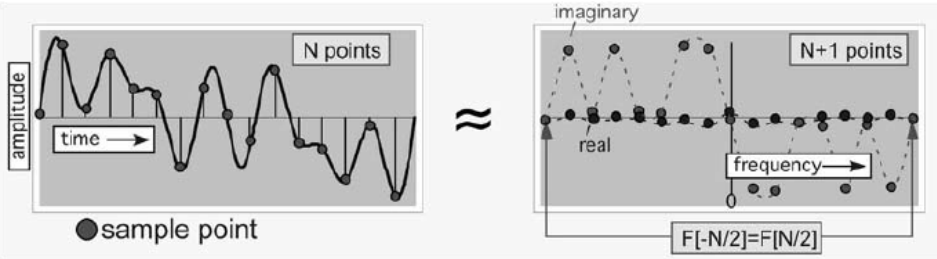

Figura 2.11: Transformada discreta de Fourier de una señal muestreada en el tiempo. Extraído de [2].

Es fácil ver por qué el número de frecuencias está limitado al número de muestras. Un seno o un coseno con una frecuencia de $N+1$ tiene los mismos valores en los puntos de muestreo que uno con frecuencia δ : $\cos[2\pi(N+1)\delta\tau] = \cos[\frac{2(N+1)\tau}{N\tau}] = \cos[\frac{2\pi(1+1)}{N}] = \cos[2\pi + 2\pi\delta\tau] = \cos(2\pi\delta\tau)$. Este fenómeno es conocido como *aliasing*. Por el mismo argumento, los valores son los mismos en $k = \pm\frac{N}{2}$.

En la Ecuación 2.5 se muestra el desarrollo básico para la transformada de Fourier. La transformada de Fourier es aproximada mediante una suma, en donde el n-ésimo punto de muestreo, $f(n)$, está multiplicado por el valor de la exponencial para la frecuencia de interés, k. Al sustituir el incremento de frecuencia, se deduce que la suma es independiente tanto del incremento como del incremento de frecuencia δ, y está determinado únicamente por el valor de N y f para un valor dado de k. La parte entre paréntesis es llamada usualmente *transformada discreta de Fourier*, con el factor τ añadido posteriormente para ajustar el tiempo y la escala de frecuencias.

$$\begin{aligned} F[k\delta] &= \tau \sum_n e^{-2\pi i n k \delta\tau} f(n) = \tau[f(0) + e^{-2\pi i k \delta t} f(1) + e^{-(2\pi)2ik\delta\tau} f(2) + \ldots] \\ &= \tau \sum_n e^{-2\pi i n k (\frac{1}{N\tau})\tau} f(n) = \tau \sum_n e^{-2\pi i k (\frac{n}{N}) f(n)} \\ &= \tau[f(0) + e^{-2\pi i k (\frac{1}{N})} f(1) + e^{-2\pi i k (\frac{2}{N})} f(2) + \ldots] \end{aligned} \tag{2.5}$$

La Figura 2.12 muestra estas sumas en forma tabular para el caso de $N = 4$. La transformada para la frecuencia normalizada k es obtenida al sumar todos los términos de la k-ésima fila de la tabla. El índice k normalmente varía entre 0 y $N - 1$; por el *aliasing*, la fila $k = 2$ es la misma que la fila $k = -2$ y la fila $k = 3$ es la misma que la $k = -1$.

Es importante notar que la suma de cada fila puede ser reescrita de manera sutil, agrupando los términos para los cuales n es par e impar, respectivamente. La Figura 2.13 muestra el reacomodo de los términos. A pesar que aún no se ha ahorrado en

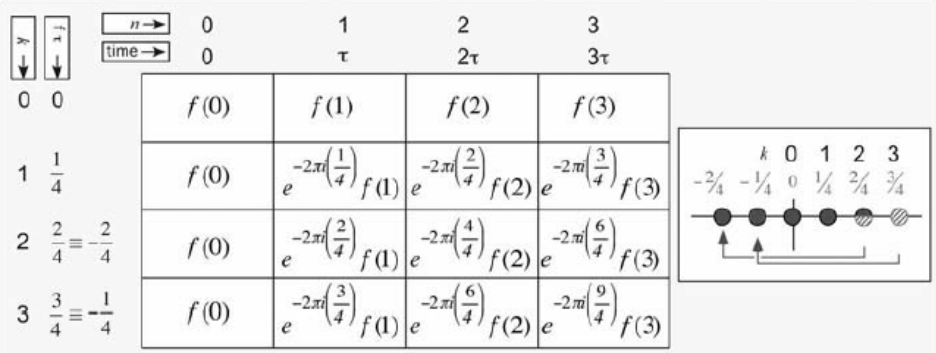

Figura 2.12: Cálculo de la transformada discreta de Fourier para $N = 4$. Extraído de [2].

ningún cálculo, es importante notar que luego del reordenamiento los términos entre paréntesis se ven como transformadas con $N = 2$.

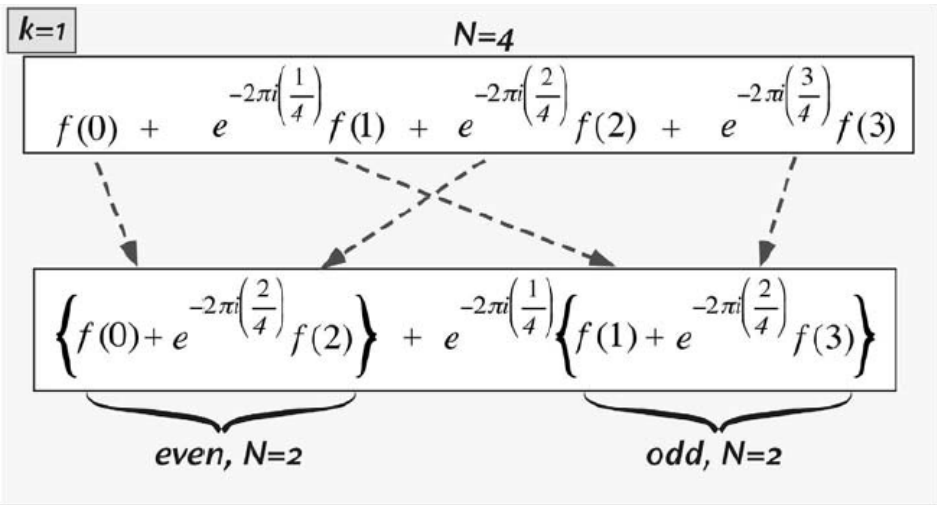

Figura 2.13: Reagrupación de la suma por cada fila en la transformada. Extraído de [2].

Es aquí donde aparece la utilidad de realizar este reacomodo. En la Figura 2.14 se puede apreciar la suma de las filas impares, $N = 1, 3$. En la parte de la izquierda, la suma de cada fila está escrita de manera recursiva como la suma de dos transformadas de orden $N = 2$. Cada una de estas puede ser a su vez reescrita como la suma de dos transformadas de orden $N = 1$. Pero se nota que la transformada $F_{2E}, k = 1$ es exactamente la misma que la transformada $F_{2E}, k = 3$: tan solo los factores multiplicadores son diferentes. Lo mismo ocurre para las transformadas impares. Por lo tanto, como se muestra en la parte derecha de la Figura 2.14, solamente se necesitan la mitad de los cálculos que en el método tradicional. El mismo argumento puede ser utilizado para las filas con $k = 0, 2$.

En general, puesto que el enfoque es recursivo, se agregan N operaciones adicionales por cada vez que se duplica el número de puntos: la complejidad del proceso final es

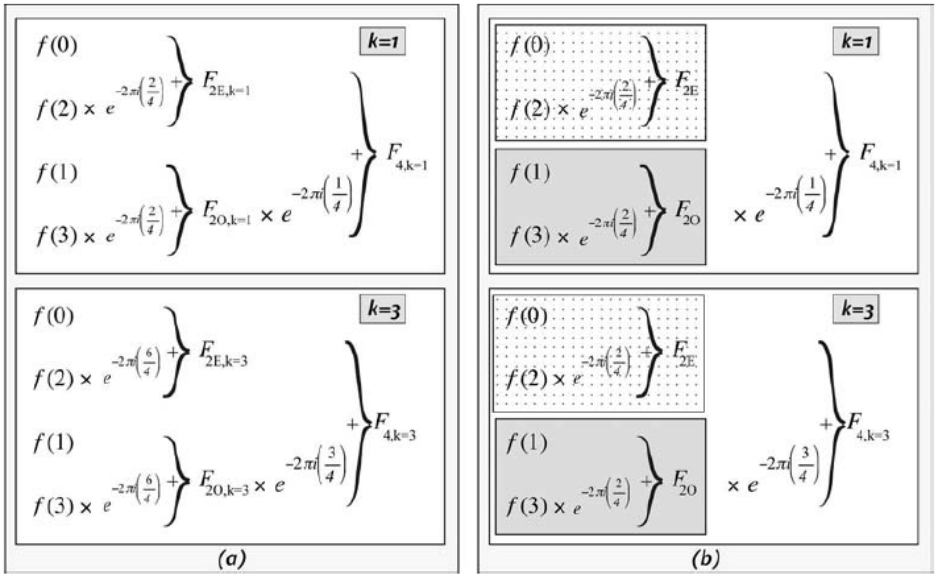

Figura 2.14: Vista recursiva de la Transformada Discreta de Fourier. Extraído de [2].

aproximadamente $N\log(N)$ en lugar de N^2.

Además de esto, muchas de las multiplicaciones son en realidad cambios de signo, por lo que las multiplicaciones pueden ser reducidas a un número pequeño de factores no triviales para un N cualquiera.

2.3.2. Construcción de un símbolo OFDM

Al manejar los conceptos básicos de la transformada rápida de Fourier, se está en condiciones de explicar cómo se realiza la construcción de un símbolo para OFDM:

- Se comienza con un conjunto de datos serial, agrupado de la manera apropiada para la modulación. Por ejemplo, si se utilizará QPSK para cada subcarrier, los datos de entrada deberían estar agrupados dos bits por vez.
- Cada conjunto de datos se convierte a un número complejo que describe la amplitud y la fase del subcarrier, representando dicho número complejo la amplitud compleja del correspondiente subcarrier.
- Se toma a continuación la FFT inversa para convertir el espectro de frecuencias a una secuencia de muestras. Este conjunto de números se lee de manera serial, y es asignado a espacios de tiempo sucesivos.
- Los números complejos resultantes son convertidos a un par de voltajes por un conversor análogo-digital (*ADC, analog-to-digital converter*); la parte real determina el canal I mientras que la parte imaginaria determina el canal Q.

- Los voltajes I y Q son multiplicados, respectivamente, por un coseno y un seno a la frecuencia del carrier, y el resultado es sumado (luego de filtrarlo) para producir el voltaje final.

El símbolo recibido es demodulado de manera similar. Luego de extraer los canales I y Q mediante una multiplicación por un coseno y un seno, un convertidor digital-análogo produce una secuencia serial de muestras complejas, las cuales luego de eliminar el intervalo de guarda, son convertidas mediante una FFT a la amplitud compleja original en cada frecuencia.

La Figura 2.15 muestra el diagrama de un transmisor OFDM. Es apreciable que la mayoría del trabajo asociado con la modulación OFDM es realizada en el dominio digital; el procesamiento análogo es muy similar al que se utilizaría en una radio convencional. Esto es característico de los sistemas de comunicación avanzados: debido a la ley de Moore, las funciones digitales son cada vez más baratas en uso de energía y en costo, mientras que el costo y el tamaño de los componentes análogos no escala de la misma manera.

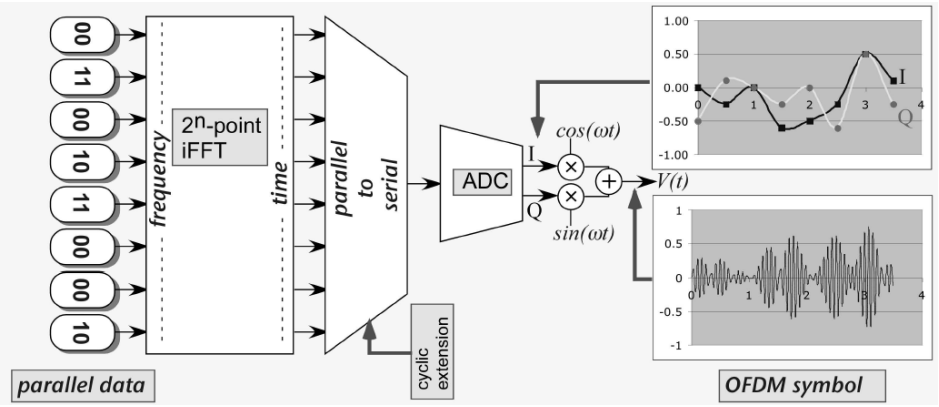

Figura 2.15: Representación esquemática de la construcción de un símbolo de OFDM para $N = 8$ subcarriers. Extraído de [2].

La definición original de la técnica de OFDM no requiere explícitamente de un mecanismo de corrección de errores para utilizar en la transmisión y recepción de datos. Sin embargo, en muchos casos se hace imprescindible contar con esta funcionalidad. Las implementaciones de OFDM que usan código de detección de errores se llaman *Coded OFDM* o COFDM. COFDM se caracteriza por utilizar corrección de errores del tipo *Forward Error Correction* (FEC), un sistema en el cual el receptor detecta bits corruptos y es capaz de recuperar el valor original de dichos bits, siempre y cuando la cantidad de bits a recuperar sea lo suficientemente pequeña.

Debido a que es natural que el algoritmo de corrección de errores modele la entrada como una secuencia de bits, las implementaciones comerciales de OFDM utilizan códigos convolucionales.

Capítulo 3

Estado del arte

3.1. Estándar Wi-Fi

En esta sección se presenta un resumen sobre el estándar Wi-Fi que se enfoca principalmente en la especificación de las tramas y en la capa física (que utiliza OFDM).

3.1.1. Estructura de la trama MAC IEEE 802.11

Cada trama está constituida por tres componentes básicos:

1. El campo *MAC Header* (Cabezal MAC), que contiene información de control (campo *Frame Control*, control de trama), duración y dirección, así como también datos opcionales que incluyen información de control de secuencia, de control de parámetros de calidad del servicio (*Quality of Service*) y de control de alto rendimiento (*High Throughput*) para algunas tramas.
2. El campo *Frame Body* (cuerpo de la trama), de longitud variable y que contiene información específica según el tipo y subtipo de la trama.
3. Un código de verificación por redundancia cíclica llamado *Frame Check Sequence (FCS*, secuencia de comprobación de trama).

La Figura 3.1 muestra de forma gráfica los distintos campos y su disposición dentro de una trama.

El orden del conjunto de campos mencionados es fijo. Sin embargo, no todas las tramas tienen la misma cantidad o los mismos tipos de campos presentes. Los primeros tres, *Frame Control*, *Duration* (duración) y *Address 1* (dirección 1), y el último, *Frame Check Sequence*, están presentes en todas las tramas, incluyendo a los tipos y subtipos

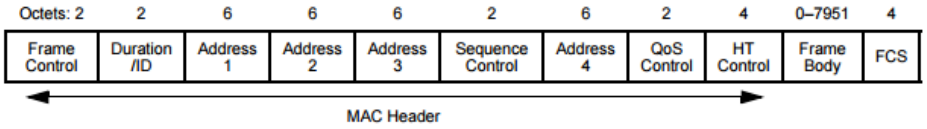

Figura 3.1: Representación gráfica del formato de trama de 802.11. Extraído de [8].

reservados. Es a este conjunto de campos que se le conoce como *formato de trama mínimo*.

Los campos *Address 2* (dirección 2), *Address 3* (dirección 3), *Sequence Control* (control de secuencia), *Address 4* (dirección 4), *QoS Control* (control de calidad del servicio), *HT Control* (Control de Alto Rendimiento) y *Frame Body* (cuerpo de la trama) solamente están presentes en algunos tipos y subtipos de tramas.

El resto de esta sección describe los campos de la trama MAC. Se omite la explicación de los campos *QoS Control* y *HT Control*, por no estar relacionados con los objetivos de este proyecto.

Campo Frame Control

El campo Frame Control contiene, entre otros, la versión del protocolo, el tipo y subtipo, información sobre fragmentación y protección de paquetes, y más campos de control. Su formato se encuentra ilustrado en la Figura 3.2.

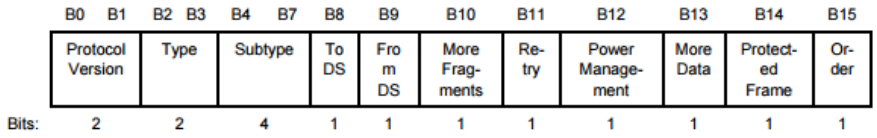

Figura 3.2: Representación gráfica del formato del campo Frame Control. Extraído de [8].

Los campos en cuestión son:

- **Versión de protocolo (campo *Protocol Version*)**: Indica la versión del protocolo utilizada por la trama. La versión es incrementada únicamente en el caso en que exista una incompatiblidad importante entre una nueva revisión del estándar y la anterior. Su longitud es de 2 bits y tiene un valor de 0 para la versión actual del estándar.

- **Tipo y subtipo (campos *Type* y *Subtype*)**: Estos campos permiten identificar la función de la trama. El estándar define tres tipos distintos de tramas: de gestión, de datos y de control. Dentro de cada una de estas categorías existen

distintos subtipos. A modo de ejemplo, dentro de las tramas de datos, hay algunas que implementan QoS y otras que no[1].

- **Hacia/desde DS[2] (campos *To DS/From DS*)**: Estos campos, de 1 bit cada uno, en combinación, permiten determinar desde y hacia dónde se dirige una trama.

- **Más fragmentos (campo *More Fragments*)**: El campo *More Fragments*, de 1 bit de longitud, se encuentra en 1 en todas aquellas tramas de datos o de gestión que tengan otro fragmento (del MSDU[3] o MMPDU[4] actual), y en 0 en las demás tramas.

- **Reintentar (campo *Retry*)**: El campo *Retry*, de 1 bit de longitud, se encuentra en 1 en todas aquellas tramas de datos o de gestión que sean una retransmisión de una trama anterior, y en 0 en las demás tramas. Las estaciones[5] receptoras hacen uso de esta indicación para identificar tramas duplicadas que deban eliminar.

- **Gestión de energía (campo *Power Management*)**: El campo *Power Management*, de 1 bit de longitud, se utiliza para indicar el modo de gestión de energía de una estación. El valor de este campo está o bien reservado, o se mantiene constante en cada trama de una estación particular durante una secuencia de intercambio de tramas. El valor indica el modo de la estación luego de que se completa con éxito la secuencia de intercambio de tramas.

- **Más datos (campo *More Data*)**: El campo *More Data*, de 1 bit de longitud, es utilizado para indicarle a una estación que se encuentre en modo de bajo consumo de energía de la recepción de nuevos paquetes[6] para dicha estación por parte del punto de acceso[7]. El campo *More Data* solamente es válido para las tramas de datos o gestión que sean direccionables de manera individual y transmitidas por un punto de acceso a una estación en modo de bajo consumo de energía. Un valor de 1 indica que hay presente al menos un nuevo paquete almacenado para la misma estación.

[1] Para ver la lista completa de los tipos de tramas definidos por estos campos, se puede consultar el estándar Wi-Fi (ver [8, p. 383]).

[2] DS, Distribution System, es el sistema utilizado para interconectar estaciones y redes de área local.

[3] MSDU, MAC Service Data Unit, es la información transmitida como una unidad entre dos puntos de acceso.

[4] MMPDU, MAC Management Protocol Data Unit, es la unidad de datos intercambiada entre dos entidades para implementar el protocolo de gestión de MAC.

[5] STA, Station, es una entidad lógica direccionable de manera única en el contexto de capas MAC y física.

[6] El estándar Wi-Fi se refiere a estos paquetes como *Bufferable Units*, que podría traducirse como unidades almacenables.

[7] AP, Access Point, es una entidad que provee acceso a los servicios de distribución para las estaciones asociadas.

- **Trama protegida (campo *Protected Frame*)**: El campo *Protected Frame*, de 1 bit de longitud, se coloca en 1 en el caso de que el cuerpo de la trama contenga información que ha sido procesada por un algoritmo de encapsulamiento criptográfico. El campo se encuentra en 1 solamente en tramas de datos, o en tramas de gestión que tengan el subtipo de autenticación, así como tramas de gestión robustas que sean direccionables de manera individual, y en 0 en todos los demás casos.

- **Orden (campo *Order*)**: El campo *Order*, de 1 bit de longitud, es utilizado para dos propósitos. En primer lugar, se coloca en 1 en tramas de datos que no sean QoS y que sean transmitidas por una estación que tampoco sea QoS para indicar que la trama contiene un MSDU o un fragmento que está siendo transmitido utilizando la clase de servicio de orden estricto (*StrictlyOrdered*). En segundo lugar, se coloca en 1 en tramas de datos o de gestión que sí sean de tipo QoS para indicar, en conjunto con otros campos, que la trama contiene un campo HT Control.

Campo de duración/ID (*Duration/ID*)

El campo *Duration/ID* tiene 16 bits de longitud. Su contenido varía dependiendo del tipo y subtipo de la trama, de la capacidades de QoS de la estación transmisora, y de si la transmisión se realiza durante el CFP[8].

El contenido del campo se define de la siguiente manera:

- En tramas de control con subtipo PS-Poll, el campo *Duration/ID* contiene el identificador de asociación de la estación que transmitió la trama en los 14 bits menos significativos, y los 2 bits más significativos en 1.

- En tramas transmitidas por estaciones no-QoS y por los PC[9] durante el CFP, el campo *Duration/ID* se coloca con el valor fijo de 32768.

- En todas las otras tramas transmitidas por estaciones no-QoS, y en tramas de control transmitidas por estaciones QoS, el campo *Duration/ID* contiene un valor de duración definido para cada tipo de trama en el estándar.

- En tramas de datos y de gestión que sean transmitidas por estaciones QoS, el campo *Duration/ID* contiene un valor de duración definido para cada tipo de trama en el estándar.

[8] El periodo sin contención (CFP, *Contention-Free Period*) es el período de tiempo durante el cual el derecho a transmitir es asignado a las estaciones, permitiendo que los intercambios de tramas se realicen sin que exista contienda por el medio inalámbrico.

[9] Un coordinador de punto (PC, *Point Coordinator*) es una entidad dentro de la estación en un punto de acceso que lleva a cabo la función de coordinación entre estaciones en un punto

Campos de dirección (*Address Fields*)

Existen cuatro campos de dirección definidos en el formato de la trama MAC. Estos campos se utilizan para indicar el identificador del BSS[10] (BSSID), la dirección de origen (SA, *Source Address*), dirección de destino (DA, *Destination Address*), dirección de la estación transmisora (TA, *Transmitting STA Address*), y dirección de la estación receptora (RA, *Receiving STA Address*). La longitud de los campos de dirección es de 48 bits. Además, pueden existir tramas que no contengan todos los campos.

Las direcciones MAC pueden pertenecer a alguna de las siguientes categorías:

- **Dirección individual**: Es la dirección asignada a una estación específica de la red.
- **Dirección de grupo**: Es una dirección multidestino, que puede ser usada por una o más estaciones de la red en forma simultánea. Dentro de esta categoría se distinguen dos subcategorías:
  - **Multicast**: Una dirección asociada a un conjunto de estaciones relacionadas lógicamente.
  - **Broadcast**: La dirección que define al conjunto de todas las estaciones que pertenecen a una misma red de área local.

Los campos de dirección posibles son los siguientes:

- **Campo BSSID**: El campo BSSID es un campo de 48 bits de longitud del mismo formato que las direcciones MAC IEEE 802.
- **Campo DA**: El campo DA contiene una dirección MAC IEEE individual o de grupo que identifica la o las entidades MAC que deberían ser los receptores finales del MSDU (o del fragmento) o A-MSDU[11] contenido en el cuerpo de la trama.
- **Campo SA**: El campo SA contiene una dirección MAC IEEE individual que identifica la entidad MAC desde la cual la transferencia del MSDU (o del fragmento) o A-MSDU contenido en el cuerpo de la trama fue iniciada.
- **Campo RA**: El campo RA contiene una dirección MAC IEEE individual que identifica la siguiente estación inmediata que debe ser la receptora de la información contenida en el cuerpo de la trama en el medio inalámbrico actual.

[10] BSS, Basic Service Set, es el conjunto de estaciones que se han sincronizado de manera exitosa utilizando las primitivas correspondientes definidas en el estándar IEEE 802.11.

[11] A-MSDU, Aggregate MSDU, es una estructura que contiene múltiples MSDUs transportados en una única trama de capa MAC.

- **Campo TA**: El campo TA contiene una dirección MAC IEEE individual que identifica la estación que ha transmitido al medio inalámbrico la MPDU[12] contenida en el cuerpo de la trama.

Campo de control de secuencia (*Sequence Control*)

El campo de control de secuencia tiene 16 bits de longitud y consiste en dos subcampos, el número de secuencia (*Sequence Number*) y el número de fragmento (*Fragment Number*). Este campo no se encuentra presente en tramas de control.

El número de secuencia, de 12 bits, indica el número de secuencia de un MSDU, A-MSDU o MMPDU. A cada MSDU, A-MSDU o MMPDU transmitido por una estación se le es asignado un número de secuencia (exceptuando las tramas de control). El número de secuencia se mantiene constante en todas las retransmisiones de un MSDU, MMPDU o fragmento de alguno de estos.

Por otra parte, el número de fragmento, de 4 bits, indica el número de fragmento de un MSDU o MMPDU. El número de fragmento se coloca en 0 en el primer o único fragmento y es incrementado en uno por cada fragmento sucesivo. Por otra parte, en el único fragmento de un A-MSDU también se coloca un 0. El número de fragmento se mantiene constante en todas las retransmisiones del fragmento.

Campo Frame Body

El campo *Frame Body* (cuerpo de la trama) es un campo de longitud variable que contiene información específica a la trama en cuestión. En el caso de una trama de datos, aquí se encontrarán los datos específicos que se quieren comunicar.

La longitud mínima de este campo es de cero bytes, mientras que su longitud máxima está determinada por el tamaño máximo de la MSDU, que es de 2304 bytes, más la longitud del campo *Mesh Control* (control de malla), de 6, 12 o 18 bytes en el caso en que esté presente, más el tamaño máximo de MMPDU sin encriptar (2304 bytes sin incluir el cabezal MAC ni el FCS) o el tamaño máximo de A-MSDU (3839 o 7935 bytes dependiendo de las capacidades de la estación actual), más cualquier contenido extra que se pueda deber a la encapsulación de tramas por mecanismos de seguridad.

Campo Frame Check Sequence

Este campo contiene un CRC de 32 bits, calculado a partir de todos los campos del cabezal MAC y del campo Frame Body.

[12] MPDU, MAC Protocol Data Unit, es la unidad de datos intercambiada entre dos entidades MAC usando los servicios de la capa física. MPDU es el sinónimo de "trama" que utiliza el estándar Wi-Fi.

3.1.2. Especificación de la capa física

En esta sección se detallan las características más importantes de la especificación de la capa física implementada en OFDM, tomando como principales referencias bibliográficas a [8] y [11].

Existen distintas implementaciones de la capa física para 802.11. En la primera revisión del estándar, publicada en el año 1997, se estandarizaron tres implementaciones, siendo estas Frequency-Hopping, Direct-Sequence e Infrared Light. En revisiones posteriores del estándar se agregaron nuevas implementaciones (entre ellas una que utiliza OFDM) y mejoras a implementaciones ya existentes.

Inicialmente se describirá la capa física de forma general, detallando en alto nivel las características que ésta presenta, independientemente de qué implementación particular se esté usando. Luego, se explicará con más detalle la implementación de capa física que utiliza OFDM para la transmisión de la señal en el medio.

Organización lógica

La capa física de 802.11 tiene dos funciones de protocolo bien diferenciadas, las cuales pueden ser clasificadas como subcapas dentro de la primera. Estas son la *Physical Layer Convergence Procedure* (PLCP) y la *Physical Medium Dependent* (PMD).

La capa de enlace se comunica directamente con la capa física a través de la PLCP, utilizando un *Service Access Point (SAP)* para realizar dicha comunicación. Es responsabilidad de la capa de enlace (también llamada capa MAC) indicar a la PLCP que debe preparar MPDUs para la transmisión. Asimismo, la capa PLCP entrega las tramas recibidas por la PMD a la capa de enlace. De esta forma, es natural pensar en la PLCP como el puente entre la capa MAC y las transmisiones de la señal en el aire. La existencia de esta capa permite un desacoplamiento importante entre la capa MAC y la PMD, permitiendo definir la capa de enlace de forma genérica independientemente de cómo la capa física transmite la señal en el medio, pudiendo ser mediante OFDM, transmisión de luz infrarroja, etcétera.

La PLCP agrega un preámbulo y cabezal a los MPDUs recibidos desde la capa MAC, que dependen de la implementación de capa física que se esté utilizando. Este cabezal y preámbulo contienen información que es imprescindible para los transmisores y receptores de capa física, ya que, por ejemplo, distintos métodos de modulación requieren de distintos preámbulos. A la trama resultante de agregar el preámbulo y cabezal a un MPDU, el estándar Wi-Fi la denomina *PLCP Protocol Data Unit (PPDU)*. El MPDU, a su vez, también puede ser llamado como *PLCP Service Data Unit (PSDU)*, y es típicamente referido de esta forma cuando se referencian operaciones de capa física en el estándar.

La PMD provee el servicio de transmisión y recepción de PPDUs a través del medio inálambrico. Es decir, se encarga de realizar la interfaz con el canal de comunicación y realizar la modulación y demodulación de las tramas transmitidas y recibidas respectivamente. Las subcapas PLCP y PMD también se comunican mediante primitivas a través de un SAP, de la misma forma que lo hace la capa MAC con la PLCP.

El estándar especifica tanto para la capa MAC como para la física entidades de gestión, que también se clasifican como subcapas dentro de la capa física. Estas son la *MAC Layer Management Entity (MLME)*, y la *Physical Layer Management Entity (PLME)*.

Ambas entidades proveen las interfaces necesarias para que una entidad externa, con el nombre de *Station Management Entity (SME)*, pueda invocar servicios de gestión. La SME es una entidad independiente de la capa física y MAC, y se puede pensar que reside en "los alrededores" de la MLME y PLME. Las funciones exactas de la SME no se encuentran especificadas en el estándar, aunque en general se la considera responsable de recolectar información de gestión y del estado de ambas Layer Management Entities, y además de configurar valores de parámetros específicos de cada capa.

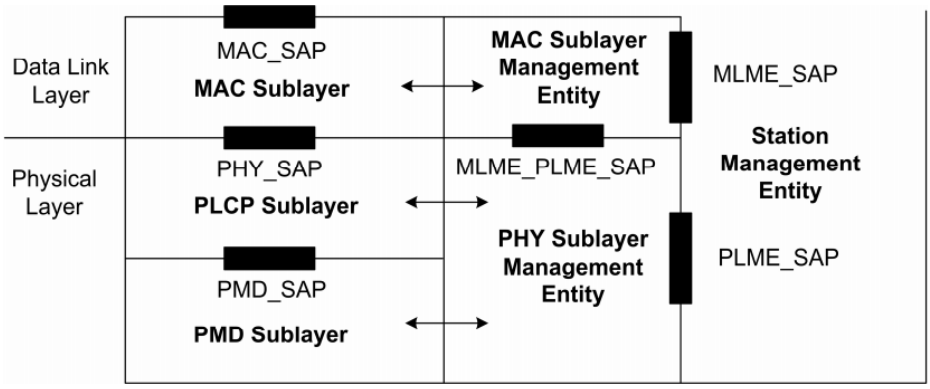

Figura 3.3: Representación gráfica de los componentes de las capas MAC y física. Se puede ver de forma clara que la comunicación entre las capas es a través de SAPs. También es posible observar la organización de las mismas, en donde la PLCP oficia de intermediaria entre la MAC y la PMD, y las subcapas de gestión se encuentran en el mismo nivel jerárquico que las capas que administran, mientras que la SME accede a todas las interfaces de gestión. Extraído de [8], Capítulo 4.

Operaciones principales

Las operaciones que realiza la capa física son muy similares en todas las implementaciones de la misma, por lo que se pueden definir de forma general. El estándar 802.11 utiliza máquinas de estado para describir estas operaciones. Cada máquina de estado realiza alguna de las siguientes funciones:

- ***Carrier Sense* y *Clear Channel Assessment* (CS/CCA)**: Estas funciones se encargan de determinar el estado del medio, y generalmente se las suele agrupar bajo el nombre de CS/CCA a pesar de ser dos operaciones distintas. Se ejecutan cuando el receptor se encuentra encendido y la estación no está transmitiendo ni recibiendo un paquete.

  La función de Carrier Sense permite detectar el inicio de una trama que está siendo recibida desde el medio. Esta función es implementada indicándole a la PMD que compruebe constantemente si el medio está ocupado o no. La capa de PLCP es la encargada de detectar el preámbulo PLCP a partir de la señal recibida por la PMD. Cuando el medio pasa a estar ocupado por una transmisión de otra estación, la PLCP intenta leer el preámbulo PLCP y el cabezal de capa física que tiene la trama para lograr sincronizar el receptor con la tasa de transferencia de datos de la señal.

  La función de Clear Channel Assessment permite evaluar si el canal está libre antes de realizar la transmisión de una trama. Si el medio está libre entonces la PLCP le envía una primitiva a la capa MAC del tipo *CCA.indicate* con el bit del campo de estado indicando que el canal está libre. De la misma forma, cuando el canal está ocupado, la PLCP envía la misma primitiva pero con el bit indicando que el canal está ocupado. Es responsabilidad de la capa MAC decidir si desea realizar la transferencia de una trama o no en base a la información proporcionada por la PLCP.

- **Transmisión**: Esta función se utiliza para enviar datos a través del medio. La operación es invocada por el PLCP CS/CCA luego de que se recibe un pedido desde la capa MAC de tipo PHY-TXSTART.request. La capa MAC utiliza el protocolo CSMA/CA con la ayuda de la operación de CS/CCA de la PLCP antes de indicar que se debe transmitir la trama.

  Una vez que se decide enviar la trama, la capa MAC envía el número de bytes de datos (que debe ser un valor entre 0 y 4095) y la tasa de transmisión. Luego, la PMD envía el preámbulo de la trama a la antena para que sea transmitido. La tasa de transferencia del preámbulo varía según la implementación de capa física que se esté utilizando. Una vez enviado el preámbulo se envía el cabezal, cuya tasa de transmisión también depende de la implementación de capa física utilizada. En el caso de OFDM, tanto el preámbulo como el cabezal de capa física se envían a una tasa de 6 Mbps.

  Una vez envíado el cabezal, el transmisor cambia la tasa de transferencia por la que se especifica en el cabezal del PSDU enviado. Una vez finalizada la transmisión del PSDU, la PLCP utiliza la primitiva TXSTEND.confirm para notificar a la capa MAC que la transmisión fue exitosa, y cambia el modo del PMD de transmisión a recepción.

- **Recepción**: Esta función se utiliza para recibir datos desde el medio. La opera-

ción también es invocada por el PLCP CS/CCA una vez que ésta última detecta una porción del preámbulo de sincronización seguido de un cabezal PLCP válido. De esta forma, lo que se recibe realmente, es decir, lo que procesa la operación de recepción, es únicamente la trama MAC, y no el cabezal PLCP ni el preámbulo, que son consumidos por la operación de CS/CCA.

La PMD indica que el medio está ocupado cuando detecta una señal con un poder de transmisión de al menos -85 dBm.

Cuando la PLCP verifica que el cabezal PLCP no contiene errores, le envía un mensaje a la capa MAC notificándole que se está recibiendo una nueva trama mediante la primitiva PHY-RXSTART.indicate. Luego, la PLCP inicializa en cero un contador encargado de controlar la cantidad de bytes de PSDUs recibidos, y a medida que la PLCP recibe bytes del PSDU, los envía a la capa MAC mediante la primitiva PHY-DATA.indicate. Una vez que el contador de bytes alcanza el valor que tenía el campo *length* del cabezal PLCP (es decir, se recibieron todos los bytes del PSDU), la PLCP utiliza la primitiva PHY-RXEND.indicate para indicar a la capa MAC que se terminó de recibir la trama.

3.1.3. Implementación de la capa física en OFDM

La capa física de OFDM organiza el espectro disponible en canales. Cada canal está formado por 52 subcarriers y tiene 20 MHz de ancho de banda. Cuatro de los subcarriers son utilizados para monitorear posibles interferencias entre subcarriers. Este tipo de interferencia tiene el nombre de *Inter Carrier Interference (ICI)*. Estos cuatro subcarriers son llamados *subcarriers piloto* y se encuentran de manera fija en las posiciones -21, -7, 7 y 21, y son modulados utilizando BPSK. Los otros 48 carriers son utilizados para transmitir datos. Los 52 subcarriers son numerados desde el -26 al 26 (el subcarrier cero no se utiliza).

El máximo teórico de datos que se puede transmitir en 802.11a es de 288 bits por canal. Sin embargo, la tasa real de transferencia es menor, debido a la redundancia introducida en la transmisión por el uso de códigos de detección y corrección de errores, así como todos los cabezales agregados con información de control. En la Tabla 3.1 se presenta el máximo teórico de transferencia en IEEE 802.11 dependiendo del esquema de modulación y la tasa de codificación elegida.

La implementación OFDM del estándar IEEE 802.11 es un COFDM[13], en el cual se recomienda fuertemente utilizar un decodificador de Viterbi para realizar la decodificación del código convolucional recibido.

Además de esto, se utilizan técnicas de reordenamiento por subcanales (*subchannel interleaving*), y de reordenamiento de tiempo (*time interleaving*) para mejorar la

[13] Coded OFDM (COFDM) es un tipo de implementación de OFDM en el cual se incluye algún tipo de código de corrección y detección de errores. Para más información, ver la Sección 2.3.2.

Modulación	Transferencia (Mbps)	Bits por subcarrier	Bits por símbolo	Bits de datos por símbolo
BPSK 1/2	6	1	48	24
BPSK 3/4	9	1	38	36
QPSK 1/2	12	2	96	48
QPSK 3/4	18	2	96	72
16QAM 1/2	24	4	192	96
16QAM 3/4	36	4	192	144
64QAM 2/3	48	6	288	192
64QAM 3/4	54	6	288	216

Tabla 3.1: Máximo teórico de transferencia de datos para cada esquema de modulación y tasa de codificación soportada en 802.11. Además, se muestra cuántos bits se codifican por cada subcarrier de datos, cuántos bits se codifican por símbolo, y cuántos bits de datos se codifican por símbolo. Esta última columna solo tiene en cuenta los bits de redundancia introducidos por el código convolucional al calcular los bits de datos por símbolo, lo que significa que el cabezal de la trama MAC se encuentra dentro de esta categoría.

efectividad de la corrección de errores. La primera estrategia consiste en transmitir bits adyacentes en subcarriers separados entre sí, mientras que la segunda estrategia transmite bits adyacentes en momentos distintos.

Lo que se logra al aplicar estas estrategias es que la probabilidad de que en el receptor se reciban muchos bits corruptos consecutivos sea menor, lográndose un incremento en la efectividad de los algoritmos de corrección de errores.

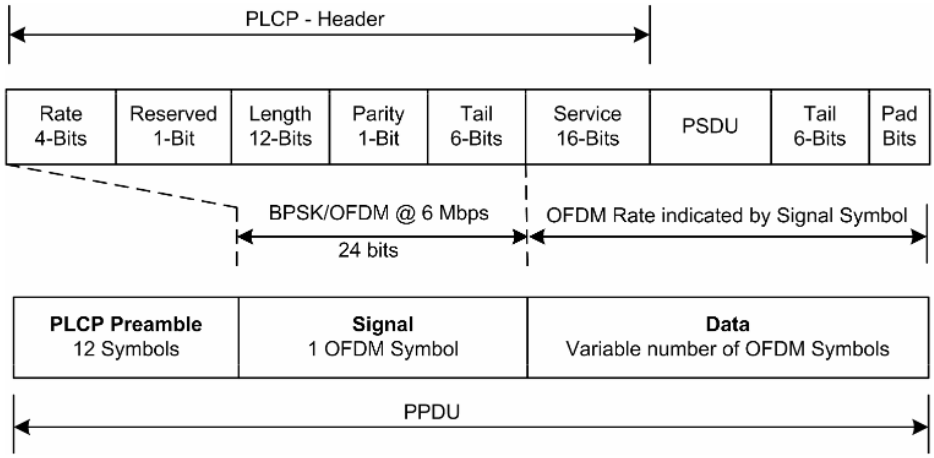

Figura 3.4: Representación gráfica del formato de un PPDU de capa física OFDM IEEE 802.11. Extraído de [16].

Preámbulo

El preámbulo hace posible que el receptor de la señal detecte la trama y pueda sincronizar su demodulador. Está constituido por 12 símbolos comúnmente llamados *símbolos de entrenamiento*.

De estos 12 símbolos, 10 de ellos son "cortos", y por esta razón es que al conjunto de los mismos se le denomina secuencia de entrenamiento corta (*Short Training Sequence*). La utilidad de esta secuencia es que el receptor pueda realizar las tareas de control automático de ganancia (*AGC, Automatic Gain Control*), selección de diversidad (*Diversity Selection*) y estimación de frecuencia (*Coarse Frequency Estimation*) sobre la señal carrier.

Los dos símbolos restantes son "largos" y a su conjunto se le conoce como secuencia de entrenamiento larga (*Long Training Sequence*). Esta secuencia se utiliza para realizar la estimación del canal (*Channel Estimation*) y la estimación de frecuencia fina (*Fine Frequency Offset Estimation*). Este preámbulo toma hasta 16 microsegundos en ser procesado a partir del momento en que se detecta la señal en el medio de comunicación.

Campo de señal (*SIGNAL*)

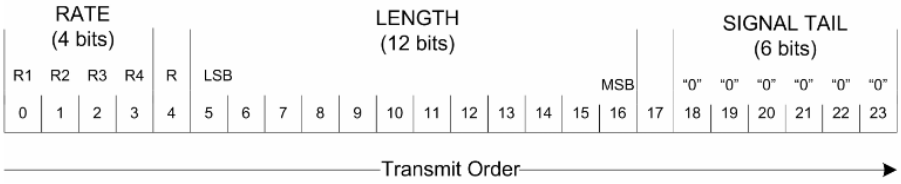

Figura 3.5: Formato de un cabezal PPDU de capa física OFDM IEEE 802.11. Extraído de [16].

En la Figura 3.5 se detalla la composición del campo *SIGNAL*. El campo de tasa (*RATE*) indica el esquema de modulación y la tasa de codificación del código convolucional utilizado para el resto del PPDU (a partir del final del campo SIGNAL). El bit de reserva siempre tiene el valor 0 ya que no se utiliza. Luego le sigue el campo de longitud (*LENGTH*) que indica la cantidad de bytes que tiene el PSDU encapsulado. El bit 17 es un bit de paridad calculado sobre los primeros 17 bits del PPDU, mientras que el campo de *SIGNAL TAIL* es relleno (*padding*) con valor 0 que se agrega al final del PPDU.

Campo de datos (*DATA*)

Como se puede ver en la Figura 3.4, el campo de datos es el último del PPDU, y el único en tener largo variable. Además, el mismo está compuesto por el campo de servicio (*SERVICE*), el PSDU, y los campos de *TAIL* y *PAD*.

El campo de servicio tiene un tamaño de 16 bits, donde los primeros 7 tienen el valor de 0 y son utilizados para sincronizar el *descrambler* en el receptor. Los otros 9 bits son reservados y tienen el valor 0 por defecto.

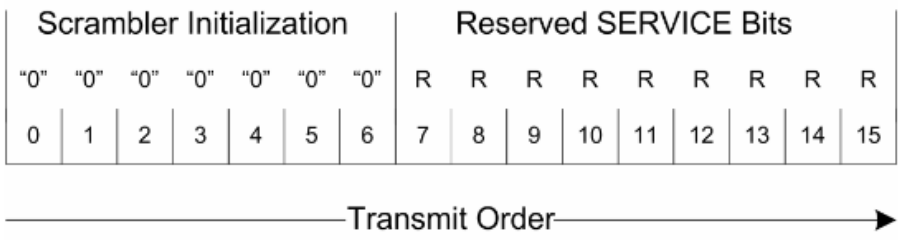

Figura 3.6: Composición del campo de servicio. Extraído de [16].

El PSDU es la trama recibida de la capa de enlace, y su largo máximo es de 4095 bytes. El campo *TAIL* contiene 6 bits con valor 0 y se utiliza para que el decodificador convolucional en el receptor pueda estimar los últimos bits del PSDU de forma más confiable. A estos 6 bits se le aplica *scrambling* antes ser transmitidos. Finalmente, el campo de *padding* se asegura de que el largo total del campo de datos sea un múltiplo de la cantidad de bits codificados por símbolo OFDM, lo cual claramente depende del esquema de modulación utilizado.

Una vez que se tiene el campo de datos, se le aplica un scrambling a todos los bits del mismo con el objetivo de evitar ráfagas largas de bits con el mismo valor de forma consecutiva. La existencia de este tipo de ráfagas tiene como consecuencia que la decodificación en el receptor sea más propensa a cometer errores de bit.

El proceso de scrambling asegura que haya un balance razonable entre la cantidad de ceros y unos consecutivos que se envían para transmitir.

3.2. Radios definidas por software

El área de radios definidas por software (Software Defined Radio, SDR) ha sido de gran interés desde hace ya varias décadas tanto para el sector de las telecomunicaciones como para el de las ciencias de la computación. Como ya se mencionó anteriormente, el hecho de que el procesamiento de la señal se haga a nivel de software presenta diversas ventajas con respecto al enfoque tradicional, en el cual es necesario disponer de hardware especializado para las distintas tareas que componen al procesamiento de la señal.

Algunas de estas ventajas son:

- Permite diseñar la lógica y los distintos algoritmos que competen al procesamiento de la señal abstrayéndose total o parcialmente del hardware donde eventualmente

se ejecutará el programa. Esto significa que una misma SDR puede ejecutarse en distintas arquitecturas. También existe el caso en que no se tienen distintas arquitecturas sino una sola con un diseño lo suficientemente genérico, (por ejemplo, con un GPP), que soporta la ejecución de un gran número de SDRs distintas. Este último aspecto tiene claras aplicaciones comerciales, donde la producción masiva de un mismo producto en general abarata costos de producción y de esta forma permite obtener una mayor ganancia al fabricante.

- Acelera de forma significativa los tiempos y costos de prototipación y evaluación de nuevos paradigmas o algoritmos para resolver una determinada tarea. Si se quisiera hacer dicha prototipación en una radio convencional, no habría otra alternativa que construir un circuito especializado en realizar el nuevo algoritmo o tarea que se quiere evaluar. En una SDR, basta con reescribir ese módulo del programa, y actualizar la versión del mismo en donde ejecute la SDR. En un artículo muy reciente publicado en la *IEEE Communications Magazine*[14] se habla de cómo el uso de SDRs logra que el tiempo de prototipación de sistemas con relevancia comercial se reduzca de meses a horas.

  Esta característica ha permitido que las SDRs, durante los últimos años, se hayan convertido gradualmente en un recurso imprescindible para la investigación, desarrollo, y enseñanza en el área de las telecomunicaciones.

- Facilita notablemente el mantenimiento y actualización de las radios. Por ejemplo, es posible distribuir actualizaciones de forma remota para dispositivos ya liberados, lo cual es un caso de uso común en el ámbito comercial. Las actualizaciones podrían ser algún cambio menor, como por ejemplo mejoras en los algoritmos utilizados, o algo tan importante como agregar soporte a un nuevo protocolo de comunicación.

Teniendo todas estas características a favor, es posible preguntarse por qué es que las SDRs no han reemplazado totalmente al enfoque tradicional en la construcción de radios. El problema es que las SDRs también tienen algunas desventajas si se comparan con las radios tradicionales, a pesar de que en los últimos años se ha logrado un avance considerable en los puntos mencionados a continuación:

- La programación del procesamiento de la señal en software añade tiempo de procesamiento, por lo que es más lenta que su contraparte implementada en un hardware dedicado. Esto implica que el hardware en donde ejecuta la SDR debe ser siempre más potente que el hardware que implementa esa radio con circuitos dedicados para lograr un rendimiento equivalente, como puede ser por ejemplo recibir a una tasa de transferencia determinada. Este es el mismo problema que se observa en la emulación de distintas arquitecturas.

[14] Para más información, ver [10].

El impacto de este problema ha ido disminuyendo con el tiempo debido al constante incremento en la capacidad de cómputo de los procesadores de propósito general que ha existido en las últimas décadas.

- El elevado costo de las plataformas de hardware con suficiente poder de cómputo para ejecutar la SDR y con posibilidades de operar en un rango amplio de frecuencias.

- La escasez de entornos para desarrollar el software que ejecuta en una SDR. En los comienzos de las SDR, no existía un entorno apropiado para desarrollar este tipo de software de forma sencilla, y a lo largo de los años ha sido una limitante importante en el crecimiento del área.

  Hoy en día se ha avanzado considerablemente en este punto, ya que existen varias alternativas como SDR Sharp, MATLAB/Simulink, y GNU Radio. De la investigación realizada, se puede concluir que GNU Radio es el entorno más potente para desarrollar software de SDRs, y de forma más general, aplicaciones DSP[15]. GNU Radio es mencionado en varios artículos como el software de cabecera para desarrollar SDRs, y existen numerosos trabajos de investigación serios hechos con este programa.

En el presente trabajo se decidió trabajar con GNU Radio debido a que, además de ser un software de alta calidad y muy bien considerado en el ámbito académico, es el entorno de desarrollo en el cual se creó la versión original del transceptor OFDM IEEE 802.11 utilizado como punto de partida, como se verá con más detalle en la Sección 4.2.

3.3. GNU Radio

GNU Radio[16] es un *framework* gratuito y de código libre (*open source*) que provee bloques de procesamiento de señales para implementar radios definidas por software y DSPs genericos. GNU Radio puede ser utilizado en conjunto con hardware de radiofrecuencia con el fin de transmitir y recibir señales en el medio, o también puede ser utilizado solamente en software, en un ambiente de simulación pura. Creado en el año 2001, GNU Radio es utilizado extensivamente en la actualidad en ámbitos académicos, comerciales y en el de la radioafición.

[15] DSP significa Digital Signal Processor, Procesador Digital de Señales, que es un sistema basado en un procesador que dispone de software optimizado para aplicaciones que requieren operaciones numéricas a muy alta velocidad.

[16] Para más información, ver [12].

3.3.1. Conceptos básicos

Bloques

GNU Radio define el concepto de *bloque* como una unidad funcional capaz de realizar una tarea determinada, y que posee puertos de entrada y salida para poder recibir y comunicar datos con otros bloques, respectivamente. Por trazar una analogía, un bloque de GNU Radio se asemeja a lo que es una clase en un lenguaje orientado a objetos.

Los bloques se pueden clasificar en dos categorías:

- **Atómicos**: Este tipo de bloques se caracteriza por implementar una determinada funcionalidad básica, que debe ser programada en C++ o Python.
- **Jerárquicos (Hierarchical)**: Los bloques jerárquicos se definen mediante la interconexión de otros bloques, que a su vez pueden ser jerárquicos o atómicos.

Esta característica significa que la arquitectura de GNU Radio es altamente escalable y modularizable.

Módulos

Los bloques a su vez se agrupan en módulos. Un módulo es una colección de bloques que son agrupados por una decisión de diseño. Por ejemplo, existe un módulo llamado *Filters* que agrupa filtros de todo tipo. Otros ejemplos son los módulos *Equalizers, Channelizers*, entre otros. Estos módulos pertenecen al árbol de fuentes de GNU Radio[17].

GNU Radio brinda la posibilidad de crear bloques y módulos propios utilizando la funcionalidad de *Out Of Tree Modules*. La implementación original, de la cual es punto de partida el presente trabajo, define un Out of Tree Module llamado IEEE 802.11 que agrupa todos los bloques del transceptor relacionados específicamente con la implementación del estándar Wi-Fi. Para crear bloques y módulos nuevos se utiliza la herramienta *gr-modtool*, lo cual es explicado con más detalle en el Anexo B debido a que como se verá más adelante, fue necesario crear un nuevo bloque para aplicar mejoras al rendimiento.

Grafos de flujo (*Flow Graphs*)

Todos los programas creados para ejecutar en GNU Radio son grafos de flujo (*Flow Graphs*) donde los nodos del grafo son bloques y las aristas representan comunicación

[17] El *GNU Radio Source Tree* refiere a los distintos módulos y bloques que pertenecen a la distribución oficial de GNU Radio. Para más información, ver [14].

entre bloques, es decir, los datos fluyen de un bloque a otro si están conectados por una arista. Asimismo, las aristas son dirigidas, lo que determina la dirección en la cual fluyen los datos. Un grafo de flujo puede ser ejecutado por GNU Radio para simular un sistema de procesamiento de señales, o puede ser utilizado para definir bloques nuevos, generando así los bloques de tipo jerárquicos.

3.3.2. Vector-Optimized Library of Kernels (VOLK)

El concepto de VOLK se introdujo por primera vez como parte del framework de GNU Radio en diciembre del 2010[18]. VOLK es una biblioteca que contiene múltiples funciones que realizan cálculos de operaciones matemáticas y que han sido programadas utilizando un enfoque *Single Instruction Multiple Data (SIMD)* por los desarrolladores y colaboradores de GNU Radio, poniendo especial foco en la eficiencia de la función. A estas funciones se las denomina *kernels*.

A su vez, para cada uno de estos kernels se tienen varias implementaciones que han sido programadas para distintas arquitecturas y distintos conjuntos de instrucciones. A estas implementaciones se les denomina *proto-kernels*. Generalmente, dentro de las implementaciones disponibles para una función determinada existe una lo suficientemente genérica como para ejecutar en cualquier máquina (por ejemplo, una programada en C).

El objetivo final de VOLK es brindar implementaciones que sean lo más eficientes y rápidas posibles para la arquitectura donde se esté ejecutando el programa, de una forma sencilla. En general, los kernels VOLK que existen realizan el cálculo de operaciones matemáticas que son recurrentes en el campo del procesamiento de señales.

Al crear un grafo de flujo que utilice algún bloque que tenga soporte VOLK, solo hace falta ejecutar el comando *volk_profile* para que el sistema determine cual de los proto-kernels disponibles es el más eficiente dada la arquitectura donde se está ejecutando. Para esto, se ejecutan una serie de pruebas que evalúan el rendimiento de cada proto-kernel en tiempo real. Una vez realizada esta etapa, GNU Radio salva en un archivo de configuración qué proto-kernel es el más eficiente para un kernel dado, en esa arquitectura específica.

3.3.3. Tipos de Datos Polimórficos (PMT)

Los *Polymorphic Data Types (PMTs)* son un tipo de dato opaco propio de GNU Radio diseñado para oficiar de contenedores génericos sobre datos con un tipo específico (por ejemplo, strings, longs, complejos, etc.), de forma que estos contenedores

[18] Para más información, ver [13].

puedan ser pasados de forma sencilla entre distintos bloques y threads de GNU Radio, y expongan una interfaz en común. Son utilizados extensivamente en la interfaz para pasaje de mensajes que provee GNU Radio, así como para definir etiquetas de flujo (*Stream Tags*), que también son parte fundamental de los bloques de tipo *Tagged Stream*, conceptos que se presentan en la siguiente sección.

3.3.4. Modelos de datos de GNU Radio

GNU Radio fue concebido originalmente como un sistema que manejaba un flujo de datos en el cual los mismos fluían de bloque en bloque en una única dirección, sin ningún otro mecanismo para la comunicación entre bloques. Este modelo es muy bueno cuando se trabaja con muestras, señales analógicas y bits entre otros, pero genera complicaciones al momento de trabajar con datos de control, y datos empaquetados. El concepto de un paquete, con inicio y fin definido, no podía ser modelado con el esquema original de GNU Radio. Para solucionar estos problemas, se introdujeron dos nuevos modelos de comunicación entre bloques: *Stream Tags* y *Message Passing*.

Pasaje de mensajes (*Message Passing*)

El pasaje de mensajes es un modelo de comunicación de datos asíncrono en el cual un bloque puede enviar mensajes a otro independientemente de la jerarquía de bloques definida por el flujo de datos principal. También es ampliamente utilizado para comunicar bloques GNU Radio con componentes externos permitiendo transmitir datos en cualquiera de las dos direcciones. Utiliza las primitivas de los PMT para encapsular los mensajes a ser transmitidos.

Cada bloque de GNU Radio posee colas de mensajes para mensajes recibidos, así como la posibilidad de publicar mensajes en las colas de otros bloques.

El mecanismo de pasaje de mensajes aplica el patrón de diseño de software *Observer*, debido a que para que un bloque reciba un mensaje publicado por otro bloque, el primero debe suscribirse como interesado a los puertos de publicación del segundo. De esta forma, cuando un bloque desea transmitir un mensaje, lo que hace es enviar el mismo a todos los bloques que se encuentren suscriptos al puerto donde se ha publicado el mensaje.

Etiquetado de flujos (*Stream Tags*)

Este mecanismo de comunicación se distingue del de pasaje de mensajes en que se utiliza un flujo de datos que es sincrónico con el flujo de datos principal. La API de pasaje de mensajes no da ninguna garantía de cuándo es que un bloque recibirá un mensaje con respecto a los datos que fluyen por el flujo principal. En cambio, cuando

se le añade una etiqueta a un ítem del flujo de datos principal (de ahí el término *tag*), GNU Radio se asegura que dicha etiqueta acompañará al ítem etiquetado a medida que este se desplaza dentro del flujo de datos.

La etiqueta es un objeto de tipo PMT, que tiene una clave que indica qué semántica tiene el valor almacenado en la etiqueta, y un valor, que es un objeto de tipo PMT, es decir, permite almacenar cualquier tipo de información. Por ejemplo, se podría utilizar un Stream Tag para indicar el inicio de un paquete, y que la clave fuera *length* y el valor el largo en bytes del paquete. De esta forma, bloques del grafo de flujo podrían realizar determinadas acciones sobre este paquete, como extraer el cabezal en caso que hubiese, entre otros.

3.3.5. Suite de herramientas

Por otra parte, GNU Radio cuenta con una suite de herramientas que facilita la tarea de crear radios definidas por software. Para el presente proyecto se utilizaron cada uno de los siguiente programas:

- **GNU Radio**: Programa central del framework, se encarga de ejecutar todos los grafos de flujo que puedan ser interpretados por la herramienta.
- **GNU Radio Companion**: IDE que permite la creación intuitiva de grafos de flujo, brindando la posibilidad de visualizar de manera gráfica todos los bloques y conexiones que tiene el grafo.
- **GNU Control Port Monitor**: Permite obtener metadatos de los bloques asociados a una instancia de ejecución del grafo de flujo. Ejemplos son la cantidad de ítems enviados y recibidos, y el tiempo consumido en CPU, entre otros.
- **GNU Radio Performance Monitor**: Esta herramienta es similar a la anterior, aunque se centra exclusivamente en el rendimiento del grafo de flujo que se está ejecutando. Brinda información detallada del porcentaje de trabajo y el tiempo que cada bloque consume con respecto a los otros bloques presentes en el grafo, así como distintas métricas interesantes como el promedio, la varianza, o el tiempo consumido en la última ejecución de cada bloque. También permite visualizar de forma gráfica el volumen de datos que fluye por las aristas del grafo (conexiones entre bloques).

3.4. Hardware para transmisión y recepción de señales en una SDR

En esta sección se explica de forma muy breve las principales opciones que se tienen hoy en día en cuanto al hardware responsable de realizar la comunicación con el medio

en una SDR.

Actualmente existen varios dispositivos en el mercado con el fin de permitir a una SDR transmitir y recibir información. En un artículo publicado en Enero de 2016 en la IEEE Communications Magazine[19], se mencionan las empresas y productos más influyentes al día de hoy en este tema. Algunas de estas son Ettus Research, ZedBoard y NooElec.

En los artículos publicados por el autor del transceptor IEEE 802.11 OFDM que se estudia en el presente trabajo, se menciona que el hardware para realizar la comunicación con el medio es de la compañía Ettus Research, dentro de la línea de los *Universal Software Radio Peripheral (USRP)*. Más concretamente, las pruebas realizadas por el autor de la implementación original fueron hechas con un USRP N210. En el caso de este trabajo, como se verá más adelante, se realizaron pruebas con un USRP B210, dispositivo que funciona de manera similar.

Por esta razón, se describen brevemente las antenas del tipo USRP debido a su relevancia en este trabajo.

3.4.1. Universal Software Radio Peripheral (USRP)

Las USRP son una línea de plataformas de hardware desarrolladas con el fin de implementar algunas de las tareas que realiza una radio definida por software, producidas por la compañía Ettus Research.

Particularmente, los USRPs cuentan con antenas, y se encargan de transmitir y recibir señales en el aire. Estos dispositivos se conectan a las computadoras convencionales a través de una conexión de red Ethernet, o de tipo USB. Existen algunos tipos de USRP que poseen FPGAs[20] integradas en sus circuitos, que permiten realizar alguna tarea de procesamiento de señal y no solo transmitir y recibir la señal en el medio.

A pesar de que tanto GNU Radio como los USRPs son concebidos como dos productos independientes que pueden interoperar con otros productos alternativos que presenten la misma funcionalidad, existe una extensa documentación y soporte que se centra en cómo construir SDRs utilizando USRPs en GNU Radio.

Una de las razones de esto es que el fundador de Ettus Research, Matt Ettus, es también uno de los managers y principales encargados del proyecto GNU Radio. Como consecuencia, el uso de USRPs es uno de los más populares, sino el más popular, para transmitir y recibir señales en una SDR desarrollada en GNU Radio, y tanto en la implementación original del transceptor como en el presente proyecto se mantiene esta tendencia.

[19] Para más información, ver [9].

[20] Field Programmable Gate Array (FPGA) es un tipo de circuito reconfigurable que permite realizar distintos tipos de operaciones con una eficiencia similar a la de un circuito dedicado.

Capítulo 4

Mejoras introducidas

4.1. Introducción

En este capítulo se presentan con un alto nivel de detalle todas las decisiones que fueron tomadas a nivel de análisis, diseño e implementación para la realización de un prototipo final que estuviera alineado con los objetivos del presente proyecto.

En primer lugar, se hará un análisis de la implementación del transmisor/receptor IEEE 802.11 OFDM que fue tomada como punto de partida para este trabajo, incluyendo los motivos de su elección y su funcionamiento detallado. En segundo lugar, se detallará el procedimiento utilizado para localizar los bloques y funciones de dicha implementación con mayor carga de trabajo, realizando un *profiling*[1] del sistema en distintos niveles. Por último, se expondrán en detalle las mejoras realizadas a la implementación original.

4.2. Implementación original

Debido a la gran magnitud del trabajo a realizar, se tomó la decisión de reutilizar una implementación previa y funcional de un transmisor y receptor IEEE 802.11 OFDM. Dicha implementación forma parte de un trabajo reciente que puede encontrarse en [5]. El código del transmisor y receptor se encuentra disponible bajo la licencia GPL v3.

Al proyecto que enmarca esta implementación del estándar Wi-Fi en GNU Radio se lo denomina con el nombre de *gr-ieee802-11*. A lo largo de esta sección se utiliza este nombre para referirse a la implementación original. Esta implementación es un

[1] El *profiling* tiene como objetivo realizar un análisis de un software en ejecución y obtener medidas sobre distintos aspectos del mismo. Para más información, ver la Sección 4.3.

prototipo funcional completo de un receptor OFDM implementado en GNU Radio y adaptable para operar junto con un Ettus USRP.

Dicho prototipo es el primero de su clase para esta tecnología, implementando todas las funcionalidades hasta el nivel de capa MAC y teniendo la capacidad de extraer la información de un cabezal MAC y de devolver los datos enviados como carga útil en una red IEEE 802.11 a/g/p. Soporta tanto Wi-Fi con un ancho de banda de 20 MHz como IEEE 802.11p DSRC con un ancho de banda de 10 MHz.

La implementación fue validada y verificada en términos de interoperabilidad con distintos sistemas. Además, según los propios autores, el hecho de que el código fuente se encuentre disponible como Open Source provee un mecanismo sencillo para que otros desarrolladores puedan experimentar con algoritmos de procesamiento de señales nuevos.

Debido a que el objetivo principal de dicho trabajo fue ofrecer una implementación funcional en GNU Radio, los aspectos de rendimiento no fueron tenidos en cuenta en gran medida por los desarrolladores. En particular, existen ciertas limitaciones del prototipo que reducen su funcionalidad:

- No están implementados los mecanismos de señalización adecuados, por lo que no es posible asociarse a otro dispositivo.
- El receptor no puede generar tramas ACK en el tiempo exigido por el estándar.
- No está implementado el mecanismo de acceso al medio CSMA/CA, definido en el modo DFC.

Además, según se describe en el artículo, si bien el receptor es capaz de realizar la detección de tramas en el tiempo exigido por el estándar, no puede decodificarlas en el tiempo necesario. Esto limita el comportamiento del receptor y del sistema en su conjunto.

Las siguientes secciones detallan el funcionamiento tanto del receptor como del transmisor.

4.2.1. Implementación del receptor

En la Figura 4.1 se presenta la representación gráfica del grafo de flujo del receptor tal como se visualiza en GNU Radio Companion. El receptor se puede dividir lógicamente en dos funciones bien definidas: la detección de una trama y la decodificación de la misma. El conjunto de bloques responsable de realizar la detección de la trama consiste en los primeros bloques presentes en el grafo de flujo (hasta el bloque OFDM Sync Short inclusive). Los bloques responsables de realizar la decodificación de las tramas son los bloques restantes (a excepción de los bloques de tipo Pad Sink que simplemente son puertos de salida).

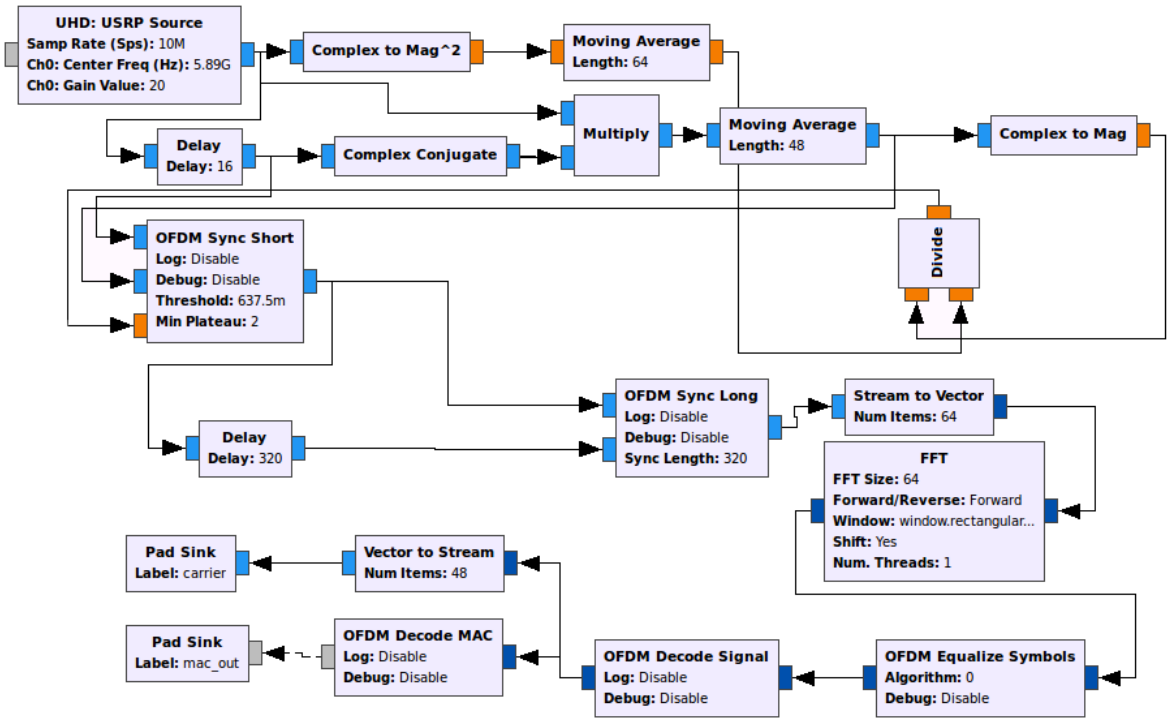

Figura 4.1: Grafo de flujo del receptor visualizado en GNU Radio Companion.

Nótese que en el comienzo del grafo de flujo hay un bloque llamado *USRP Source*. Este bloque oficia de interfaz entre una antena USRP conectada al equipo y el grafo de flujo definido en GNU Radio. Como el bloque que recibe la señal y la introduce en el grafo de flujo está desacoplado del procesamiento de la señal propiamente dicho, es muy simple reemplazar la fuente de la señal por otro bloque que genere una señal de distinta forma (por ejemplo, leyendo de un archivo del sistema de archivos). En estos casos, el receptor funciona de la misma forma.

En etapas tempranas del presente trabajo, se utilizó el enfoque de generar la señal a partir de un archivo, de forma de poder prescindir del USRP para evaluar características generales de la operación de recepción. Esto se explica con más detalle en el Capítulo 5.

Detección de la trama

La primera tarea que debe realizar el receptor es detectar el inicio de la trama. Como se explicó en la Sección 3.1, cada trama IEEE 802.11 a/g/p comienza con un preámbulo que consiste de un patrón que ocupa 16 muestras y se repite 10 veces. La implementación del receptor utiliza un algoritmo de detección de trama que se detalla en [19] y que basa su funcionamiento en explotar la naturaleza repetitiva de la Short Training Sequence para detectar el inicio de la trama. Para esto, se hace uso del concepto de *autocorrelación*.

La autocorrelación se define como:

$$a[n] = \sum_{k=0}^{N_{win}-1} s[n+k]\overline{s}[n+k+16], \tag{4.1}$$

en donde N_{win} es el tamaño de la ventana sobre la cual se calcula la autocorrelación. La implementación original utiliza 48 como valor de N_{win}.

Para mantener independencia del nivel absoluto de muestras recibidas, se normaliza la autocorrelación por el promedio de la potencia, dando lugar a una autocorrelación normalizada. Se define el promedio de la potencia como:

$$p[n] = \sum_{k=0}^{N_{win}-1} s[n+k]\overline{s}[n+k], \tag{4.2}$$

y a la autocorrelación normalizada como:

$$c[n] = \frac{|a[n]|}{p[n]}, \tag{4.3}$$

en donde $|a[n]|$ es la magnitud de $a[n]$.

La autocorrelación de la señal es particularmente alta durante la *Short Training Sequence*, que a su vez se corresponde con el inicio de la trama. De esta forma, la estrategia está en calcular la autocorrelación de la señal y compararla con un umbral para determinar si se está recibiendo una trama o no. Para esto, se define el concepto de *Plateau*, que significa que tres muestras consecutivas se encontraron por encima del umbral elegido. De esta forma, detectar el inicio de la trama equivale a detectar los *Plateaus* que se reciben en la señal.

El bloque *Delay* recibe la muestra actual de la señal y la retrasa de forma tal que quede alineada con la dieciseisava muestra posterior. A esta muestra retrasada se le calcula la conjugada compleja y se multiplica el resultado por el valor inicial de la muestra (el valor antes de retrasarla). Esto completa el cálculo para cada paso de la sumatoria. El resultado final (es decir, la suma de cada paso) lo hace el bloque *Moving Average*, que permite parametrizar el tope de la sumatoria fácilmente (valor correspondiente al tamaño de la ventana N_{win}).

Por su parte, el bloque *Complex to Mag* calcula la magnitud de la autocorrelación desnormalizada $a[n]$, valor que se divide para calcular la autocorrelación normalizada de la señal. Los bloques que realizan el cálculo de $c[n]$ se muestran en la Figura 4.2.

Luego de esto, el bloque *OFDM Sync Short* se encarga de detectar los *Plateaus* en la señal de autocorrelación, y en base a eso determina si envía las muestras a los bloques sucesivos en el grafo de flujo de recepción o si las descarta.

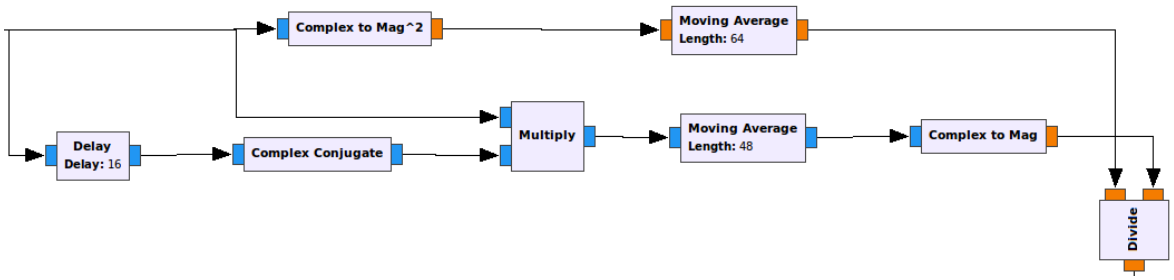

Figura 4.2: Conjunto de bloques encargados de calcular la autocorrelación de la Short Training Sequence.

Decodificación de la trama

Como se explicó en la Sección 3.1, la primera tarea a realizar una vez que se detecta la trama se conoce como Frequency Offset Correction. Esta corrección es necesaria ya que las frecuencias de los osciladores del transmisor y el receptor pueden estar levemente desfasadas. Para compensar (y en el mejor caso, corregir), la implementación utiliza un algoritmo que fue propuesto por primera vez en [20] y que hace uso de la Short Training Sequence para determinar el desfase entre el transmisor y el receptor.

La idea básica del algoritmo de corrección es la siguiente. En un caso ideal, luego de la detección de la Short Training Sequence, las muestras $s[n]$ y la $s[n+16]$ son idénticas, ya que cada 16 muestras se repite el valor enviado. Sin embargo, debido al ruido del canal y el desfase en la frecuencia de los osciladores, las muestras no son exactamente iguales, y como consecuencia, $s[n]\overline{s}[n+16]$ no siempre da como resultado un número real. Ignorando el ruido, el argumento del producto anterior corresponde a la rotación introducida por el desfase de frecuencias entre muestras multiplicada por 16, ya que se acumuló un desfase correspondiente a 16 muestras enviadas. De esta forma, para estimar el valor final se utiliza la siguiente fórmula:

$$\mathrm{d}f = \frac{1}{16}\arg\Big(\sum_{n=0}^{N_{short}-1-16} s[n]\overline{s}[n+k]\Big), \tag{4.4}$$

en donde N_{short} es el largo de la Short Training Sequence. Una vez calculado $\mathrm{d}f$, se aplica la corrección a cada muestra de la siguiente forma:

$$s[n] \leftarrow s[n]e^{\mathrm{i}\ (\mathrm{n}\ \mathrm{d}f)} \tag{4.5}$$

Este paso es implementado por el bloque *OFDM Sync Long*. Este bloque también es responsable de realizar la tarea de alinear símbolos (*Symbol Alignment*). El objetivo de la alineación de símbolos es calcular exactamente dónde empieza un símbolo, y extraer los datos para luego enviarlos como entrada a un bloque que computa la Transformada Rápida de Fourier (FFT). Esta alineación se hace con la ayuda de la Long Training

Sequence, que está compuesta por un patrón de 64 muestras que se repite dos veces. Esta etapa utiliza una técnica llamada *matched filtering*, ya que a pesar de que es costoso, la alineación debe ser extremadamente precisa.

El resultado de este bloque está listo para ser enviado al bloque que computa la FFT, con la salvedad de que el bloque *FFT* tiene como entrada un vector de complejos y el OFDM Sync Long tiene como salida un flujo (stream) de complejos. Para esto se utiliza un bloque intermedio a modo de adaptador que realiza la conversión de flujo a vector, llamado *Stream to Vector*. Una vez que se tiene el vector de complejos, se pasa la información al bloque *FFT*, quien convierte la señal, que se encuentra en el dominio del tiempo, al dominio de la frecuencia.

El próximo paso en el grafo de flujo de la decodificación de la trama es corregir el desfase en la fase. Debido a que los tiempos de muestreo en el transmisor y receptor no están sincronizados y a que la alineación de símbolos no es perfecta, se introduce un desfase en la fase. Este desfase es lineal con la frecuencia, y es corregido utilizando los subcarriers piloto.

La siguiente etapa es la de estimación de canal. Además de la fase, la magnitud de los carriers también debe ser corregida. Esto es especialmente importante cuando se utiliza un esquema de modulación 16QAM o 64QAM, donde la amplitud de la señal se usa para codificar información. Las tareas de estimación de canal y de corrección de desfase en la fase son realizadas por un mismo bloque, el *OFDM Equalize Symbols*. Este bloque también se encarga de eliminar los subcarriers de guarda y pilotos, por lo que consume 64 símbolos y produce 48 por ejecución.

El próximo bloque en el grafo de flujo es el *OFDM Decode Signal*. En cada trama, tanto la Short Training Sequence como la Long Training Sequence son sucedidas por un campo denominado *Signal Field*. Este campo contiene información sobre la longitud y la codificación de los símbolos. La codificación de este campo es BPSK 1/2. Para decodificar el código convolucional, se recurre a la biblioteca IT++.

Si el campo es decodificado sin problemas, es decir, si el campo SIGNAL contiene un valor válido para la tasa, y además el bit de paridad es correcto, entonces el bloque OFDM Decode Signal se encarga de marcar con una etiqueta a la muestra. Dicha etiqueta contiene una dupla con la codificación y la longitud de la trama. Esta etiqueta es requerida por el bloque posterior en el grafo, y es la forma que tiene GNU Radio para pasar metadatos asociados a una muestra.

El último paso consiste en decodificar los datos propiamente dichos, es decir, la carga útil. El bloque encargado de realizar este trabajo es el *OFDM Decode MAC*. Este último paso, que es especialmente importante, se divide en tareas bien definidas:

- **Demodulación**: Se recibe un vector de 48 puntos de constelación en el plano complejo, uno por cada subcarrier de datos que tiene un símbolo OFDM. Dependiendo del esquema de modulación utilizado, estos puntos son convertidos a números en punto flotante, que representan los bits de la modulación utilizada.

- **Deinterleaving**: Este paso depende de la modulación y esquema de codificación elegidos. Lo que se hace es permutar los bits de un símbolo. La permutación es la misma para todos los símbolos de una trama.
- **Decodificación convolucional**: Aquí nuevamente se recurre a la biblioteca IT++ para hacer la decodificación del código convolucional. Este paso está descrito con detalle en la Sección 2.2.
- **Descrambling**: Este es la última etapa en el proceso de decodificación. El estado inicial del scrambler está configurado en el transmisor a un valor pseudo-aleatorio. Los primeros siete bits de la carga útil son parte del *Service Field* y siempre son configurados en cero, de forma que el receptor pueda deducir el estado inicial del scrambler en el transmisor. El estado inicial se obtiene de estos siete bits a partir de una búsqueda en una tabla en el receptor.

Finalmente, la salida del proceso de decodificación es un mensaje de GNU Radio que representa el PDU del mensaje.

4.2.2. Implementación del transmisor

La implementación del transmisor resulta ser mucho más sencilla que la del receptor. En este caso, es suficiente seguir los pasos que se describen en el estándar, e ir agregando los elementos que conforman la trama. A diferencia del receptor, no es necesario ningún mecanismo complejo como el que se utiliza para detectar o realizar la sincronización de una trama. La implementación del transmisor se puede ver en la Figura 4.3.

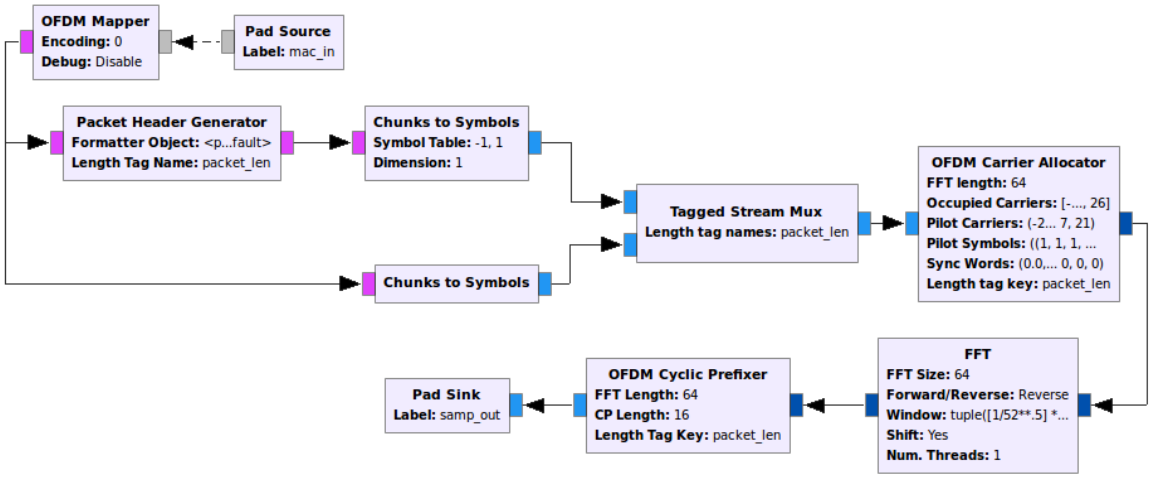

Figura 4.3: Implementación del transmisor en GNU Radio Companion.

El primer bloque del transmisor, llamado *OFDM Mapper*, realiza los siguientes pasos, que se detallan con precisión en el estándar Wi-Fi:

1. Calcular, del campo *RATE* del *TXVECTOR*, el número de bits de datos por cada símbolo OFDM, la tasa de codificación, el número de bits que tendrá cada subcarrier de OFDM y el número de bits codificados por cada símbolo OFDM.

2. Añadir el PSDU al campo *SERVICE* del *TXVECTOR* y extender la cadena de bits resultante con ceros (por lo menos seis bits) de forma tal que la longitud final sea un múltiplo del número de bits de datos por símbolo OFDM. La cadena de bits final constituirá la sección *DATA* del paquete.

3. Iniciar el scrambler con una semilla pseudo-aleatoria distinta de cero, generar una secuencia de scrambling y realizar el *XOR* de la misma con la cadena extendida del punto anterior.

4. Reemplazar los seis bits de ceros que fueron mezclados en el paso anterior con seis bits de ceros sin mezclar. Dichos bits devolverán al codificador convolucional al estado inicial y son denominados *bits de cola* (*tail bits*).

5. Realizar la codificación de los datos que fueron extendidos y mezclados utilizando un codificador convolucional. Omitir parte de la cadena de salida, elegida de acuerdo a un patrón de puncturing[2], para alcanzar la tasa de codificación deseada.

6. Dividir la cadena de bits codificada en grupos de bits según el número de bits codificados por cada símbolo OFDM. Dentro de cada grupo, realizar el interleaving de los bits según alguna regla que dependa del RATE deseado.

7. Dividir la cadena de datos codificada y reordenada en grupos de bits según el número de bits de cada subcarrier en OFDM. Para cada grupo de bits, convertirlo en un número complejo según las tablas de codificación para la modulación elegida.

A continuación, la salida se redirige al bloque *Packet Header Generator*, que es el encargado de generar el cabezal de la trama. Éste se compone de dos partes:

- Un campo de preámbulo PLCP, compuesto de diez repeticiones de la Short Training Sequence y dos de la Long Training Sequence, precedidas por un intervalo de guarda.

- Un campo de cabezal PLCP, que se genera rellenando los campos de bits apropiados a partir de los campos RATE, LENGTH y SERVICE del TXVECTOR. Los campos RATE y LENGTH del cabezal PLCP se codifican mediante un código convolucional a una tasa de R = 1/2, y son subsecuentemente mapeados a un único símbolo OFDM codificado mediante BPSK, y denotado *SIGNAL*. Para

[2] El puncturing se encuentra explicado en la Sección 2.2.4.

facilitar una detección confiable y precisa de los campos RATE y LENGTH, seis bits de ceros se insertan al final del cabezal PLCP. La codificación del campo SIGNAL en un símbolo OFDM sigue los mismos pasos para la codificación convolucional, interleaving, modulación BPSK, inserción de piloto y transformación de Fourier que se utilizan para la transmisión de datos con la versión BPSK de OFDM a una tasa de codificación de 1/2, salvando el hecho de que no se realiza el scrambling del contenido del campo SIGNAL. Esta tarea es realizada por el bloque *Chunks to Symbols* superior.

El resto de la trama se modula utilizando el esquema indicado como parámetro, tarea realizada por el bloque *Chunks to Symbols* inferior, y el cabezal y la trama se juntan mediante el bloque *Tagged Stream Mux*.

El siguiente bloque, *OFDM Carrier Allocator*, divide la cadena de números complejos en grupos de 48 números. Cada uno de dichos grupos es asociado con un símbolo OFDM. En cada grupo, los números complejos se numeran de 0 a 47, y se mapean consecutivamente en subcarriers de OFDM que se numerarán de -26 a -22, -20 a -8, -6 a -1, 1 a 6, 8 a 20, y 22 a 26 respectivamente. Los subcarriers -21, -7, 7 y 21 se saltean puesto que son utilizados para insertar subcarriers de tipo piloto. El subcarrier 0, asociado con la frecuencia central, es omitido y puesto en valores ceros. De esta manera, el número de subcarriers total es de 52 (48 subcarriers con la trama y 4 de tipo piloto).

Para cada grupo de subcarriers desde el -26 al 26, se realiza su conversión desde el dominio de frecuencias al dominio del tiempo mediante el uso de la transformada de Fourier inversa. Esta tarea es realizada por el bloque *FFT*. A la onda final resultante de realizar la transformada de Fourier se le antepone una extensión circular de sí misma, y se trunca la onda final (que es periódica) a la longitud de un símbolo OFDM aplicando una técnica conocida como *Time Domain Windowing*, siendo esta tarea final realizada por el bloque *OFDM Cyclic Prefixer*.

4.2.3. Implementaciones adicionales

Además de los grafos de flujo del transmisor y el receptor que se encuentran adaptados para ser usados con un USRP, la implementación contiene un grafo de flujo de un transceptor (un transmisor y un receptor en el mismo grafo) y otro grafo de flujo que conecta la salida de un transmisor con la entrada de un receptor, simulando un canal de *loopback*[3] entre ellos. Este grafo fue utilizado extensivamente al realizar las pruebas sin antena.

[3] Se denomina *canal de loopback* a aquel que contiene un transmisor y receptor conectados de manera secuencial. Dicho canal se utiliza principalmente para realizar pruebas.

4.3. Profiling

Tal como se describió en la sección anterior, la implementación original (particularmente la del receptor) cuenta con ciertos defectos que hacen que su funcionalidad sea limitada. En particular, la incapacidad de decodificar tramas en el tiempo exigido por el estándar hace que sea imposible su utilización a la par de las implementaciones comerciales, que pueden alcanzar tasas de transferencia máxima de 54 Mbps en el caso que se esté utilizando un esquema de modulación 64QAM con tasa de codificación 3/4.

El *profiling*, palabra que podría traducirse como "perfilaje", consiste en analizar el rendimiento de un software para determinar el tiempo que fue empleado en la ejecución de las distintas partes del mismo. Se suele utilizar como forma de identificar los puntos que consumen mayor tiempo de procesamiento, para luego poder centrarse en su implementación y lograr mejoras de rendimiento a nivel global.

En las siguientes secciones se describen las técnicas de profiling utilizadas para detectar las etapas de la implementación original con mayor peso en el tiempo de ejecución final. En primer lugar, se presenta una perspectiva de alto nivel, que trabaja a nivel de los grafos de flujo de GNU Radio. En segundo lugar, se describe una perspectiva diferente, de bajo nivel, que trabaja a nivel del código C++ que implementa los bloques de mayor consumo e identifica las funciones más demandantes, con el objetivo posterior de optimizarlas.

En esta etapa de profiling existen dos métricas fundamentales que son representativas del rendimiento alcanzado por el programa: el *tiempo de ejecución* de cada bloque y la *tasa de transferencia del programa*.

4.3.1. Profiling de alto nivel

Profiling en el grafo de loopback

GNU Radio dispone de una herramienta para realizar profiling de los grafos de flujo llamada GNU Radio Performance Monitor. Esta herramienta, cuyo funcionamiento fue introducido en la Sección 3.3.5 y cuya instalación se describe en el Anexo A, realiza la medición del tiempo de ejecución de todos los bloques del grafo de flujo que se encuentra en ejecución, permitiendo determinar comparativamente los bloques de mayor peso porcentual.

Haciendo uso de esta herramienta, se realizó el profiling del grafo de loopback[4]. El mismo fue configurado para transmitir un archivo de tamaño arbitrario a través de un canal simulado como canal perfecto. La Tabla 4.1 muestra los porcentajes de ejecución de los diez bloques que más tiempo demandaron para todas las modulaciones

[4] Ver Sección 4.2.3.

Bloque	BPSK (%)	QPSK (%)	16QAM (%)	64QAM (%)
OFDM Decode MAC	41,59	57,49	69,27	76,11
OFDM Carrier Allocator	27,66	24,25	21,54	16,58
OFDM Equalize Symbols	4,74	3,49	1,50	1,59
OFDM Sync Short	3,85	2,12	0,86	0,81
OFDM Mapper	3,21	1,90	1,07	0,70
FFT	2,88	1,08	0,43	0,37
OFDM Sync Long	2,21	1,51	0,79	0,55
Fractional Resampler	1,96	1,19	0,54	0,41
FIR Filter	1,80	1,11	0,57	0,35
Fastnoise Source	1,21	0,81	0,56	0,44

Tabla 4.1: Porcentaje de tiempo de ejecución según bloque de GNU Radio para los esquemas de modulación BPSK, QPSK, 16QAM y 64QAM, con tasa de codificación de 1/2 (2/3 para 64QAM) y PDU de tamaño 1500.

disponibles y con una tasa de codificación de 1/2 (2/3 para 64QAM), para el caso en el cual cada trama MAC tiene 1500 bytes de datos. La Tabla 4.2 muestra la misma información pero para el caso en el cual la trama enviada contiene un PDU de tamaño 50 bytes.

Bloque	BPSK (%)	QPSK (%)	16QAM (%)	64QAM (%)
OFDM Carrier Allocator	78,26	80,29	80,58	70,53
OFDM Decode MAC	7,82	7,82	8,58	19,40
Fastnoise Source	2,35	2,01	1,69	1,53
OFDM Sync Long	1,19	0,86	0,85	0,69
OFDM Equalize Symbols	0,96	0,64	0,48	0,37
FIR Filter	0,92	0,68	0,53	0,42
FFT	0,57	0,66	0,55	0,44
OFDM Sync Short	0,54	0,38	0,33	0,25
Multiply Const	0,53	0,35	0,28	0,11

Tabla 4.2: Porcentaje de tiempo de ejecución según bloque de GNU Radio para los esquemas de modulación BPSK, QPSK, 16QAM y 64QAM, con tasa de codificación de 1/2 (2/3 para 64QAM) y PDU de tamaño 50.

Es importante apreciar que, independientemente de la modulación y el tamaño de la trama, los dos bloques más demandantes siempre son los mismos: OFDM Decode

MAC y OFDM Carrier Allocator. En el caso de la Tabla 4.1, el bloque OFDM Decode MAC es quien tiene un consumo mayor, mientras que el OFDM Carrier Allocator se encuentra en segundo lugar. En el caso de la Tabla 4.2, el orden es el opuesto. Este comportamiento se podrá apreciar numerosas veces a lo largo del presente trabajo, y se debe principalmente a que un paquete de tamaño grande lleva una tarea de decodificación más compleja que uno más pequeño.

También cabe destacar que a medida que el esquema de modulación se hace más complejo, el porcentaje del bloque OFDM Decode MAC se hace más grande aún. Esto es razonable si se considera que dicho bloque es el encargado de demodular la señal recibida, y que demodular un esquema con más puntos en la constelación siempre llevará más tiempo que demodular un esquema simple como BPSK. El bloque OFDM Carrier Allocator, por su parte, disminuye su consumo gradualmente al incrementar el esquema de modulación, debido a que el peso relativo del bloque OFDM Decode MAC aumenta en mayor medida que el de él.

Profiling en el transmisor

También se realizó una etapa de profiling exclusivamente sobre el transmisor, de forma de analizar con más detalle este componente del sistema. En la sección anterior, se vio que la gran mayoría de los bloques más costosos pertenecían al receptor, con excepción del bloque OFDM Carrier Allocator. A pesar de esto, sigue siendo razonable realizar un profiling de forma aislada sobre el transmisor, debido a que es posible un escenario en el cual se tenga una comunicación unidireccional entre este y un router convencional. En este caso, la calidad y velocidad de la comunicación dependerán exclusivamente del transmisor, ya que se asume que el router será capaz de decodificar la señal de acuerdo a las velocidades especificadas en el estándar.

Las tablas 4.3 y 4.4 presentan los datos recabados a partir de las pruebas realizadas con GNU Radio Performance Monitor.

Bloque	BPSK (%)	QPSK (%)	16QAM (%)	64QAM (%)
OFDM Carrier Allocator	60,14	78,91	86,12	87,88
OFDM Mapper	8,26	5,75	3,97	3,96
IFFT	7,97	2,93	1,41	0,89
OFDM Cyclic Prefixer	3,04	1,56	0,72	0,52

Tabla 4.3: Porcentaje de tiempo de ejecución de los bloques más costosos del transmisor gr-ieee802-11. Se presentan los valores obtenidos para los distintos esquemas de modulación BPSK, QPSK, 16QAM y 64QAM, con tasa de codificación de 1/2 (2/3 para 64QAM) y PDU de tamaño 1500.

Se observa que en la Tabla 4.3 el consumo del bloque OFDM Carrier Allocator es el mayor por un margen muy amplio, y aumenta a medida que se elige un esquema

de modulación más complejo. Podría parecer que esta tendencia se contradice con la observada en la Sección 4.3.1, en donde este bloque decrece su consumo porcentual a medida que se opta por un esquema de modulación más complejo. Lo que sucede en realidad es que, si bien el costo del bloque OFDM Carrier Allocator es mayor al aumentar el esquema de modulación, también lo es el de otros bloques como OFDM Decode MAC (y con un factor de aumento más grande), lo que provoca un consumo porcentual menor.

También es posible observar bloques que no se encontraban en la Sección 4.3.1, debido a que al evaluar transmisor y receptor juntos, su costo resultaba demasiado pequeño. Esto es coherente con lo observado en esta tabla, ya que la diferencia entre el primer y el segundo bloque más costoso del transmisor es muy importante.

Bloque	BPSK (%)	QPSK (%)	16QAM (%)	64QAM (%)
OFDM Carrier Allocator	94,32	94,50	94,85	94,42
OFDM Mapper	0,54	0,58	0,61	0,69
IFFT	0,68	0,56	0,48	0,49
OFDM Cyclic Prefixer	0,30	0,26	0,24	0,48

Tabla 4.4: Porcentaje de tiempo de ejecución de los bloques más costosos del transmisor gr-ieee802-11. Se presentan los valores obtenidos para los esquemas de modulación BPSK, QPSK, 16QAM y 64QAM, con tasa de codificación de 1/2 (2/3 para 64QAM) y PDU de tamaño 50.

La Tabla 4.4 muestra claramente cómo el consumo del bloque OFDM Carrier Allocator depende del tamaño de PDU con el cual se están creando las tramas. En este caso, el bloque alcanza un consumo superior al 94 % del tiempo total de procesamiento del transmisor en el sistema. Esto es coherente con lo observado en la Sección 4.3.1, en donde el bloque OFDM Carrier Allocator representaba un porcentaje importante del consumo total del sistema al utilizar un tamaño de PDU más pequeño.

Profiling en el receptor

Para poder realizar profiling del receptor de forma exclusiva, se generó una traza de una señal con el transmisor y se almacenó en un archivo en el disco duro. Luego se modificó el grafo de flujo del receptor para tomar la señal a partir del archivo. En la Tabla 4.5 se presentan los datos recabados.

En este caso, sí es posible identificar varios bloques que también se encuentran presentes en las tablas de la Sección 4.3.1. Esto establece la pauta de que el receptor como componente del sistema es mucho más costoso que el transmisor, posicionando varios de sus bloques por delante de todos los bloques del transmisor (con excepción del bloque OFDM Carrier Allocator).

Bloque	BPSK (%)	QPSK (%)	16QAM (%)	64QAM (%)
OFDM Decode MAC	72,02	85,50	92,15	94,42
OFDM Equalize Symbols	9,54	5,56	3,01	2,11
OFDM Sync Short	8,37	4,1	1,98	1,19
OFDM Sync Long	3,99	2,38	1,24	0,78

Tabla 4.5: Porcentaje de tiempo de ejecución según bloque de GNU Radio para los esquemas de modulación BPSK, QPSK, 16QAM y 64QAM, con tasa de codificación de 1/2 (2/3 para 64QAM) y PDU de tamaño 1500, ejecutando exclusivamente el receptor.

En esta tabla también es posible observar que el costo del bloque OFDM Decode MAC aumenta al complejizar el esquema de modulación utilizado. Esto es razonable debido a que la demodulación es más costosa en estos casos, y la misma se realiza dentro del bloque OFDM Decode MAC. Este resultado coincide con el de la Sección 4.3.1.

Si bien no se incluyó una tabla para la tasa de codificación 3/4, vale la pena aclarar que el costo del bloque OFDM Decode MAC incrementa al usar esta tasa de codificación por sobre 1/2 para un esquema de modulación dado. Esto también es coherente, debido a que en una codificación 3/4, donde se tiene menos información de redundancia en el mensaje transmitido, es necesario que el decodificador convolucional realice un trabajo mayor en la trellis.

Profilers descartados

Debido a que la arquitectura híbrida de GNU Radio define a los bloques como clases de C++ a bajo nivel e implementa *wrappers* en Python sobre estos bloques, es razonable realizar el profiling de los grafos tanto a nivel de las clases de C++ como de estos *wrappers*. Por esta razón, además de GNU Radio Performance Monitor, se consideraron otros dos profilers para determinar las secciones de código con mayor demanda de poder de cómputo. A continuación se hace un resumen de las capacidades de dichos profilers:

- **Valgrind:** Valgrind es una plataforma ampliamente utilizada y muy completa que brinda una gran cantidad de funcionalidades para depurar aplicaciones, permitiendo tanto encontrar errores comunes, como detectar problemas en el manejo de la memoria. Además, Valgrind permite realizar un profiling muy detallado para encontrar cuellos de botella en un programa.

- **profile (profiler nativo de Python):** Debido a que los bloques de GNU Radio están implementados como wrappers en Python (que ejecutan código C++), es posible utilizar el profiler nativo de Python para medir su tiempo de ejecución.

Este profiler, de uso muy sencillo, permite ejecutar un programa Python, retornando una tabla que detalla el tiempo de ejecución que tomó cada función del programa original.

En última instancia, se resolvió por no utilizar estos profilers en esta etapa. Una de las principales razones fue que en varias referencias se mencionaba que este tipo de profilers era muy intrusivo para realizar el profiling de una aplicación en GNU Radio, y que por esta razón la información generada en las trazas resultantes no resultaba confiable.

4.3.2. Profiling de bajo nivel

Además del profiling de alto nivel descrito en la sección anterior, se realizó un profiling más detallado sobre los bloques que fueron identificados como los más demandantes (OFDM Carrier Allocator y OFDM Decode MAC). Este segundo profiling fue realizado sobre las implementaciones en C++ de dichos bloques con el objetivo de lograr ubicar las secciones y/o funciones con mayor demanda de tiempo de ejecución.

Para la realización de estas mediciones, se optó por desarrollar una herramienta muy sencilla que brinda la capacidad de medir el tiempo de ejecución de una sección arbitraria de código. Si bien se podría haber hecho uso de los profilers descartados mencionados en la Sección 4.3.1, la opción no fue tomada en cuenta por la extrema simplicidad y flexibilidad de la herramienta desarrollada.

Nuevamente, al igual que en el profiling de alto nivel, las mediciones fueron realizadas sobre el grafo de loopback.

Profiling de bajo nivel en OFDM Decode MAC

Para el bloque OFDM Decode MAC, las funcionalidades analizadas fueron cuatro: *demodulate*, *deinterleave*, *decode_conv* y *descramble*. Dichas funcionalidades se corresponden con los pasos del mismo nombre detallados en la Sección 4.2.1 para el bloque OFDM Decode MAC. La Tabla 4.6 muestra los porcentajes de ejecución de cada sección luego de transmitir un archivo de tamaño arbitrario por el canal, además del porcentaje correspondiente al resto del código, para cada modulación disponible y con una tasa de codificación de 3/4. Esta tasa de codificación se eligió para forzar al bloque a resolver el puncturing[5], que no existe en una codificación de 1/2.

En base a la Tabla 4.6, resulta claro que la inmensa mayoría del trabajo realizado por el bloque OFDM Decode MAC se corresponde con los pasos de demodulación y decodificación convolucional, siendo el segundo el más demandante de los dos.

[5] Para más información, ver la Sección 2.2.4.

Sección del programa	BPSK (%)	QPSK (%)	16QAM (%)	64QAM (%)
demodulate	26,52	20,23	27,35	42,71
deinterleave	0,48	0,94	0,87	0,28
decode_conv	68,34	75,26	69,25	55,00
descramble	0,41	0,54	0,61	0,58
Resto del código	4,24	3,03	1,92	1,43

Tabla 4.6: Porcentaje de ejecución según funciones dentro del bloque OFDM Decode MAC para todas las modulaciones disponibles y con una tasa de codificación de 3/4.

Profiling de bajo nivel en OFDM Carrier Allocator

En las tablas 4.7 y 4.8 se presentan los resultados obtenidos en la etapa de profiling para este bloque. La implementación de este bloque realiza un *memset* sobre todo el arreglo de salida cada vez que es invocado para su ejecución (antes de asignar datos al arreglo de salida). El valor asignado por el *memset* es 0x00, ya que la finalidad de ejecutar esta instrucción es limpiar el arreglo de posibles valores inválidos. Posteriormente a la ejecución de esta instrucción, se asignan a cada uno de los 48 carriers de datos los símbolos a ser transmitidos. También se asignan los carriers piloto. Todo esto se realiza escribiendo sobre el arreglo de salida.

Es a este *memset* inicial que hace referencia la primera fila de las tablas 4.7, y 4.8. Esta sentencia es muy costosa debido al tamaño del arreglo de salida sobre el cual ejecuta la sentencia, que es de aproximadamente 23 MB.

Sección del programa	BPSK (%)	QPSK (%)	16QAM (%)	64QAM (%)
memset	76,40	88,91	96,72	98,77
Resto del código	23,60	11,09	3,28	1,23

Tabla 4.7: Porcentaje de ejecución según funciones dentro del bloque OFDM Carrier Allocator para todas las modulaciones disponibles y con una tasa de codificación de 1/2 (2/3 para 64QAM), con una PDU de 1500 bytes.

Se puede concluir a partir de las tablas 4.7 y 4.8 que la ejecución del *memset* sobre el arreglo de salida del bloque representa el costo más grande en la ejecución del mismo de forma contundente, independientemente del tamaño del PDU y el esquema de modulación utilizado.

Además, se observa que el peso del *memset* incrementa a medida que disminuye el PDU enviado en cada trama MAC. También incrementa a medida que aumenta la complejidad del esquema de modulación utilizado. Si bien la tasa de codificación utilizada para completar ambas tablas fue la más simple para cada esquema de modulación

(1/2 para todos excepto 64QAM, en el cual se hizo con 2/3), los porcentajes de ejecución para una codificación del tipo 3/4 son prácticamente idénticos en base a pruebas realizadas con estas tasas de codificación.

Sección del programa	BPSK (%)	QPSK (%)	16QAM (%)	64QAM (%)
memset	99,51	99,54	99,58	99,58
Resto del código	0,49	0,46	0,42	0,42

Tabla 4.8: Porcentaje de ejecución según funciones dentro del bloque OFDM Carrier Allocator para todas las modulaciones disponibles y con una tasa de codificación de 1/2 (2/3 para 64QAM), con una PDU de 50 bytes.

En la sección 4.4.2 se explica con más detalle la razón por la cual el arreglo de salida tiene este tamaño, y por qué es que al disminuir el tamaño del PDU el peso porcentual del *memset* en la ejecución aumenta, entre otros detalles de la implementación del bloque.

Además, se presenta la estrategia tomada para eliminar la ejecución de este *memset* y de esta forma reducir el costo del bloque. Por esta razón, en esta sección solamente se presentan los resultados del profiling sobre el bloque.

4.3.3. Simulación: picos teóricos de transferencia

Una prueba interesante para realizar antes de modificar el funcionamiento de los bloques para aumentar su rendimiento es la de evaluar los picos teóricos de transferencia del grafo de flujo. Se define como pico teórico de transferencia a la tasa que se obtiene al reducir el tiempo de ejecución de los bloques más costosos a cero, o a un valor muy cercano a cero. El realizar esta simulación proporciona información muy valiosa, por más de una razón:

- **Es una manera de confirmar las observaciones realizadas en la etapa de profiling.**

  Es esperable que al reducir a cero el costo de los bloques más demandantes la ejecución del programa sea más rápida, incrementando la tasa de transferencia alcanzada por el mismo. Por contraposición, si se redujo el costo de los bloques más costosos y no se logró una mejora en la tasa de transferencia, esto generaría dudas acerca de si la información obtenida en la etapa de profiling es correcta.

  Por lo tanto, los resultados obtenidos de la simulación sirven para afirmar o desacreditar las conclusiones sacadas en la etapa de profiling.

- **Permite aproximar qué tanto se puede optimizar el grafo de flujo.**

  Al reducir el tiempo de ejecución de los bloques más costosos a un tiempo cercano a cero, se obtiene una cota superior de la tasa de transferencia final a la que se

puede aspirar. Esta cota superior es una tasa que en la realidad es imposible de alcanzar, ya que implicaría que el bloque no realice trabajo alguno. Esto no puede suceder nunca en un escenario real debido a que, para producir una salida correcta, el bloque debe realizar operaciones sobre los datos. Esto implica un tiempo de ejecución mayor a cero, sin importar que tan optimizada sea la ejecución de dichas operaciones.

De todas maneras, es un dato útil para tener una idea aproximada de cómo influye la ejecución de cada bloque en la tasa de transferencia final que presenta el programa, ya que la misma no solo es afectada por la ejecución de los bloques, sino por GNU Radio en sí mismo (el scheduler), la velocidad de lectura/escritura en disco duro, entre otros factores.

Simular un costo de ejecución nulo para un bloque no es algo trivial, ni alcanza con comentar todo el código que ejecuta el mismo. Los bloques deben generar una salida coherente para que bloques subsiguientes en el grafo de flujo tengan datos correctos en sus buffers de entrada con los cuales puedan trabajar. De no respetar esta regla, el grafo de flujo deja de funcionar y no hay ningún tipo de información valiosa que se pueda extraer en estos casos.

La estrategia diseñada para lograr que los bloques tengan un tiempo de procesamiento cercano a cero mientras siguen produciendo datos correctos en sus buffers de salida consistió en aprovecharse de que el pipeline de bloques se invoca periódicamente para cada trama que se desea transmitir.

Aquí se explica la estrategia aplicada a un bloque en particular, el OFDM Decode MAC, pero la misma idea es extensible a los otros bloques pertenecientes al conjunto de bloques más costosos (los cuales son OFDM Carrier Allocator, OFDM Mapper, OFDM Sync Short y OFDM Sync Long, entre otros).

Supóngase que mientras se está ejecutando *gr-ieee802-11* no hay modificación en los parámetros de la ejecución mismo (como pueden ser el tamaño de los datos a ser enviados por trama MAC, o el esquema de modulación utilizado). Por ejemplo, se asume que son 1500 bytes de datos para todas las tramas transmitidas durante esa ejecución específica de *gr-ieee802-11*. En este caso, cada vez que se invoque al bloque OFDM Decode MAC para decodificar una trama recibida, se tendrá en su buffer de entrada una secuencia de símbolos desde los cuales, después de que se realice el procesamiento correspondiente, se obtienen los datos de la trama MAC original, equivalentes a 1500 bytes de datos.

Si luego de la primera llamada al bloque OFDM Decode MAC, y por consiguiente, luego de la decodificación de la primera trama, se almacenara el resultado de la misma en un buffer miembro de la clase del bloque, entonces sería posible, para llamadas subsiguientes en las cuales se deba decodificar tramas nuevas, devolver directamente el buffer almacenado. Claro está, estos datos almacenados corresponderían a la decodificación de la trama inicial, que no necesariamente es idéntica a la trama que se

quiere decodificar en una llamada posterior, por lo que el resultado sería incorrecto. Sin embargo, esto no es relevante a los efectos de medir la capacidad de transferencia del programa, debido a que ambas tramas tienen un tamaño de datos de 1500 bytes.

Además, como el resultado devuelto es una decodificación válida de una trama anterior, los bloques posteriores al OFDM Decode MAC deberían funcionar con absoluta normalidad, debido a que no tienen forma de conocer que el valor decodificado no corresponde a los símbolos recibidos por el OFDM Decode MAC. El archivo resultante del lado del receptor tendrá el mismo tamaño que el archivo enviado por el transmisor, que también será idéntico al que tendría si se hubiera decodificado de forma normal, con la única salvedad que el contenido del archivo será una secuencia repetitiva de los 1500 bytes decodificados de la trama inicial.

Aplicando esta estrategia, solo se incurre en el costo de procesamiento para la primera trama de la transferencia. Las N tramas subsiguientes tienen un costo nulo, o muy pequeño (debido a que aún debe copiarse a la salida el buffer miembro de la clase). Por lo tanto, si se realiza la transferencia de un archivo con un tamaño tal que se necesite un número grande de tramas, el promedio de la tasa de transferencia observado sobre un período largo de tiempo tenderá a representar la tasa de transferencia que se obtiene cuando el bloque en cuestión no consume tiempo de procesamiento.

Esta fue la estrategia utilizada para los distintos bloques, experimentando con dichos cambios sobre cada bloque de forma individual en primera instancia, para luego hacer pruebas que involucran varios bloques con este comportamiento.

Resultados obtenidos

Se aplicó esta estrategia sobre varios bloques de *gr-ieee802-11* de forma gradual, comenzando por el bloque más costoso (según los datos recabados por la etapa de profiling). Este proceso generó versiones consecutivas del programa en donde sus bloques fueron modificados para que tuvieran un peso nulo en el tiempo de ejecución. En esta sección se presenta un resumen de las combinaciones probadas para los distintos bloques, y solo se presentan versiones que son acumulativas en la cantidad de bloques con trabajo cero.

En las tablas 4.9 y 4.10 se muestran las tasas de transferencia obtenidas para la versión original y las distintas versiones creadas para tramas con un PDU de 50 y 1500 bytes respectivamente. Como se puede apreciar, hay una mejora sustancial en la tasa de transferencia a medida que se reduce el trabajo de los bloques más costosos, lo cual confirma las observaciones hechas en la etapa de profiling. Asimismo, se realizó la verificación de que los bloques modificados realizan trabajo nulo en GNU Radio Performance Monitor, lo cual también aumenta la confiabilidad de la herramienta.

A partir de los datos presentes en ambas tablas, es posible concluir que disminuyendo el costo de los bloques OFDM Decode MAC y OFDM Carrier Allocator simultánea-

Versión	BPSK (KB/s)	QPSK (KB/s)	16QAM (KB/s)	64QAM (KB/s)
Original	15	15	16	16
Versión 1	15	16	16	16
Versión 2	93	129	155	174
Versión 3	93	131	161	178

Tabla 4.9: Tasa de transferencia de la versión original y distintas versiones producto de reducir el costo de ejecución de los bloques a cero. El tamaño del PDU de datos en cada trama MAC es de 50 bytes, y la tasa de codificación es de 1/2 a excepción de 64QAM donde es 2/3. **Versión 1:** Original sin OFDM Decode MAC. **Versión 2:** Versión 1 sin OFDM Carrier Allocator. **Versión 3:** Versión 2 sin OFDM Mapper.

mente se logra un salto muy grande en las tasas de transferencia. Esto no es casualidad ya que como se vio en la sección anterior, estos son los dos bloques más costosos de todo el programa. La mejora en la tasa conseguida luego de reducir a cero el trabajo de los otros bloques más costosos (OFDM Sync Long, OFDM Sync Short y OFDM Mapper) es menor si se la compara con la obtenida por estos dos bloques, lo que otorga la pauta de que el mayor incremento de rendimiento que se puede alcanzar es a través de la optimización de dichos bloques, tal como se había determinado en la etapa de profiling.

Versión	BPSK (KB/s)	QPSK (KB/s)	16QAM (KB/s)	64QAM (KB/s)
Original	142	167	160	118
Versión 1	292	549	464	481
Versión 2	294	561	1028	1740
Versión 3	298	577	1081	1847

Tabla 4.10: Tasa de transferencia de la versión original y distintas versiones producto de reducir el costo de ejecución de varios bloques a cero. El tamaño del PDU de datos en cada trama MAC es de 1500 bytes, y la tasa de codificación es de 1/2 a excepción de 64QAM donde es 2/3. **Versión 1:** Original sin OFDM Decode MAC. **Versión 2:** Versión 1 sin OFDM Carrier Allocator. **Versión 3:** Versión 2 sin OFDM Mapper.

4.4. Mejoras realizadas

Habiendo identificado las etapas del programa representativas de la mayoría del tiempo de cómputo, se fijó como objetivo acelerar la ejecución de las mismas de forma de poder conseguir un aumento en el rendimiento de la SDR. Se estudiaron distintas estrategias para optimizar el código original del programa, muchas de las cuales fueron descartadas por distintas razones explicadas a continuación.

En esta sección se presenta un breve resumen de la investigación y desarrollo de todas las estrategias evaluadas, haciendo especial mención a la estrategia final elegida para cada uno de los distintos bloques.

4.4.1. Estrategias evaluadas

Modificaciones en la implementación de BLAS y LAPACK

Como se explicó en la Sección 4.2, el bloque encargado de la decodificación de la trama MAC hace un uso extensivo de la biblioteca IT++ para las tareas de demodulación de la señal y decodificación del código convolucional. Por otra parte, como se mostró en la Sección 4.3, este bloque es el que más recursos consume por una amplia diferencia, y dentro de las funciones ejecutadas por el bloque, aquellas pertenecientes a la biblioteca IT++ representan aproximadamente un 90 % del tiempo total de procesamiento utilizado.

Es por esto que es razonable estimar que, si pudiera lograrse una mejora importante en el rendimiento de IT++, habría un aumento en el rendimiento del bloque OFDM Decode MAC en órdenes de magnitud similares, y consecuentemente, en la implementación de *gr-ieee802-11*.

La biblioteca IT++ utiliza BLAS y LAPACK[6] para el manejo de operaciones vectoriales y matriciales, que son comunes en algoritmos de demodulación y decodificación de Viterbi.

Basic Linear Algebra Subprograms (BLAS) es una especificación de rutinas de bajo nivel útiles para realizar operaciones de álgebra lineal convencionales como ser multiplicación de vectores y matrices y combinaciones lineales de vectores, entre otras. Por lo tanto, BLAS es en realidad una interfaz que puede ser implementada de diversas maneras, pero se debe cumplir que cada implementación exponga los mismos servicios. A su vez, *Linear Algebra Package (LAPACK)* es una biblioteca que utiliza las primitivas de BLAS para poder brindar funciones de más alto nivel, como factorización LU, QR y resolución de sistemas de ecuaciones, entre otras.

Existen diversas distribuciones de BLAS y LAPACK disponibles, que implementan la misma interfaz que la distribución de referencia. Específicamente, las implementaciones más veloces son OpenBLAS, y MKL ó ACML dependiendo de la arquitectura que se esté utilizando, ya que las últimas dos son bibliotecas desarrolladas por los fabricantes de microprocesadores Intel y AMD, respectivamente.

La diferencia en el rendimiento de las bibliotecas mencionadas anteriormente, en comparación con otras opciones como ATLAS o la de referencia, es significativa. Los

[6] Para más información, ver [23].

binarios de IT++ obtenidos a partir del manual de instalación de gr-ieee802-11 utilizan la implementación de referencia presente en NetLib. Por esta razón, se consideró reemplazar la versión utilizada por defecto por una de las alternativas más veloces, y evaluar el cambio en el rendimiento del bloque.

Por otra parte, IT++ es compilada a partir de Makefiles creados por la herramienta CMake. CMake es una aplicación diseñada con el fin de permitir la configuración del armado de un programa de forma única y genérica por el desarrollador del mismo. Luego, CMake es capaz de generar automáticamente los Makefiles específicos para la arquitectura y sistema operativo donde se está compilando el programa. De esta forma, el desarrollador programa una vez en alto nivel el armado de sus binarios, y luego CMake genera automáticamente, por ejemplo, Makefiles para la compilación con g++ en Ubuntu, o proyectos Visual C++ para Windows, según corresponda.

Para poder realizar el *linking* de IT++ con OpenBLAS fue necesario crear nuevos scripts que implementasen el concepto de *package* que define CMake, debido a que esta biblioteca no tiene soporte para la compilación directa contra esta implementación de BLAS y LAPACK. Estos packages (scripts) son los encargados de asignar un valor a todos los parámetros relacionados con la configuración del linking de OpenBLAS con IT++, como por ejemplo, la ruta de los *include* correspondientes y de la biblioteca compilada, así como las flags de compilación y linking, entre otras cosas.

Una vez compilada IT++ para que utilizara OpenBLAS, se realizaron pruebas que determinaron una mejora despreciable en el rendimiento. No hubo ninguna métrica relevante que indicara una mejora con respecto a la versión original de IT++. También se experimentó con la biblioteca MKL, debido a que es la implementación más eficiente para procesadores Intel, incluso más que OpenBLAS. La utilización de MKL tampoco mostró ninguna mejora respecto a las otras dos implementaciones.

De esta forma, se estima que la importancia de la eficiencia en la implementación de BLAS y LAPACK utilizada por IT++ no es relevante en cuanto al costo total de procesamiento de esta última. Esto es, al menos, para las funciones utilizadas en la demodulación y decodificación de códigos convolucionales.

Aceleración mediante ejecución híbrida CPU-GPU

El enfoque inicial del proyecto proponía realizar las optimizaciones de rendimiento mediante la utilización de una Unidad de Procesamiento Gráfico (*Graphic Processor Unit, GPU*) para realizar procesamiento de propósito general y de esta forma dividir el costo computacional entre este dispositivo y el procesador.

Sin embargo, luego de la etapa de profiling quedó en evidencia que el único bloque que podría llegar a ser apto para que se le aplicase procesamiento en GPU era el bloque OFDM Decode MAC. Específicamente, las tareas de demodulación, deinterleaving, de-

codificación convolucional y descrambling, que son las más costosas del bloque, podrían ser ejecutadas en GPU.

Un análisis de esta estrategia dejó en evidencia algunos problemas importantes. Por ejemplo, sería necesario invocar un *CUDA Kernel*[7] por cada trama a decodificar. Esto implicaría una transferencia de datos entre la memoria principal y la GPU, la ejecución del algoritmo propiamente dicha, y finalmente otra transferencia desde la GPU hacia la memoria principal para obtener los datos decodificados. La operación de copiar de la memoria de un dispositivo a otro es costosa, y este costo de transferencia podría opacar las mejoras en el tiempo de procesamiento. Esta es una de las principales razones por las cuales muchos algoritmos logran menos desempeño al ser ejecutados en GPU.

En este caso, la invocación se realizaría para cada trama recibida, lo que sugiere que el costo de la transferencia podría degradar el rendimiento de forma significativa, teniendo en cuenta que en una conexión Wi-Fi normal, la cantidad de tramas recibidas por segundo es muy alta. Además, generalmente los algoritmos que muestran una aceleración considerable en GPU trabajan sobre un conjunto grande de datos, y realizan múltiples cálculos. En el caso de la decodificación de una trama MAC, si bien las operaciones son costosas, el tamaño de los datos sobre los que se trabaja puede considerarse pequeño para ser abordado siguiendo una estrategia de GPU, debido a que el tamaño máximo del PDU es de 1500 bytes por trama. En el caso de paquetes más pequeños, el costo de realizar el procesamiento en GPU resulta aún mayor.

Por esta razón, se decidió acelerar la ejecución utilizando la CPU en primera instancia, y en caso de no alcanzar una mejora significativa, volver a evaluar la posibilidad de utilizar la GPU.

Vale la pena aclarar que podría ser atractiva la posibilidad de utilizar la GPU para realizar la decodificación *batch* de múltiples tramas. Una aplicación de esto podría ser la recepción de un archivo, ya que en este caso este último solo es útil cuando es recibido en su totalidad. Por esta razón, en este escenario es posible decodificar todas las tramas en conjunto al finalizar la transferencia, ya que solo interesa el momento en que se decodifican *todas* las tramas y el archivo es recibido de forma íntegra. Sin embargo, para otros casos de uso, como por ejemplo un chat, la estrategia de procesamiento *batch* no tiene sentido. Por esta razón, para no perder generalidad en la solución, se evitó esta estrategia.

Aceleración mediante el uso de Open Multi-Processing

Open Multi-Processing (OpenMP) es una API utilizada para crear de forma sencilla programas *multi-threading* que aprovechen los recursos de cómputo en una arquitectura multiprocesador.

[7] Los *CUDA Kernels* son las unidades de ejecución de código en GPU que brinda la biblioteca CUDA, que es la plataforma de procesamiento paralelo en GPU de la empresa NVIDIA para sus GPUs.

El equipo utilizado en el presente trabajo posee una CPU Intel Core i5 2400 de 4 núcleos, por lo que el uso de OpenMP debería, a priori, incrementar el rendimiento. Se realizaron modificaciones para utilizar OpenMP en varias secciones del código que contenían operaciones sobre vectores y que además se podían aplicar concurrentemente.

Sin embargo, los resultados obtenidos a partir de las pruebas de rendimiento luego de los cambios no mostraron ningún beneficio con respecto al rendimiento original.

Esto se debe principalmente a que GNU Radio define un *thread* por cada bloque del grafo de flujo, por lo que es extremadamente difícil que durante la ejecución del grafo de flujo existan núcleos del microprocesador en estado ocioso. Debido a esto, no se logra ningún beneficio al definir threads separados para este tipo de operaciones vectoriales. De hecho, el pequeño costo extra agregado por la creación de los distintos threads es, en este caso, contraproducente.

Sin embargo, es posible que existan situaciones en las cuales utilizar OpenMP mejore el rendimiento de un grafo de flujo creado en GNU Radio. Por ejemplo, en el caso que un bloque sea costoso y se encuentre cerca del principio del pipeline de bloques del grafo de flujo, puede suceder que no sea capaz de producir datos lo suficientemente rápido como para que los bloques posteriores del pipeline siempre tengan datos disponibles en sus buffers de entrada. En este caso, utilizar OpenMP en el bloque costoso puede acelerar el rendimiento debido a que se disminuye el tiempo en el cual se encuentran bloques en estado ocioso por no tener datos suficientes en sus buffers de entrada.

Este no es el caso de *gr-ieee802-11*, principalmente porque los bloques costosos se encuentran en el final del pipeline de bloques, reduciendo las oportunidades de que bloques posteriores estén ociosos por la lentitud de la ejecución de los primeros.

4.4.2. Asignación de carriers

Como se explicó en la Sección 4.3, el bloque OFDM Carrier Allocator es el segundo bloque de mayor costo de todo el transceptor, y el más costoso cuando solo se tiene en cuenta al transmisor. Este bloque se invoca una vez por cada trama que la capa MAC envía para que sea transmitida, y su función es asignar símbolos provenientes de un *stream* a los distintos carriers de datos.

Además, se cumple que cuanto más pequeño es el tamaño de la trama y más simple el esquema de modulación, más grande es el costo relativo de este bloque en comparación con el resto. Esta observación es razonable debido a que un tamaño de trama más pequeño implica la necesidad de enviar un mayor número de tramas para transmitir una misma cantidad de datos. Esto no solo significa que el costo relativo del cabezal de la trama es mayor, sino que también, al haber un mayor número de tramas, se realizan más invocaciones al bloque OFDM Carrier Allocator, incrementando su consumo.

En el caso de que solo se cambie el algoritmo de modulación por uno más simple, también es lógico observar un incremento en el consumo de este bloque[8]. Al simplificar la modulación, y consecuentemente, la demodulación, se logra que los bloques OFDM Mapper y OFDM Decode MAC, responsables de estas tareas, disminuyan su consumo, por lo que el peso relativo del bloque OFDM Carrier Allocator aumenta. Adicionalmente, al tener un esquema de modulación más simple, la cantidad posible de bits codificables por símbolo es menor, lo que obliga a un mayor número de símbolos a ser asignados en el bloque OFDM Carrier Allocator para un mismo conjunto de datos.

En la Sección 4.3.2 se comprobó que el consumo del bloque OFDM Carrier Allocator se debe principalmente a la ejecución de un *memset* que asigna el valor 0x00 sobre el buffer de datos de salida del bloque, que tiene un tamaño de aproximadamente 23 MB. Evitando la ejecución de esta sentencia, el costo del bloque decrece en forma general en un 90 %. De esta manera, optimizar el rendimiento de este bloque se traduce en lograr optimizar el rendimiento del *memset*.

La ejecución del *memset* se produce cada vez que el scheduler de GNU Radio invoca a la función *work()* del bloque, lo que quiere decir que se ejecuta una vez por cada trama a ser transmitida. Es posible observar que el tamaño del buffer de salida es excesivamente grande para los datos que luego se escriben en el mismo y que son utilizados por el bloque siguiente (la FFT), habiendo muchas posiciones de memoria que no son utilizadas. Asimismo, el tamaño del buffer siempre es el mismo, independientemente de la cantidad de símbolos que estén disponibles en el buffer de entrada.

Esto se debe a que en la definición del grafo de flujo en el archivo Python correspondiente, el bloque OFDM Carrier Allocator tiene asignado explícitamente un valor mínimo para el buffer de salida, que acota inferiormente su tamaño. De no existir esta asignación explícita, GNU Radio utiliza el tamaño de buffer de salida por defecto, que es 64 KB, lo que equivale a 8192 valores de tipo *gr_complex* y no es suficiente para casos de tramas grandes. Por esta razón, en la implementación original se asigna un valor lo suficientemente grande como para cubrir el escenario en el cual se transmite una trama con 1500 bytes de datos. En el caso de este bloque, asignar un buffer de salida excesivamente grande penaliza la performance en los casos en que no se utilizan bloques con 1500 bytes de datos, debido a que en el código de ejecución se hace un *memset* de este buffer.

La mejora que se hizo con respecto a este bloque fue eliminar por completo la ejecución del *memset* al comienzo de la función work(). La utilidad de este *memset* es dejar en un estado consistente el contenido del buffer de salida del bloque (con el valor 0), que es administrado por el scheduler de GNU Radio y que potencialmente puede tener valores inválidos en celdas del mismo al momento de la ejecución de work(). Sin embargo, el hacer *memset* del arreglo entero, además de ser costoso, es innecesario, debido a que la gran mayoría de los valores de las celdas del arreglo son reasignados antes de que termine la ejecución del bloque.

[8] Esto se refiere al caso del grafo de *loopback*.

De esta forma, es posible escribir en dichas celdas directamente, reduciendo la cantidad de asignaciones hechas a cada celda de dos a una. Se recuerda que como se vio en la Sección 3.1.2, el esquema OFDM utilizado en IEEE 802.11 emplea 64 carriers, 48 de datos, 4 de tipo piloto, y 12 carriers nulos. Tanto los carriers de datos como los pilotos, son asignados en el OFDM Carrier Allocator original, por lo que simplemente se hizo necesario asignar en el buffer de salida el valor 0x00 a las posiciones correspondientes a los 12 carriers nulos.

Debido a que el bloque OFDM Carrier Allocator es un bloque genérico utilizado en todos los esquemas de transmisión OFDM y que pertenece a la distribución oficial de bloques de GNU Radio, no es viable realizar las modificaciones propuestas directamente en el código del mismo. Esto se debe a que las modificaciones son útiles solo para el caso en que se esté utilizando el bloque para modelar la asignación de carriers según la especificación del estándar IEEE 802.11. Por esta razón, se creó un bloque nuevo bajo el nombre de "OFDM Carrier Allocator IEEE 802.11" que tiene las modificaciones mencionadas sobre la implementación original, y se asoció dicho bloque al módulo de IEEE 802.11. En el Anexo B se describen en detalle los pasos que fueron necesarios para crear dicho bloque en GNU Radio.

4.4.3. Mejoras en el algoritmo de demodulación

El algoritmo de demodulación utilizado, parte de la biblioteca IT++, suponía el 30 % del tiempo de ejecución del bloque OFDM Decode MAC. La implementación por defecto de IT++, que era la que se utilizaba, no realizaba diferenciación alguna dependiendo de la modulación; esto es, se invocaba un algoritmo genérico que tenía como entrada un conjunto de números complejos representando los puntos de la constelación y, para cada uno de estos puntos, realizaba una iteración sobre cada punto base de la constelación elegida para determinar el que se encontraba más cerca, determinando a continuación el mapeo final en base a este punto.

Puede entenderse la inclusión de este algoritmo genérico en una biblioteca de amplio alcance como IT++ debido a la variedad de esquemas de modulación que se utilizan en distintos ámbitos de la realidad. Sin embargo, en el contexto de su uso particular en una implementación del estándar WiFi, en donde las modulaciones que se utilizan son conocidas de antemano (BPSK, QPSK, 16QAM y 64QAM), se determinó que crear funciones nuevas que hicieran las demodulaciones específicas sería mucho más conveniente y consumiría menos tiempo en última instancia.

Por lo tanto, se simplificó el algoritmo original adaptándolo a cada una de las modulaciones. En el caso de BPSK, el nuevo algoritmo simplemente debe verificar si la parte real del complejo recibido como entrada es mayor o menor a 0, y en base a esto demodularlo como un 0 o un 1 (según el mapeo). Para QPSK, esto mismo se aplica pero distinguiendo según el cuadrante en que se encuentra el punto complejo. Ideas similares también aplican para 16-QAM y 64-QAM.

4.4.4. Mejoras en el algoritmo de decodificación convolucional

Como se explicó en secciones anteriores, el estándar Wi-Fi utiliza códigos convolucionales para la detección y corrección de errores. Además, en el mismo se recomienda explícitamente utilizar el algoritmo de Viterbi para la decodificación de los códigos convolucionales. En la versión original del proyecto *gr-ieee802-11*, se utiliza la biblioteca *IT++* para la implementación del decodificador de Viterbi.

El tiempo de procesamiento que consume esta implementación del decodificador de Viterbi es considerable, siendo aproximadamente el 63 % de la totalidad de tiempo de cómputo del bloque *OFDM Decode MAC*, que además es el bloque más costoso de todo el transceptor. Por esta razón, el optimizar la ejecución de esta funcionalidad se convirtió en uno de los objetivos principales del presente trabajo.

Debido a que la funcionalidad de códigos convolucionales que provee IT++ es muy genérica, se evaluó la eficiencia de la biblioteca en este caso concreto.

Puesto que la decodificación de Viterbi es sumamente popular, es utilizada no solo por el estándar IEEE 802.11, sino también en estándares de TV Digital (DVB-T) y comunicaciones radiosatelitales, entre otras. Por esta razón, se consideraron otras implementaciones de decodificadores de Viterbi. La estrategia consistió, en caso de que dichas implementaciones fueran superiores a la original de IT++, evaluar la factibilidad de realizar una integración con *gr-ieee802-11* y el eventual costo de dicha tarea.

Existen varias implementaciones candidatas, dos de ellas respaldadas por la comunidad de GNU Radio por pertenecer a la distribución oficial de la herramienta. La primera de ellas se encuentra dentro de la sección *gr-fec* de la distribución oficial de GNU Radio. En esta sección se tienen bloques que están especializados en códigos de corrección de errores de tipo FEC. Esta implementación se encuentra altamente optimizada ya que posee su propio núcleo VOLK.

La segunda implementación se encuentra en la sección *gr-dvbt* de la distribución oficial de GNU Radio a partir de la versión 3.7.9. Esta implementación fue diseñada especialmente para lograr un *throughput* muy alto, debido a las tasas requeridas por los estándares de TV Digital.

Se optó por utilizar esta última implementación debido a la gran cantidad de opiniones positivas que se encontraron en el ámbito académico, así como benchmarks comparando esta implementación con otras (como la de IT++ y la del decodificador de gr-fec) que determinaron que tenía el mayor rendimiento, por un amplio margen. Los benchmarks publicados indicaban que la misma alcanzaba un *throughput* máximo de 30 Mbits por segundo.

Esta implementación es el resultado de varias iteraciones sobre una implementación inicial de un decodificador de Viterbi creado por P. Karn en 1995 [17]. Esta primera versión fue mejorada posteriormente por M. Ettus y finalmente B. Diaconescu realizó las modificaciones necesarias para que el algoritmo utilizara instrucciones *Streaming*

SIMD Extensions 2 (SSE2)[9]. El conjunto de instrucciones SSE2 sirve para paralelizar operaciones vectoriales y en forma general, operaciones del tipo Single Instruction Multiple Data (SIMD).

Integración con *gr-ieee802-11*

La integración se hizo siguiendo un enfoque iterativo e incremental en el sentido de que los cambios necesarios para que ambos componentes funcionaran en conjunto se hicieron de forma gradual y verificando en cada paso el logro del objetivo propuesto. En esta sección se describen de forma resumida los cambios más importantes que fueron necesarios para poder integrar ambos componentes.

El primer paso para integrar el decodificador de Viterbi en *gr-ieee802-11* fue modificar los polinomios generadores del código convolucional utilizados por el decodificador de Viterbi. Para lograr esto, fue necesario recalcular las llamadas a una macro C y los parámetros de la misma, utilizada por el algoritmo principal. Esta macro tiene el nombre de *Butterfly*[10].

Las invocaciones que deben realizarse a Butterfly y los parámetros con los cuales debe invocarse son calculados automáticamente por un script, creado por P. Karn y distribuido como un utilitario junto a la versión original del algoritmo. Este script, llamado *genbut*, recibe como parámetros los polinomios generadores que se desea utilizar en formato octal, y devuelve como salida la lista de llamadas a la macro Butterfly con los parámetros correspondientes. Una vez que se tienen estas llamadas, solo hace falta reemplazarlas en el fuente que realiza la decodificación de Viterbi para que el decodificador trabaje con los polinomios generadores elegidos.

El siguiente paso consistió en adaptar las interfaces entre el decodificador de Viterbi y el bloque llamador (que se recuerda, es el OFDM Decode MAC). Como se mencionó anteriormente, el decodificador de Viterbi fue desarrollado para TV Digital, bajo la sección gr-dvbt de GNU Radio. El mismo está programado como un bloque de GNU Radio, por lo que no es posible una integración directa con un programa genérico programado en C++. Esto no es compatible con el diseño original de *gr-ieee802-11*, donde la decodificación del código convolucional se encuentra embebida dentro del bloque OFDM Decode MAC, que además realiza varias tareas más, como demodulación, deinterleaving, y descrambling.

Para resolver este problema, se evaluaron dos posibles soluciones:

- Fragmentar el bloque OFDM Decode MAC en tres bloques, uno que realice la demodulación y el deinterleaving de la señal, un segundo bloque que realice la de-

[9] Para más información, ver [17] y [18].

[10] Butterfly, que significa mariposa en inglés, es el nombre que se le dio a la macro debido a la apariencia que generan en el grafo de trellis las operaciones de ACS. Para más información, ver [17].

codificación convolucional (que sería el que ya proporciona gr-dvbt), y un último bloque que realice el descrambling.

- Separar la lógica del bloque de GNU Radio del decodificador de Viterbi propiamente dicho, de forma de poder utilizar el decodificador desde cualquier programa en C++, y consecuentemente, desde la implementación del bloque OFDM Decode MAC.

A pesar de que la primera opción es claramente más sencilla ya que la integración es más simple, se optó por la segunda por más de una razón:

- Se respeta el diseño original del autor de *gr-ieee802-11*.
- Se evita el costo extra que se genera al agregar dos nuevos bloques al grafo de flujo en el Scheduler de GNU Radio.

Tanto el tipo de datos como los valores que representan a los bits y la forma de representar los bits omitidos (en el caso de que se esté utilizando puncturing) son distintos para el decodificador de Viterbi de IT++ y el de gr-dvbt. Esto implicó que fuera necesario modificar las funciones de deinterleaving y descrambling del bloque OFDM Decode MAC, ya que la primera genera los datos de entrada para el decodificador, y la última realiza el descrambling a partir de la salida generada por el decodificador, por lo que fue necesario adaptar estas funciones para que utilicen los nuevos tipos de datos.

También hubo que modificar la forma en la que se invocaba al algoritmo de decodificación con respecto a como lo hacía el bloque GNU Radio de gr-dbvt. En la implementación de este último, las tramas se recibían como un flujo de bits codificados, lo que significaba que para la decodificación del código convolucional, de los primeros bits de una trama se estaban utilizando los últimos bits de la trama inmediatamente anterior. Esto no tiene sentido en IEEE 802.11 ya que la codificación del código convolucional de una trama es totalmente independiente de la codificación de otras. Por lo tanto, esto debía verse reflejado del lado del receptor al realizar la decodificación.

El cambio realizado en el decodificador convolucional presentó mejoras sustanciales de manera inmediata tanto al evaluar el porcentaje de tiempo de procesamiento consumido por el bloque como al evaluar la tasa de transferencia neta lograda por el programa *gr-ieee802-11*. En el Capítulo 5 se describen con detalle las mejoras logradas.

Capítulo 5

Evaluación experimental

Este capítulo describe las pruebas realizadas sobre la versión original del programa *gr-ieee802-11*, y sobre una versión modificada que contiene las mejoras detalladas en el Capítulo 4. En primer lugar, se presentan las pruebas realizadas en grafos de GNU Radio funcionando completamente en software. En segundo lugar, se incluyen las pruebas de concepto realizadas en hardware USRP Ettus B210.

5.1. Pruebas en software

Se describen a continuación las pruebas realizadas en software para comparar el desempeño de la implementación original frente a la implementación con las mejoras propuestas en este trabajo. Todas las pruebas fueron realizadas sobre el grafo de loopback[1], modificando el tamaño de la PDU, los esquemas de modulación y las tasas de codificación utilizados.

El grafo se modificó para realizar el envío de un archivo de tamaño fijo. De esta manera, la tasa de transferencia inyectada en el canal fue la más rápida posible, limitada únicamente por el hardware de la computadora utilizada (particularmente por la velocidad del disco duro) y mucho mayor que la tasa de transferencia real (de forma tal que no fuera el cuello de botella del canal).

Debido a que la entrada de datos al sistema se realiza sin ningún control de flujo, podría llegar a ocurrir que se inserten datos a una velocidad mayor de la que el sistema puede procesar. En este caso, los buffers de entrada de aquellos bloques que no puedan consumir muestras a la velocidad requerida comenzarán a llenarse, hasta que en determinado momento el scheduler de GNU Radio comenzará a descartar muestras debido a que el buffer de algún bloque se encuentra completamente lleno. Por esta razón, se realizaron modificaciones a la configuración de GNU Radio para incrementar el tamaño

[1] Ver Sección 4.2.3.

por defecto de los buffers de cada bloque, de forma de evitar que existan datos que puedan llegar a ser descartados.

En cualquier caso, por más de que no existan muestras descartadas, muchas de ellas serán encoladas en buffers de entrada de bloques, y procesadas a una velocidad menor en la versión original con respecto a la versión optimizada. Esto es esperable y es lo que permite que un archivo sea transmitido de forma correcta por ambas implementaciones, pero con un tiempo total de transmisión variable de una versión del programa a la otra.

Las pruebas fueron realizadas sobre una computadora de escritorio con un procesador Intel Core i5 2400 funcionando a 2.4 GHz, con 8 GB de memoria RAM DDR 3 disponibles y operando en un sistema operativo Ubuntu 14.04.3 LTS. La versión de GNU Radio utilizada fue la 3.7.8. Las mediciones fueron tomadas mediante una utilidad de creación propia. Dicha utilidad permite medir la tasa de transferencia desde que se introducen los datos en el transmisor hasta que son recibidos correctamente en el receptor.

Para comprender de mejor manera las tasas obtenidas en las pruebas presentadas en este capítulo, es importante tener en claro la tasa de transferencia teórica máxima que tiene cada esquema de modulación y codificación posible. Dicha información puede encontrarse en la Sección 3.1.3, en la Tabla 3.1.

5.1.1. Pruebas individuales

La primera serie de mediciones fue realizada con el objetivo de comparar por separado cada una de las mejoras propuestas en la Sección 4.4. Esto se hace con el objetivo de poder determinar el peso relativo de cada una de ellas, así como de descubrir ciertos comportamientos que pueden llegar a presentarse.

Versión	BPSK (KB/s)	QPSK (KB/s)	16QAM (KB/s)	64QAM (KB/s)
Original	15	15	15	15
Mejora 1	60	68	69	54
Mejora 2	15	15	16	16
Mejora 3	15	15	16	16

Tabla 5.1: Tasas de transferencia obtenidas al utilizar el grafo de loopback para transmitir un PDU de 50 bytes de información y aplicando cada una de las mejoras realizadas por separado.

La Tabla 5.1 presenta las tasas de transferencia resultantes de aplicar las mejoras propuestas de manera individual con un PDU de 50 bytes. La Mejora 1 es la mejora realizada sobre el bloque OFDM Carrier Allocator, la Mejora 2 sobre la sección *demodulate_bits* del bloque OFDM Decode MAC, y la Mejora 3 la realizada sobre la función

decode_conv del mismo bloque. Las tasas de transferencia son calculadas para cada esquema de modulación disponible, con tasa de codificación 1/2 para BPSK, QPSK y 16QAM y 2/3 para 64QAM.

Es interesante observar que la única de estas mejoras que logra un incremento de desempeño significativo es la Mejora 1. Este comportamiento coincide con el esperado, ya que como fue presentado en las tablas 4.2 y 4.8, ante paquetes pequeños, el bloque OFDM Carrier Allocator representa más del 80 % del consumo total de CPU, desplazando completamente al bloque OFDM Decode MAC que predomina en paquetes más grandes.

En otras palabras, ante paquetes pequeños, el cuello de botella del sistema es el transmisor y no el receptor. Esto tiene sentido ya que si un paquete es pequeño, el receptor debe realizar menos trabajo para decodificarlo.

Versión	**BPSK (KB/s)**	**QPSK (KB/s)**	**16QAM (KB/s)**	**64QAM (KB/s)**
Original	128	162	162	123
Mejora 1	128	165	165	123
Mejora 2	158	194	203	175
Mejora 3	238	407	470	313

Tabla 5.2: Tasas de transferencia obtenidas al utilizar el grafo de loopback para transmitir un PDU de tamaño 1500, aplicando cada una de las mejoras realizadas por separado. Las mejoras son las mismas de la Tabla 5.1, al igual que las modulaciones utilizadas.

Por otra parte, la Tabla 5.2, que describe el comportamiento de cada mejora con PDUs de tamaño 1500, también coincide con los resultados previstos en las tablas 4.1 y 4.6. En este caso, el cuello de botella del sistema es el receptor, y la explicación es análoga a la que se dio anteriormente. Cuando el paquete es grande, el transmisor tiene más bytes para enviar, pero debido a que su costo más grande es una sentencia que es ejecutada independientemente del tamaño del paquete, el trabajo realizado por el transmisor no es mucho mayor al que realiza cuando envía paquetes pequeños. Sin embargo, el receptor tiene mucho más costo, ya que su trabajo sí es proporcional al tamaño del paquete. Por ende, la Mejora 1 no representa ningún cambio en las tasas de transferencia; pero las mejoras 2 y 3, que en su conjunto aumentan el rendimiento del bloque OFDM Decode MAC, permiten aumentar las tasas de transferencias finales.

Resulta interesante observar que, en todos los casos, utilizar un esquema de modulación más complejo no produce grandes mejoras. En particular, el esquema de modulación 64QAM es significativamente más lento que 16QAM y que QPSK. Este comportamiento, que es el mismo que se puede observar en la implementación original, ocurre debido a que las secciones no optimizadas actúan como cuello de botella del sistema, impidiendo a este beneficiarse de la utilización de un esquema de modulación

más complejo (y que requiere mayor cantidad de cálculos). Esto sugiere que las mejoras deben ser aplicadas de manera simultánea para conseguir los mejores resultados.

5.1.2. Pruebas acumulativas

A pesar de que los números de la Sección 5.1.1 muestran que se obtienen mejoras de desempeño importantes al aplicar las modificaciones propuestas en la Sección 4.4, es razonable que el mejor resultado se obtenga cuando todas las mejoras son aplicadas de manera simultánea. Para demostrar esto, en esta sección se presentan dos tablas con las mediciones obtenidas al aplicar de manera acumulativa las tres mejoras propuestas.

Versión	BPSK (KB/s)	QPSK (KB/s)	16QAM (KB/s)	64QAM (KB/s)
Original	15	15	15	15
Mejora 1	60	68	69	54
Mejoras 1 y 2	71	75	79	62
Mejoras 1, 2 y 3	89	124	149	162

Tabla 5.3: Tasas de transferencia obtenidas al utilizar el grafo de loopback para transmitir PDUs de 50 bytes, aplicando cada una de las mejoras realizadas de manera acumulativa. Las mejoras son las mismas de la Tabla 5.1, al igual que las modulaciones utilizadas.

La Tabla 5.3 muestra las tasas de transferencia obtenidas al aplicar las mejoras de manera acumulativa con PDUs de 50 bytes. Cabe destacar que luego de aplicar la Mejora 1, el peso total del bloque OFDM Carrier Allocator disminuye notoriamente, propiciando el aumento porcentual del peso del bloque OFDM Decode MAC. Es por esto que, a diferencia del caso presentado en la Tabla 5.1, las mejoras 2 y 3 sí producen incrementos en el desempeño, aunque estos son menores.

También puede apreciarse que al aplicar las tres mejoras de manera simultánea, el sistema adquiere la capacidad de decodificar los esquemas de modulación más pesados de manera más rápida. Como efecto de esto, la decodificación de 64QAM, el esquema más pesado y costoso, puede realizarse en el tiempo necesario para dejar de ser el cuello de botella del sistema, lo que hace que por primera vez su tasa de transferencia supere a la de 16QAM.

La Tabla 5.4 muestra el mismo resultado pero al utilizar PDUs de tamaño 1500 bytes. En este caso se puede observar que es necesario aplicar mejoras en el receptor para conseguir un aumento en el desempeño, lo cual coincide con el argumento presentado en la Sección 5.1.1 de que ante paquetes grandes, el cuello de botella del sistema es el receptor.

Además de esto, también puede observarse que al aplicar las tres mejoras de manera simultánea el sistema adquiere la capacidad de decodificar los esquemas de modulación

Versión	BPSK (KB/s)	QPSK (KB/s)	16QAM (KB/s)	64QAM (KB/s)
Original	128	162	162	123
Mejora 1	128	165	165	123
Mejoras 1 y 2	155	202	210	187
Mejoras 1, 2 y 3	263	506	881	1289

Tabla 5.4: Tasas de transferencia obtenidas al utilizar el grafo de loopback para transmitir PDUs de 1500 bytes, aplicando cada una de las mejoras realizadas de manera acumulativa. Las mejoras son las mismas de la Tabla 5.1, al igual que las modulaciones utilizadas.

más pesados de manera más rápida, comportamiento similar al descrito para PDUs de 50 bytes. Esto permite que utilizar esquemas más grandes redunden en una tasa de transferencia mucho mayor.

Por otra parte, es interesante observar que independientemente del esquema de modulación elegido, las tasas de transferencia obtenidas con PDUs de 1500 bytes son mucho mayores que las que pueden obtenerse con PDUs de 50 bytes. Esto tiene sentido si se considera que al enviar un paquete más grande (y siempre y cuando el receptor pueda decodificar paquetes en el tiempo necesario), se está enviando más información útil por cada cabezal MAC, lo que redunda en un menor trabajo para el sistema en general, que debe detectar y decodificar menos paquetes.

5.1.3. Pruebas de sistema

Para concluir las pruebas sobre software, esta sección presenta una serie de mediciones realizadas para comparar directamente la implementación original con la final. En ese sentido, estas pruebas pueden ser entendidas como pruebas de caja negra, con el único propósito de determinar el incremento de desempeño producido por las modificaciones introducidas.

Las tablas contienen los datos de tasas de transferencia promedio, máxima y mínima, calculados a partir de mediciones por intervalos de un segundo para una misma ejecución. También se calcula la aceleración (también conocida como *speed-up*) dividiendo la transferencia promedio de la versión modificada entre la transferencia promedio de la versión original.

Tanto en la Tabla 5.5 como en las tablas 5.6 y 5.7 se pueden apreciar los siguientes resultados:

- Observando las tablas, es claro que la implementación original no se ve beneficiada de utilizar esquemas de modulación más grandes (y si lo hace, generalmente el incremento es marginal). La razón de esto se encuentra explicada en la Sección 5.1.1. Al aplicar todas las modificaciones, tanto el receptor como el transmisor pueden operar a una velocidad mucho mayor; en particular, los esquemas de

Modulación	*Versión original* Promedio (KB/s)	Máx. (KB/s)	Mín. (KB/s)	*Versión modificada* Promedio (KB/s)	Máx. (KB/s)	Mín. (KB/s)	**Aceleración**
BPSK 1/2	135	144	88	244	250	217	1,81
BPSK 3/4	131	139	77	348	355	279	2,66
QPSK 1/2	154	168	105	430	443	352	2,79
QPSK 3/4	135	149	89	574	592	469	4,25
16QAM 1/2	139	158	106	678	732	463	4,88
16QAM 3/4	120	126	113	873	989	258	7,28
64QAM 2/3	88	89	84	989	1121	714	11,24
64QAM 3/4	89	91	77	1040	1147	989	11,69

Tabla 5.5: Tasas de transferencia promedio, máxima y mínima de la implementación original y de la modificada para cada esquema de modulación y tasa de codificación disponible, con un PDU de 500 bytes.

modulación más complejos pueden ser decodificados en menos tiempo, y la ventaja de su utilización se hace obvia, permitiendo que 64QAM sextuplique la tasa de transferencia lograda en BPSK. El resultado final de esto es una aceleración considerable para los esquemas de modulación más altos (16QAM y 64QAM), y más grande aún con puncturing.

Modulación	*Versión original* Promedio (KB/s)	Máx. (KB/s)	Mín. (KB/s)	*Versión modificada* Promedio (KB/s)	Máx. (KB/s)	Mín. (KB/s)	**Aceleración**
BPSK 1/2	142	146	134	261	267	235	1,84
BPSK 3/4	137	140	97	378	389	129	2,76
QPSK 1/2	167	172	113	487	498	415	2,92
QPSK 3/4	149	161	107	670	680	586	4,50
16QAM 1/2	160	166	115	840	853	790	5,25
16QAM 3/4	139	145	107	1057	1161	978	7,60
64QAM 2/3	118	124	95	1198	1249	1107	10,15
64QAM 3/4	115	121	91	1222	1318	1042	10,63

Tabla 5.6: Tasas de transferencia promedio, máxima y mínima de la implementación original y de la modificada para cada esquema de modulación y tasa de codificación disponible, con un PDU de 1000 bytes.

- Existe cierta fluctuación en las tasas de transferencias, que se puede apreciar observando los valores máximo y mínimo para cada modulación. Los valores mínimos llegan a ser bastante más bajos en algunos casos (por ejemplo, para 16QAM 3/4 con PDU de 500 bytes). Los valores mínimos generalmente se dan al principio de la transferencia, cuando los buffers de GNU Radio están llenándose.

Modulación	*Versión original* Promedio (KB/s)	Máx. (KB/s)	Mín. (KB/s)	*Versión modificada* Promedio (KB/s)	Máx. (KB/s)	Mín. (KB/s)	**Aceleración**
BPSK 1/2	128	133	111	264	275	250	2,06
BPSK 3/4	134	137	127	392	416	353	2,93
QPSK 1/2	163	168	115	504	530	483	3,09
QPSK 3/4	147	152	109	703	744	669	4,78
16QAM 1/2	158	175	117	876	941	836	5,54
16QAM 3/4	144	147	109	1129	1195	1069	7,84
64QAM 2/3	123	125	96	1209	1369	1119	9,83
64QAM 3/4	121	125	98	1289	1413	1163	10,65

Tabla 5.7: Tasas de transferencia promedio, máxima y mínima de la implementación original y de la modificada para cada esquema de modulación y tasa de codificación disponible, con un PDU de 1500 bytes.

Por otra parte, existe cierta fluctuación producto de que la máquina utilizada nunca está completamente dedicada a la ejecución de GNU Radio, por lo que no acapara completamente el procesador.

- Aunque utilizar una tasa de codificación más alta generalmente implica un aumento en la tasa de transferencia (porque a más tasa de codificación, menos información redundante enviada), la implementación original no se veía beneficiada por esto. Como fue mencionado en las Sección 5.1.1, la decodificación de paquetes realizada por el receptor es el cuello de botella cuando se envían paquetes de gran tamaño. Una mayor tasa de codificación significaba la existencia de puncturing, lo que sobrecargaba aún más al receptor, que debía realizar más cálculos que los que realizaba al no haber puncturing (tasa de codificación de 1/2). Al realizar las modificaciones propuestas, el receptor consigue decodificar los paquetes de manera mucho más rápida, por lo que esto ya no es un cuello de botella y la tasa de transferencia se ve beneficiada.

Las figuras 5.1 y 5.2 contienen la misma información que se encuentra en las tablas anteriores pero representada en forma de gráfico de barras. Las barras azules representan la tasa de transferencia obtenida por la versión original de la implementación, mientras que las barras rojas representan el incremento alcanzado por la implementación que contiene las modificaciones propuestas.

La Figura 5.1 contiene las mediciones para los esquemas BPSK 1/2, QPSK 1/2, BPSK 3/4 y QPSK 3/4, mientras que la Figura 5.2 contiene las mediciones de 16QAM 1/2, 16QAM 3/4, 64QAM 2/3 y 64QAM 3/4. Para cada esquema, la barra de más a la izquierda se corresponde con la medición con PDUs de 500 bytes, la barra del medio con PDUs de 1000 bytes y la de la derecha con PDUs de 1500 bytes

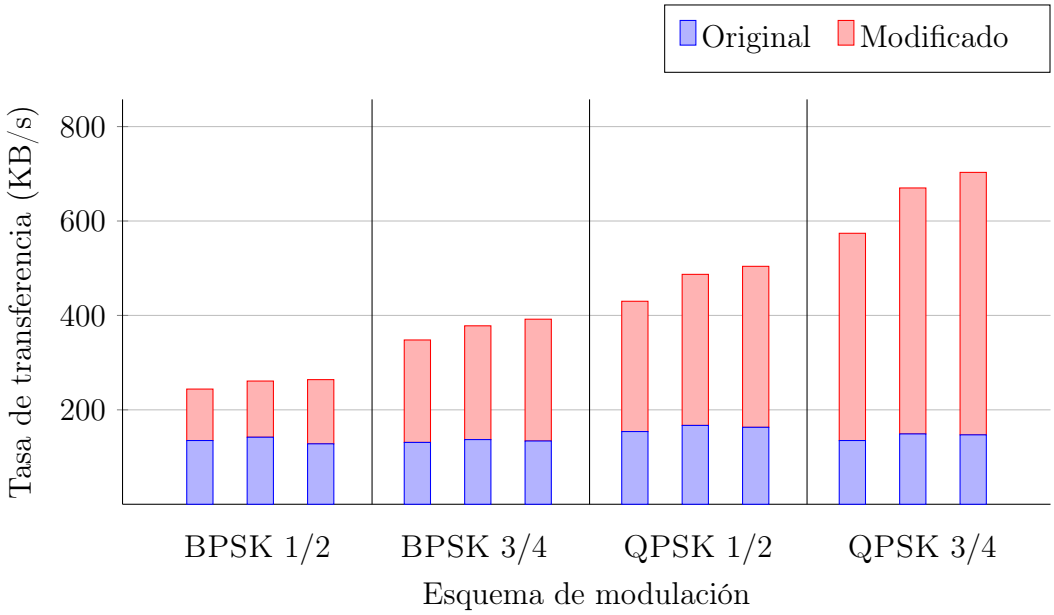

Figura 5.1: Tasas de transferencia promedio de la implementación original (en azul) y de la modificada (rojo más azul) para los esquemas de modulación BPSK 1/2, BPSK 3/4, QPSK 1/2 y QPSK 3/4. Para cada esquema de modulación, se encuentran las mediciones con PDUs de 500, 1000 y 1500 bytes respectivamente (de izquierda a derecha).

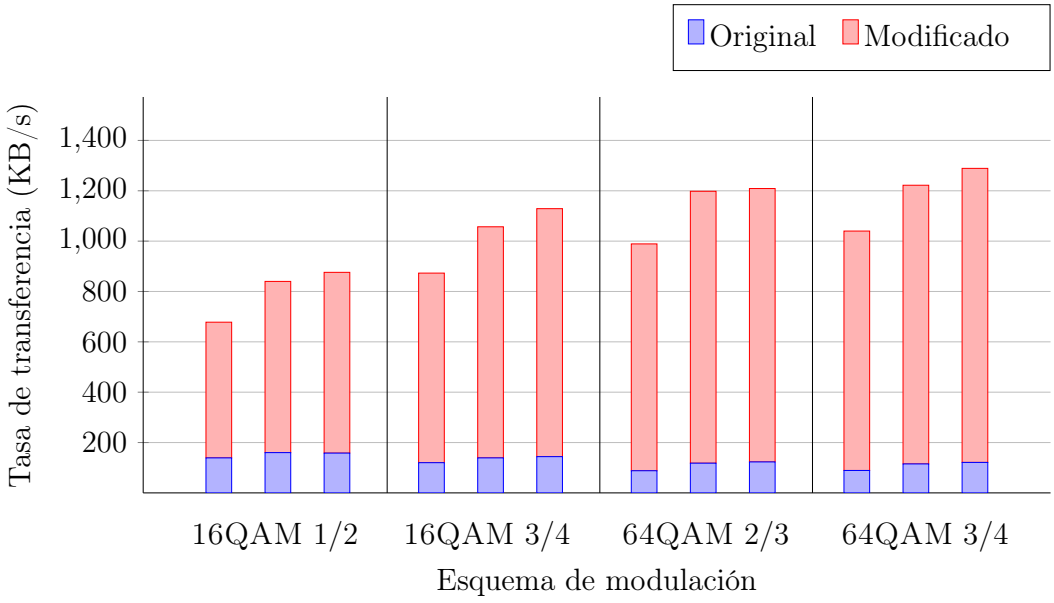

Figura 5.2: Tasas de transferencia promedio de la implementación original (azul) y de la modificada (rojo más azul) para los esquemas de modulación 16QAM 1/2, 16QAM 3/4, 64QAM 2/3 y 64QAM 3/4. Para cada esquema de modulación, se encuentran las mediciones con PDUs de 500, 1000 y 1500 bytes respectivamente (de izquierda a derecha).

Para finalizar con las mediciones realizadas en software, se presenta una comparación del resultado de la ejecución de la herramienta GNU Radio Performance Monitor tanto sobre la implementación original como sobre la modificada. La Tabla 5.8 contiene el porcentaje del tiempo de ejecución total que representan los bloques que habían sido identificados como los más demandantes, OFDM Decode MAC y OFDM Carrier Allocator, para todos los esquemas de modulación disponibles, con un PDU de 50 bytes. Por su parte, la Tabla 5.9 contiene los mismos datos pero al utilizarse PDUs de 1500 bytes.

La primera observación que puede hacerse es que en todos los casos, las modificaciones propuestas logran reducir el porcentaje de ejecución de los bloques identificados. Para PDUs de 50 bytes, la reducción más grande se encuentra en el bloque OFDM Carrier Allocator, lo que coincide con el comportamiento esperado y descrito detalladamente en la Sección 5.1.1. Por otra parte, para PDUs de 1500 bytes se producen reducciones significativas tanto en el bloque OFDM Decode MAC como en OFDM Carrier Allocator.

	Versión original		***Versión modificada***	
Modulación	OFDM Decode MAC (%)	OFDM Carrier Allocator (%)	OFDM Decode MAC (%)	OFDM Carrier Allocator (%)
BPSK 1/2	7,9	78,4	7,0	1,7
BPSK 3/4	8,8	78,1	9,2	1,5
QPSK 1/2	7,8	80,2	9,4	1,6
QPSK 3/4	8,5	79,9	13,4	1,45
16QAM 1/2	8,4	80,2	12,9	1,5
16QAM 3/4	16,5	72,8	17,0	1,4
64QAM 2/3	19,9	70,0	19,0	1,4
64QAM 3/4	17,3	72,3	17,0	1,2

Tabla 5.8: Porcentaje de tiempo de ejecución de los bloques OFDM Decode MAC y OFDM Carrier Allocator para todos los esquemas de modulación disponibles, con un PDU de tamaño 50.

En segundo lugar, es importante destacar que para el caso de PDUs de 1500 bytes, la reducción en el bloque OFDM Decode MAC es menor para los esquemas de modulación más complejos (16QAM y 64QAM) que para los más simples (BPSK y QPSK). Esto se debe en gran medida al costo de demodular dichos esquemas, que es mayor si el esquema es más complejo. En esos casos, tanto la demodulación como la decodificación del código convolucional siguen representando un porcentaje importante del tiempo de ejecución, incluso con las mejoras aplicadas.

Por último, es importante aclarar que debido a la disminución del costo de los bloques OFDM Decode Mac y OFDM Carrier Allocator en la versión mejorada, otros bloques que antes tenían un consumo marginal pasaron a tener un porcentaje de tiempo de ejecución a la par de los dos primeros, y en muchos casos los superaron. En particular,

Modulación	*Versión original*		*Versión modificada*	
	OFDM Decode MAC (%)	OFDM Carrier Allocator (%)	OFDM Decode MAC (%)	OFDM Carrier Allocator (%)
BPSK 1/2	41,6	27,7	5,0	8,5
BPSK 3/4	42,4	26,3	9,8	6,1
QPSK 1/2	57,5	24,3	13,1	5,0
QPSK 3/4	57,1	25,1	26,0	3,4
16QAM 1/2	69,3	21,5	33,2	2,5
16QAM 3/4	70,1	21,2	27,3	3,2
64QAM 2/3	76,1	16,6	49,1	1,2
64QAM 3/4	77,7	15,1	52,2	1,0

Tabla 5.9: Porcentaje de tiempo de ejecución de los bloques OFDM Decode MAC y OFDM Carrier Allocator para todos los esquemas de modulación disponibles, con un PDU de tamaño 1500.

se destacan los bloques OFDM Mapper y OFDM Sync Short, que pasaron a consumir en promedio el 10 % y 12 % respectivamente en el caso de utilizar un PDU de 1500 bytes, superando a los bloques OFDM Decode Mac y OFDM Carrier Allocator en caso de utilizar esquemas de modulación simples (BPSK y QPSK).

Al trabajar con un PDU de tamaño chico como 50 bytes, los bloques que más se destacan luego de aplicadas las mejoras son el OFDM Sync Short y el OFDM Sync Long, ambos manteniéndose arriba del 10 % para todas las modulaciones y tipos de codificación disponibles. El OFDM Mapper se mantiene con un promedio de 5 %, el cual es superior en todos los casos al costo del OFDM Carrier Allocator.

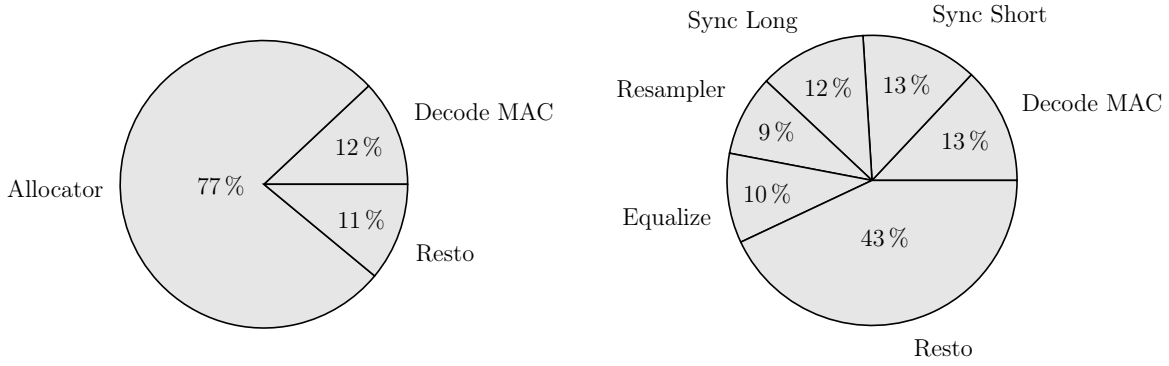

Figura 5.3: Porcentaje de tiempo de ejecución según bloque para un PDU de 50 bytes, versión original (izquierda) y modificada (derecha).

En las Figuras 5.3 y 5.4 se representa de forma gráfica la información presentada en las tablas 5.8 y 5.9 respectivamente. El porcentaje asignado a cada bloque es el promedio de todos los porcentajes registrados para ese bloque para los distintos esquemas de modulación y tipos de codificación, de forma de mostrar un único valor representativo

del consumo general del mismo.

En ambas figuras puede apreciarse con facilidad que, en el caso de la implementación original, los dos bloques más costosos representan un porcentaje muy alto del total de tiempo de ejecución del sistema. Por el contrario, la versión mejorada presenta un mayor balance en el costo de procesamiento de cada bloque, habiendo varios bloques por encima del 10 %, y no existiendo ninguno en particular por encima del 30 % del tiempo total de ejecución.

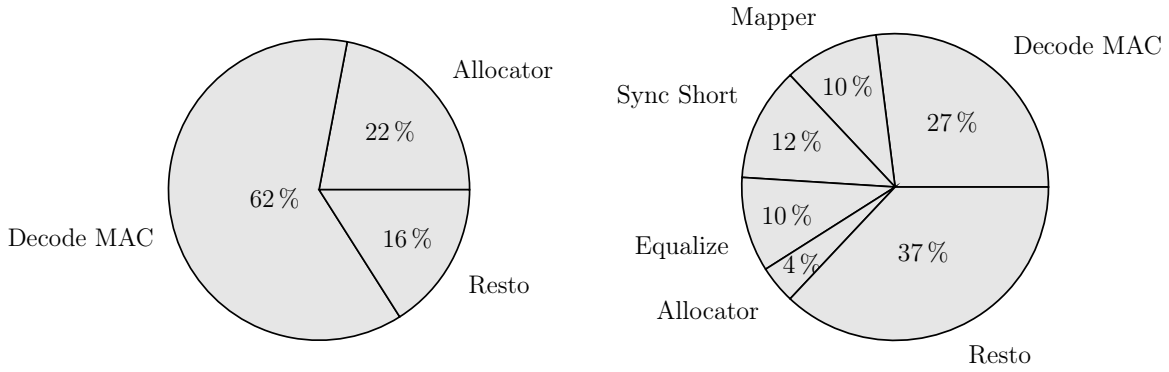

Figura 5.4: Porcentaje de tiempo de ejecución según bloque para un PDU de 1500 bytes, versión original (izquierda) y modificada (derecha).

5.2. Pruebas en hardware

Esta sección describe las *pruebas de concepto* realizadas sobre hardware físico con el objetivo de determinar el desempeño que podría alcanzar la implementación realizada al utilizarse hardware de SDR. Estas pruebas no pretenden ser de ninguna manera exhaustivas, sino tan solo mostrar cómo se trasladan al hardware las mejoras obtenidas sobre el software.

Todas las pruebas fueron realizadas utilizando los grafos del receptor y transmisor presentados en las secciones 4.2.1 y 4.2.2 respectivamente, modificando el tamaño del PDU, los esquemas de modulación y las tasas de codificación utilizadas.

Las pruebas fueron realizadas utilizando una computadora de escritorio con un procesador Intel Core i3 5400 con 8 GB de memoria RAM DDR3 y operando en un sistema operativo Ubuntu 14.04.3 LTS, una máquina con hardware distinto al utilizado en la Sección 5.1[2]. Se realizaron pruebas exclusivas sobre el transmisor y receptor *gr-ieee802-11*, interoperando con una computadora portátil convencional. También se realizaron pruebas en donde tanto el transmisor como el receptor utilizados contenían

[2] El cambio de equipo fue necesario debido a que la máquina utilizada para pruebas en software no contaba con puertos USB 3.0, que eran requeridos por los USRP B210.

la implementación *gr-ieee802-11*, operando sobre una misma máquina, en un escenario de loopback.

En todos los casos, el USRP utilizado para la transmisión y recepción de la señal en el medio fue un Ettus USRP B210. La versión de GNU Radio utilizada fue la 3.7.8. Las mediciones fueron tomadas modificando el código fuente del bloque OFDM Decode MAC para contar la cantidad de paquetes recibidos correctamente al terminar la ejecución del receptor. En el caso en que se utilizó la antena como transmisora, se contó la cantidad de paquetes recibidos correctamente en la computadora portátil mediante la utilidad tcpdump.

5.2.1. Pruebas sobre el transmisor

La primera serie de pruebas fue realizada sobre una antena configurada para utilizar el grafo de un transmisor, con la capacidad de enviar paquetes de tamaño configurable a una tasa también configurable. Se utilizó una computadora portátil configurada en modo monitor para poder capturar los paquetes enviados por el transmisor sin necesidad de asociarse con el mismo mediante Wi-Fi.

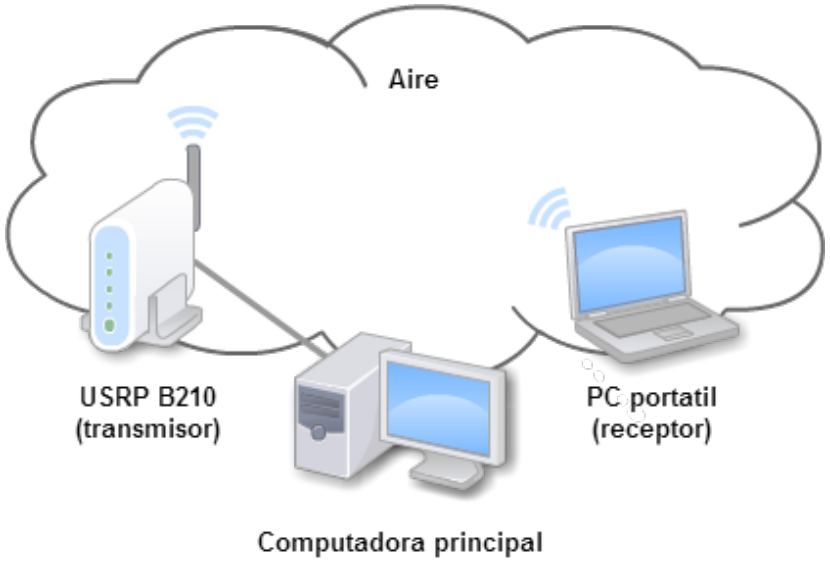

Figura 5.5: Esquema de la prueba realizada sobre el transmisor de *gr-ieee802-11*. En el esquema se puede apreciar que la computadora de escritorio se encuentra conectada a un USRP B210 que oficia de transmisor en la conexión, inyectando los paquetes recibidos desde el equipo en el aire. El receptor es simplemente una computadora portátil convencional configurada en modo monitor para capturar paquetes que se encuentren en el aire.

Para realizar las mediciones, se utilizaron los esquemas de modulación BPSK con tasa de codificación de 1/2 y 64QAM con tasa 3/4. Se eligieron dos tamaños de paquete, 50 bytes y 1500 bytes, midiéndose la cantidad de paquetes recibidos correctamente por

la computadora portátil en un tiempo fijo, y en base a esta información se calculó la tasa de transferencia promedio.

Modulación	Tamaño	Ideal (KB/s)	Original (KB/s)	% sobre ideal	Mod. (KB/s)	% sobre ideal	Acel.
BPSK 1/2	50	48,83	8,21	16,81	45,86	93,92	5,59
BPSK 1/2	50	341,80	7,98	2,33	246,36	72.08	30,89
BPSK 1/2	50	732,42	8,13	1,11	240,80	32,88	29,62
BPSK 1/2	1500	14,65	14,52	99,12	14,52	99,12	1,00
BPSK 1/2	1500	146,48	144,43	98,60	143,40	97,89	0,99
BPSK 1/2	1500	732,42	199,17	27,19	671,52	91,68	3,37
64QAM 3/4	50	48,83	8,42	17,24	45,55	93,28	5,41
64QAM 3/4	50	976,56	8,17	0,84	355,76	36,43	43,56
64QAM 3/4	50	6835,94	8,07	0,12	322,15	4,71	39,94
64QAM 3/4	1500	146,48	145,33	99,21	144,94	98,95	1,00
64QAM 3/4	1500	1464,84	237,97	16,25	1373,10	93,74	5,77
64QAM 3/4	1500	6884,77	221,40	3,22	4349,81	63,18	19,65

Tabla 5.10: Resultados obtenidos al realizar las pruebas sobre el transmisor. Las columnas contienen, de izquierda a derecha: el esquema de modulación utilizado, el tamaño del paquete enviado, la tasa de transferencia a la cual se inyectaron paquetes en el transmisor, la tasa obtenida por la implementación original, el porcentaje de esta tasa sobre la ideal, la tasa de transferencia obtenida por la implementación modificada, el porcentaje de esta tasa sobre la ideal, y la aceleración obtenida por la implementación modificada sobre la original.

La Tabla 5.10 presenta los resultados obtenidos. La columna *Ideal* contiene la tasa a la cual se inyectaron los paquetes en el canal. Para cada esquema de modulación y tamaño de paquete elegido, se eligieron tres tasas ideales. La más grande es la tasa máxima soportada por dicho esquema[3], mientras que las otras dos son valores más pequeños elegidos para probar el comportamiento del transmisor ante un canal no saturado.

Las columnas *Original* y *Mod.* contienen las tasas de transferencia obtenidas por la implementación original y la modificada respectivamente, mientras que las columnas *% sobre ideal* contienen el porcentaje que representa la tasa de la implementación correspondiente sobre la tasa ideal. Por último, la columna *Acel.* contiene la aceleración del transmisor modificado frente al original.

En base a los resultados obtenidos, resulta claro que la implementación propuesta obtiene mejoras sobre el transmisor en casi todos los casos, especialmente cuando la tasa ideal se encuentra en valores cercanos al límite para la modulación actual. Esto se debe a que al acercarse a la tasa ideal, la implementación original es incapaz de mantener el ritmo impuesto por la tasa de paquetes inyectados, algo que la implementación modificada es capaz de hacer hasta cierto punto. Además de esto, si se observan los

[3] Esto es, 6 Mbps (768 KB/s) para BPSK 1/2 y 54 Mbps (6912 KB/s) para 64QAM 3/4.

porcentajes obtenidos sobre la tasa ideal por la implementación mejorada, se observan tasas de transferencia muy cercanas al valor requerido por el estándar, especialmente en paquetes de 1500 bytes. Esto sugiere que, en dichos casos, el desempeño del transmisor es muy cercano al de un transmisor real funcionando en hardware.

Por otra parte, la aceleración es mayor cuando el tamaño de paquete es de 50 bytes. Este comportamiento coincide con el que se detalló en la Sección 5.1.1.

5.2.2. Pruebas sobre el receptor

En segundo lugar, se realizaron pruebas sobre una antena configurada para utilizar el grafo del receptor. Se utilizó una computadora portátil configurada en modo monitor, pero esta vez tomando el rol del transmisor. Para realizar el envío de paquetes, se hizo uso del programa *packetspammer* incluido en *gr-ieee-802-11*. Este programa fue configurado para enviar paquetes de tamaño configurable a una tasa configurable, y según el esquema de modulación y tasa de codificación deseados.

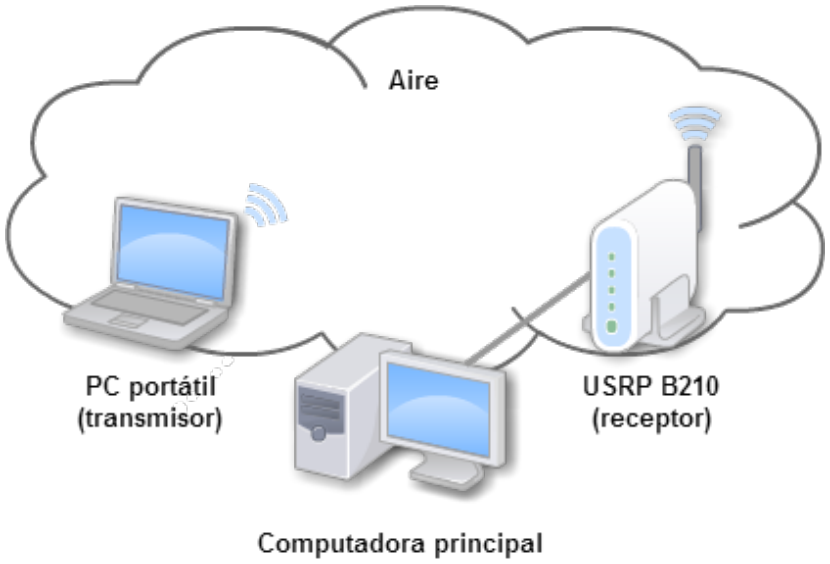

Figura 5.6: Esquema de la prueba realizada sobre el receptor de *gr-ieee802-11*. En el esquema se puede apreciar que la computadora de escritorio se encuentra conectada a un USRP B210 que oficia de receptor en la conexión, capturando todos los paquetes que se detectan en el aire. El transmisor es simplemente una computadora portátil convencional configurada para enviar paquetes de tamaño fijo a una tasa fija, a través de un programa simple que maneja *sockets*.

Las mediciones se realizaron utilizando nuevamente los esquemas de modulación BPSK 1/2 y 64QAM 3/4, con paquetes de tamaño 50 y 1500 bytes y midiéndose la cantidad de paquetes efectivamente recibidos por el receptor en un tiempo fijo. Esta información se utilizó para calcular la tasa de transferencia promedio.

Modulación	Tamaño	Ideal (KB/s)	Original (KB/s)	% sobre ideal	Mod. (KB/s)	% sobre ideal	Acel.
BPSK 1/2	50	48,83	9,06	18,55	9,67	19,80	1,07
BPSK 1/2	50	341,80	34,29	10,03	37,19	10,88	1,08
BPSK 1/2	50	732,42	36,89	5,04	42,53	5,81	1,15
BPSK 1/2	1500	14,65	8,87	60,53	9,25	63,16	1,04
BPSK 1/2	1500	146,48	50,37	34,39	54,74	37,37	1,09
BPSK 1/2	1500	732,42	54,10	7,39	75,17	10,26	1,39
64QAM 3/4	50	48,83	9,09	18,62	9,57	19,60	1,05
64QAM 3/4	50	976,56	40,66	4,16	59,08	6,05	1,45
64QAM 3/4	50	6835,94	40,19	0,59	57,77	0,85	1,44
64QAM 3/4	1500	146,48	56,15	38,33	61,04	41,67	1,09
64QAM 3/4	1500	1464,84	79,54	5,43	228,85	15,62	2,88
64QAM 3/4	1500	6884,77	89,69	1,30	527,09	7,66	5,88

Tabla 5.11: Resultados obtenidos al realizar las pruebas sobre el receptor. Las columnas contienen la misma información que en la Tabla 5.10, teniendo en cuenta que la tasa ideal es la tasa efectiva enviada por el transmisor.

La Tabla 5.11 presenta los resultados obtenidos. Es importante observar que las aceleraciones obtenidas son mucho menores que las obtenidas en la sección anterior. Sin embargo, se mantiene el resultado de que cuanto más cerca de la tasa ideal, mejor se comporta el receptor modificado frente a la versión original. Esto es especialmente cierto en el caso de 64QAM 3/4 para paquetes de tamaño 1500 bytes, en donde se obtiene una aceleración pico de 5,88 cuando el transmisor opera a una tasa ideal de $6884,77$ KB/s (correspondiente a 54 Mbps). El receptor modificado es capaz de decodificar los paquetes en mucho menos tiempo, lo que resulta en una tasa de transferencia final mayor a la original.

Además, cuanto más complejo es el esquema de modulación y mayor el tamaño del paquete, más verdadera resulta la afirmación anterior, puesto que el receptor debe realizar mucho más trabajo para decodificar un paquete.

5.2.3. Pruebas con transmisor y receptor

Para realizar esta serie de pruebas, se utilizó una única máquina[4] con las especificaciones detalladas al comienzo de la sección. Se utilizaron dos USRP B210 conectados a dicha máquina para oficiar de transmisor y receptor de la señal. Existe una cierta similitud entre esta prueba y las realizadas en la Sección 5.1.3 debido a que ambos

[4] No se utilizaron dos máquinas independientes para el transmisor y receptor debido a que los USRP B210 requieren de una conexión USB 3.0 y solo se contaba con un equipo que soportara esta característica.

son escenarios de *loopback*[5], con la única diferencia de que en la primera el canal de comunicación es el aire y en la segunda es una simulación realizada en software.

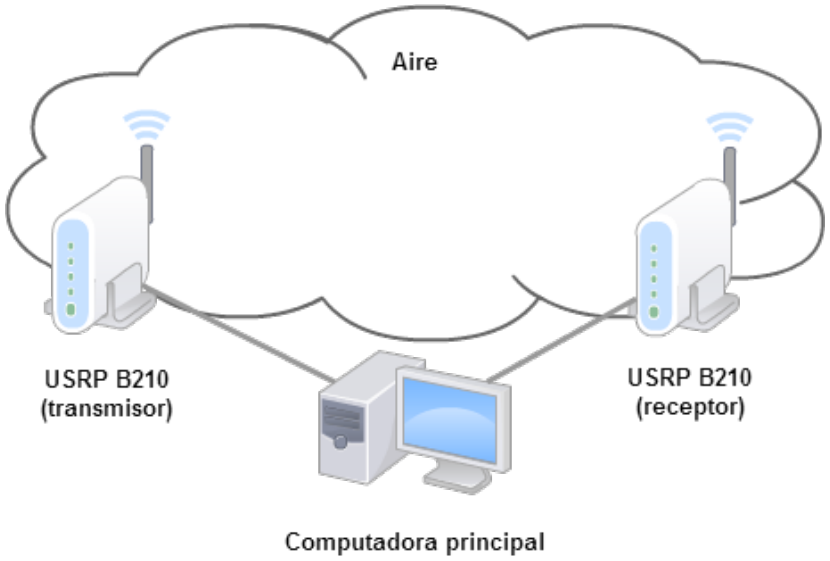

Figura 5.7: Esquema de la prueba realizada con transmisor y receptor de *gr-ieee802-11* operando de forma simultánea en una misma máquina. En el esquema se puede apreciar que la computadora de escritorio se encuentra conectada a dos USRP B210, uno que oficia de transmisor y otro de receptor. La computadora principal ejecuta los grafos de flujo de GNU Radio para el transmisor y receptor y se comunica con cada USRP a través de un puerto USB 3.0 independiente. El transmisor inyecta los paquetes en el aire y el receptor decodifica los paquetes detectados en el medio.

Debido a que el canal de comunicación es el aire, que es un canal imperfecto, y a que se está transmitiendo sin la utilización de ningún protocolo que asegure la recepción de paquetes en orden y sin pérdidas, no es razonable transferir un archivo y evaluar la tasa de recepción como se hizo en la sección de pruebas de loopback en software. Por esta razón, se configuró el transmisor para enviar paquetes a una tasa fija durante un tiempo determinado, y se contó la cantidad de paquetes que fueron recibidos y decodificados de forma correcta por el receptor.

En la Tabla 5.12 se muestran los datos obtenidos a partir de las pruebas realizadas tanto sobre la implementación original como sobre la mejorada, para los distintos tamaños de paquetes, tasas de transmisión y esquemas de modulación probados. La columna *Tasa ideal* representa la tasa a la cual el transmisor intenta inyectar paquetes en el medio. Esta tasa es una opción de configuración del programa, y no necesariamente es alcanzada por el transmisor, debido a la posible incapacidad del mismo para alcanzarla. A modo de ejemplo, para lograr una tasa ideal de 48,83 KB/s con un tamaño de paquete de 50 bytes, se debe configurar el transmisor para que inyecte 1000 paquetes por segundo en el medio.

[5] Se denomina *loopback* cuando el equipo encargado de recibir y decodificar es el mismo que transmitió dicha señal.

Modulación	Tamaño	Ideal (KB/s)	Original (KB/s)	% sobre ideal	Mod. (KB/s)	% sobre ideal	Acel.
BPSK 1/2	50	48,83	0,84	1,71	6,00	12,28	7,18
BPSK 1/2	50	341,80	0,69	0,20	9,10	2,66	13,20
BPSK 1/2	50	732,42	0,73	0,10	9,77	1,33	13,35
BPSK 1/2	1500	14,65	2,57	17,54	3,98	27,19	1,55
BPSK 1/2	1500	146,48	9,77	6,67	21,07	14,39	2,16
BPSK 1/2	1500	732,42	10,92	1,49	24,67	3,37	2,26
64QAM 3/4	50	48,83	0,85	1,75	6,34	12,98	7,44
64QAM 3/4	50	976,56	0,85	0,09	11,69	1,20	13,79
64QAM 3/4	50	6835,94	1,12	0,02	12,03	0,18	10,76
64QAM 3/4	1500	146,48	23,13	15,79	50,63	34,56	2,19
64QAM 3/4	1500	1464,84	19,27	1,32	122,58	8,37	6,36
64QAM 3/4	1500	6884,77	17,99	0,26	161,13	2,34	8,96

Tabla 5.12: Resultados obtenidos al realizar las mediciones de las pruebas de loopback en el aire. Las columnas contienen la misma información que en la Tabla 5.10.

Las columnas *Original* y *Mod.* contienen las tasas de transferencia obtenidas por la implementación original y la modificada respectivamente, mientras que las columnas *% sobre ideal* contienen el porcentaje que representa la tasa de la implementación correspondiente sobre la tasa ideal. Por último, la columna *Acel.* contiene la aceleración del transmisor modificado frente al original.

La primera observación que puede hacerse respecto de las tasas de transferencia es que son mucho menores a las tasas obtenidas en las pruebas en software, tanto para la implementación original como para la mejorada. Esto tiene sentido debido a que, como se explicó anteriormente, al estar transmitiendo en el aire existe una gran cantidad de paquetes que se pierden, mientras que las pruebas de software, que simulaban un canal perfecto, no tenían pérdidas.

En segundo lugar, se puede apreciar que la aceleración más grande es alcanzada en paquetes de tamaño 50 bytes, y con tasas ideales altas, es decir, con un alto número de paquetes por segundo inyectados en el medio. Esto es razonable, y es coherente con todas las observaciones realizadas hasta el momento, tanto en la etapa de pruebas como en la de profiling. Como se vio en secciones anteriores, es en los paquetes de tamaño pequeño en donde más costo tiene el bloque OFDM Carrier Allocator, y consecuentemente, donde mayor es la mejora lograda al realizar modificaciones sobre el mismo. Por esta razón, es lógico que paquetes de tamaño pequeño muestren una mejora más grande en la tasa de transferencia que paquetes de mayor tamaño, debido a que los primeros eran los más afectados por la ineficiencia del OFDM Carrier Allocator.

De la misma manera, también es razonable que la aceleración sea más grande a medida que se transmiten más paquetes por segundo, debido a que luego de las mejoras realizadas, el receptor es significativamente más veloz para realizar la decodificación de

la tramas, lo que permite que para una tasa de transmisión alta el receptor mejorado pueda realizar más decodificaciones que el de la implementación original.

Además, también es importante observar que la aceleración más grande es lograda al transmitir usando un esquema de modulación 64QAM con codificación 3/4, para una tasa ideal alta y un paquete pequeño. Es razonable que esta configuración presente la mayor aceleración debido a que en la misma se combina un paquete pequeño, lo cual explota de mejor manera la mejora realizada sobre el bloque OFDM Carrier Allocator, con un esquema de modulación complejo y tasa de codificación alta, lo cual explota de mejor manera las mejoras realizadas sobre el algoritmo de demodulación y decodificación convolucional dentro del bloque OFDM Decode MAC.

En cuanto a los datos de las columnas *% sobre ideal*, se puede concluir que la implementación propuesta mejora ampliamente el porcentaje de cumplimiento de la tasa a la cual se intentó transmitir, lo cual está relacionado con las columnas de aceleración y tasa de transferencia ya analizadas. A partir de las columnas de *% sobre ideal*, también es posible concluir que, si bien las mejoras logradas son muy significativas al comparar con la implementación original, las tasas alcanzadas por la versión mejorada no son lo suficientemente grandes como para poder competir con una implementación del estándar en hardware dedicado, al menos cuando se está ejecutando la SDR en una máquina similar a la utilizada en estas pruebas, cuyas características fueron especificadas en la Sección 5.1.

Capítulo 6

Conclusiones y trabajo futuro

El área de investigación de radios definidas por software ha estado en un constante crecimiento y evolución desde su incepción, hace más de veinte años, hasta la actualidad. Distintos factores hacen posible que hoy en día las radios definidas por software tengan la mayor aplicabilidad de toda su historia en distintos escenarios y sectores, como pueden ser el académico, el educativo y el comercial. A pesar de esto, y de todas las numerosas ventajas que poseen sobre las radios tradicionales, siguen teniendo algunos limitantes evidentes que no hacen posible que las mismas sustituyan completamente a las radios convencionales. Algunos de estos problemas fueron manifestados a lo largo del presente trabajo, especialmente en la Sección 5.2.3. Sin embargo, se considera que es un área muy interesante y que cuenta con mucho potencial para seguir creciendo.

En base al trabajo realizado, se considera que se alcanzaron los objetivos planteados al inicio del proyecto de forma satisfactoria. Se lograron mejoras importantes en el rendimiento de *gr-ieee802-11*, aumentando las tasas de transferencia alcanzadas por el programa de forma considerable.

En primer lugar, se realizó un estudio de las bases teóricas que permiten la existencia de los sistemas de comunicación inalámbrica. Este estudio tuvo una importancia mayor para que el proyecto lograra llevarse a cabo, debido a que los temas a tratar se encuentran principalmente en el área de estudio de la Ingeniería Eléctrica, y no en el de la Computación. El entendimiento de temas como OFDM y Códigos Convolucionales fue vital para el desarrollo posterior del proyecto.

En segundo lugar, se realizó un estudio del estado del arte tanto de SDR como del estándar Wi-Fi. Esto permitió, en conjunto con el punto mencionado en el párrafo anterior, adquirir las herramientas teóricas necesarias para entender una implementación de un transmisor y receptor Wi-Fi en GNU Radio.

En tercer lugar, se realizó el profiling necesario para determinar los puntos de la implementación cuyo desempeño podía ser mejorado. Se realizó un análisis exhaustivo de estos puntos, logrando encontrar posibles mejoras que más tarde fueron llevadas

a cabo. Por último, se realizó la comparación del desempeño alcanzado tanto por la implementación original como por la implementación con las mejoras propuestas, y se realizaron pruebas de concepto sobre hardware SDR real.

Se valora como muy positivo el haber adquirido conocimientos de comunicaciones inalámbricas, códigos convolucionales, OFDM, estándar Wi-Fi, Radios Definidas por Software y de las ventajas que estas brindan a la hora de entender cómo funcionan tanto las comunicaciones como Wi-Fi a bajo nivel, además de herramientas y librerías comúnmente utilizadas como lo son GNU Radio, IT++ y BLAS.

Es común que para lograr grandes mejoras en el rendimiento de un grafo de flujo de GNU Radio sea necesario unir varios bloques distintos en uno solo, de forma de reducir el costo producto de la división. Esto es algo que se intenta evitar debido a que atenta contra la modularidad del programa, ya que tiende a convertir a la SDR en una aplicación monolítica. Por esta razón, se destaca que las mejoras realizadas se lograron respetando el diseño original del programa; dicho de otra forma, manteniendo la misma modularidad que existía originalmente.

En cuanto a las tasas finales alcanzadas, son muy buenas tanto al comparar con las tasas logradas por la implementación original (las cuales eran entre dos y diez veces más lentas) como al evaluar la tasa neta alcanzada, ya que por ejemplo, en 64QAM se alcanza una transferencia de 10.8 Mbits por segundo en software.

A su vez, este rendimiento fue alcanzado en una computadora obsoleta para la tecnología disponible hoy en día. Si bien no se evaluó en un equipo con un procesador más potente (como por ejemplo, un Intel Core i7) o con una memoria RAM de mayor frecuencia, se puede estimar con bastante certeza que las mejoras realizadas lograrían tasas de transferencia aún mayores en un equipo con estas características.

Por otra parte, se destaca la excelente disposición que mostraron Bastian Bloessl y Bogdan Diaconescu al ser contactados por algunas consultas y dudas acerca de la implementación original de *gr-ieee802-11* y del decodificador de Viterbi de *gr-dvbt*, respectivamente. En el caso del primero, el autor se mostró muy interesado por el presente trabajo, solicitando la posibilidad de que una vez que el mismo estuviese finalizado, se le enviaran las modificaciones realizadas. Actualmente, el trabajo realizado se encuentra publicado en el sitio de GitHub, creado como un *fork* de la versión original, y es posible obtenerlo en [29]. Al momento de escribir este documento, las modificaciones realizadas han sido enviadas a Bastian Bloessl para su evaluación y verificación.

También se destaca la muy buena disposición que hubo por parte de la comunidad de GNU Radio para contestar dudas enviadas a la GNU Radio Mailing List, que fue principalmente utilizada cuando la documentación disponible era insuficiente o incompleta.

Para finalizar, existen muchas líneas de investigación como trabajo a futuro. Se podría evaluar el diseñar e implementar una versión alternativa que cumpla con el estándar y que utilice tanto el CPU como la GPU para realizar el procesamiento de la

señal. Sin embargo, parece muy difícil que esto logre una mejora si se sigue trabajando con la implementación de *gr-ieee802-11* y se intenta aplicar a bloques por separado, sin fusionar varios bloques en uno sólo primero.

También sería interesante estudiar posibles aplicaciones sobre los nuevos estándares de Wi-Fi, 802.11n y 802.11ac. Estos estándares permiten alcanzar tasas de transferencia mucho mayores, utilizando tecnologías como MIMO (Multiple-Input Multiple-Output), que hacen uso de múltiples antenas transmisoras y receptoras funcionando en paralelo para mejorar el desempeño del sistema. Los nuevos estándares también permiten realizar agrupación de tramas (*aggregation*), es decir, enviar hasta 64 KB de tramas sin necesidad de esperar por un ACK. Podría ser interesante utilizar una arquitectura con GPUs para realizar la decodificación de este gran número de tramas en paralelo.

Otra línea de investigación abierta sería evaluar la posibilidad de mover parte del procesamiento de la señal que hoy se hace en CPU a la FPGA del USRP. Esto aumentaría la disponibilidad del CPU para aquellos bloques que continuasen su ejecución en software en GNU Radio, mientras que para las tareas delegadas a la FPGA, se lograría obtener un rendimiento equivalente al de un hardware dedicado tradicional.

Bibliografía

[1] J. Mitola, *Software Radio Architecture*, IEEE Commun. Mag., vol. 33, no. 5, May 1995, pp. 26–38

[2] Daniel M. Dobkin, *RF Engineering for Wireless Networks: Hardware, Antennas and Propagation*, Newnes, Elsevier, London, 2005.

[3] Charan Langton, *Intuitive Guide to Principles of Communication: All About Modulation - Part I*, 2002.

[4] Hari Balakrishnan, Christopher Terman, y George Verghese, *Bits, Signals, and Packets: An Introduction to Digital Communications and Networks*, M.I.T. Department of EECS, 2012.

[5] B. Bloessl, M. Segata, C. Sommer y F. Dressler, *An IEEE 802.11 a/g/p OFDM Receiver for GNU Radio*, ACM SIGCOMM 2013, Segundo Taller del ACM SIGCOMM Software Radio Implementation Forum (SRIF 2013), páginas 9–16, Hong Kong, China, Agosto 2013. ACM.

[6] Prof. Brad Osgood, *Lecture Notes for EE 261: The Fourier Transform and its Applications*, Electrical Engineering Department, Standford University, 2012.

[7] A.J. Viterbi, *Error bounds for convolutional codes and an asymptotically optimum decoding algorithm*, IEEE Trans. Inform. Theory, volumen IT-13, páginas 260-269, Abril 1967.

[8] IEEE Computer Society, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE, 3 Park Avenue New York, March 2012.

[9] A. M. Wyglinski, D. P. Orofino, M. N. Ettus y T. W. Rondeau, *Revolutionizing Software Defined Radio: Case Studies in Hardware, Software, and Education*, IEEE Communications Magazine, January 2016.

[10] G. Sklivanitis, A. Gannon, S. N. Batalama y D. A. Pados, *Addressing Next-Generation Wireless Challenges with Commercial Software-Defined Radio Platforms*, IEEE Communications Magazine, January 2016.

[11] Matthew E. Gast. *802.11 Wireless Networks: The definitive guide*, O'Reilly

[12] *GNU Radio Site*, http://gnuradio.org/redmine/projects/gnuradio/wiki/WhatIsGR (Obtenido 06/07/2015)

[13] *VOLK — GNU Radio*, https://gnuradio.org/redmine/projects/gnuradio/wiki/Volk (Obtenido 02/12/2015)

[14] *Out-of-tree modules*, https://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules (Obtenido 06/07/2015)

[15] *Wikipedia: Software-defined radio*, https://en.wikipedia.org/wiki/Software-defined_radio (Obtenido 06/07/2015)

[16] *Certified Wireless Analysis Professional Course, Chapter 8*, http://media.techtarget.com/searchMobileComputing/downloads/CWAP_ch8.pdf (Obtenido 28/01/2016)

[17] *Convolutional Decoders for Amateur Packet Radio Phil Karn, KA9Q*, http://www.ka9q.net/papers/cnc_coding.html (Obtenido 14/01/2016)

[18] *Fuente que realiza la decodificación convolucional en gr-dvbt*, https://github.com/BogdanDIA/gr-dvbt/blob/master/lib/d_viterbi.c (Obtenido 14/01/2016)

[19] L. Chia-Horng. On the design of OFDM signal detection algorithms for hardware implementation. In *IEEE GLOBECOM 2003*, pages 596–599, San Francisco, CA, December 2003. IEEE.

[20] E. Sourour, H. El-Ghoroury, and D. McNeill. Frequency Offset Estimation and Correction in the IEEE 802.11a WLAN. In *IEEE VTC2004-Fall*, pages 4923–4927, Los Angeles, CA, September 2004. IEEE.

[21] *Log for C++ Project*, http://log4cpp.sourceforge.net/ (Obtenido 21/09/2015)

[22] *Apache Thrift - Home*, https://thrift.apache.org/ (Obtenido 21/09/2015)

[23] *IT++ Documentation*, http://itpp.sourceforge.net/4.3.1/ (Obtenido 21/09/2015)

[24] *Bastian Bloessl, bastibl/gr-foo*, https://github.com/bastibl/gr-foo (Obtenido 23/05/2015)

[25] *Bastian Bloessl, bastibl/gr-ieee802-11*, https://github.com/bastibl/gr-ieee802-11/ (Obtenido 23/05/2015)

[26] *John Hunter, matplotlib: Python plotting*, (Obtenido 21/09/2015) `http://matplotlib.org/`

[27] *NetworkX, High-productivity software for complex networks*, `https://networkx.github.io/` (Obtenido 21/09/2015)

[28] *PyGraphviz, a Python interface to the Graphviz graph layout and visualization package*, `https://pygraphviz.github.io/` (Obtenido 21/09/2015)

[29] Gonzalo Arcos, Rodrigo Ferreri *Mejoras de performance en la implementación de un transceptor OFDM IEEE 802.11 /a/g/p*, `https://github.com/gonza1207/gr-ieee802-11` (Obtenido 02/04/2016)

Anexo A

Instalación de GNU Radio

Este anexo detalla el procedimiento llevado a cabo para realizar la instalación de GNU Radio y de todos los componentes que fueron utilizados para la realización del presente proyecto. Esto incluye, entre otros, la implementación del estándar 802.11 desarrollada por Bastian Bloessl en GNU Radio[1] que fue utilizada como punto de partida del proyecto, así como un conjunto de bibliotecas auxiliares necesarias. Además, fueron necesarias más configuraciones para activar la interfaz de *Control Ports*, que permite el uso del *GNU Radio Performance Monitor*, que fue vital para realizar la medición de la *performance* de las distintas implementaciones y cuya configuración no es trivial.

La instalación fue realizada en una computadora con sistema operativo Ubuntu 14.04[2]. Los pasos se detallan a continuación:

1. Instalar la biblioteca **log4cpp**[3]. Esta biblioteca contiene un conjunto de clases de C++ que se pueden utilizar para realizar *logging* flexible a distintas ubicaciones:

```
sudo apt-get install liblog4cpp5-dev
```

2. Instalar los prerrequisitos de GNU Radio. Esto puede hacerse mediante el uso del script **build-gnuradio**, que puede encontrarse en Internet:

```
wget http://www.sbrac.org/files/build-gnuradio && chmod a+x
    ./build-gnuradio && ./build-gnuradio -m -v prereqs
```

[1] Para más información del proyecto, ver [25].
[2] Es necesario disponer de permisos de administrador para realizar el procedimiento.
[3] Para más información, ver [21].

3. Instalar **Apache Thrift**[4], un *framework* que permite desarrollar servicios entre distintos lenguajes de programación de manera escalable. Dicha utilidad es necesaria para el funcionamiento del **GNU Radio Performance Monitor**:

```
sudo apt-get install libssl-dev
sudo apt-get install byacc flex
git clone https://git-wip-us.apache.org/repos/asf/thrift.git thrift
cd thrift
./bootstrap.sh
./configure
make
sudo make install
export PYTHONPATH=$PYTHONPATH:/usr/lib/python2.7/site-packages
```

4. Realizar la instalación completa de GNU Radio. Para esto, se vuelve a utilizar el script **build-gnuradio**:

```
wget http://www.sbrac.org/files/build-gnuradio && chmod a+x
    ./build-gnuradio && ./build-gnuradio -m -v
```

5. Instalar la biblioteca **IT++**[5], que provee clases y funciones matemáticas, de procesamiento de señales y comunicaciones. Esta dependencia es utilizada por la implementación del estándar 802.11 en GNU Radio:

```
sudo apt-get install libitpp-dev
```

6. Instalar **gr-foo**, una colección de bloques utilizados por la implementación del estándar 802.11 en GNU Radio[6]:

```
git clone https://github.com/bastibl/gr-foo.git
cd gr-foo
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

7. Instalar **gr-ieee802-11**, la colección de bloques y los archivos creados por Bastian Bloessl que implementan el estándar 802.11 en GNU Radio[7]:

[4] Para más información, ver [22].
[5] Para más información, ver [23].
[6] Para más información, ver [24].
[7] Para más información, ver [25].

```
git clone git://github.com/bastibl/gr-ieee802-11.git
cd gr-ieee80211
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig
```

8. Ajustar el máximo de memoria compartida. Debido a que el transmisor utiliza bloques de tipo *Tagged Stream*, debe almacenar tramas completas en su *buffer* antes de poder procesarlas. La cantidad máxima de memoria compartida que viene configurada por defecto en la mayoría de los sistemas Linux podría no ser suficiente. Se puede incrementar mediante el siguiente comando:

```
sudo sysctl -w kernel.shmmax=2147483648
```

9. Editar el archivo *.bashrc*, añadiendo al *PATH* de Python la ubicación de los paquetes de GNU Radio instalados. Luego de realizar este paso, es necesario reiniciar la sesión actual para que surta efecto:

```
export PYTHONPATH=/usr/local/lib/python2.7/dist-packages:
/usr/lib/python2.7/site-packages
```

10. Abrir GNU Radio Companion, la utilidad que se utiliza para editar y visualizar los gráficos de flujo de GNU Radio. A continuación, localizar el archivo **wifi_phy_hier.grc** que se encuentra en la carpeta *examples*, dentro del proyecto gr-ieee802-11, y generarlo para que esté disponible como bloque.

11. Instalar Matplotlib[8], una biblioteca que permite producir figuras y gráficos 2D con calidad de publicación en una variedad de formatos y plataformas. Es utilizada por **GNU Radio Performance Monitor** para realizar todo tipo de gráficas:

```
sudo apt-get install python-matplotlib
```

12. Instalar NetworkX[9], un paquete de software de Python utilizado para la creación, manipulación y estudio de la estructura, dinámismo y funciones de redes complejas. Nuevamente, **GNU Radio Performance Monitor** depende de esta utilidad:

[8] Para más información, ver [26].
[9] Para más información, ver [27].

```
sudo apt-get install python-networkx
```

13. Instalar PyGraphviz[10], una interfaz Python para el paquete Graphviz, utilizado para visualizar gráficos. Nuevamente, una dependencia de **GNU Radio Performance Monitor**:

```
sudo apt-get install python-pygraphviz
```

[10] Para más información, ver [28].

Anexo B

Creación de un bloque en GNU Radio

La referencia [14] posee un enlace al tutorial oficial de GNU Radio, que describe con detalle cómo crear módulos y bloques personalizables desde cero, por lo que no se replica esa información en este anexo. Sin embargo, aquí se explican los pasos que fueron necesarios para crear el bloque OFDM Carrier Allocator IEEE 802.11 dentro del módulo IEEE 802.11. Los pasos seguidos no fueron exactamente los que se describen en el tutorial oficial debido a que el módulo al cual el nuevo bloque debía pertenecer ya existía, y la implementación del bloque y su estructura también ya existía (dentro de la distribución oficial de GNU Radio).

Las rutas de archivos que se indican a continuación son relativas a la carpeta que contiene los fuentes de GNU Radio, o a la que contiene los fuentes de *gr-ieee802-11* según corresponda. De esta forma, los pasos seguidos fueron:

1. Copiar los fuentes de la implementación original del bloque OFDM Carrier Allocator desde la distribución oficial de GNU Radio hacia la carpeta de instalación del proyecto *gr-ieee802-11*.

   Para esto, se deben copiar los archivos con nombre *ofdm_carrier_allocator_impl* y con extensión *.cc* y *.h*, que se encuentran dentro de la carpeta de instalación de GNU Radio en la ruta **gnuradio/gr-digital/lib**, hacia la carpeta lib de la instalación de *gr-ieee802-11*. Además, es necesario copiar el fuente *ofdm_carrier_allocator_cvc.h* que se encuentra en **gnuradio/gr-digital/lib** hacia la carpeta **include/ieee802-11** dentro de la carpeta de instalación de *gr-ieee802-11*.

   También es necesario copiar el fuente *digital_ofdm_carrier_allocator_cvc.xml* ubicado en la carpeta **gnuradio/gr-digital/lib** a la carpeta **grc/** de la instalación de *gr-ieee802-11*. Este archivo es el que se encarga de definir la visualización y propiedades del bloque que luego son mostradas en la herramienta GNU Radio Companion.

2. Modificar los fuentes para cambiar el nombre del nuevo bloque de forma que no conflictue con el ya existente OFDM Carrier Allocator, y que pertenezca al módulo IEEE 802.11. Para esto, es necesario realizar cambios en el nombre de los fuentes, el nombre de la clase que define el bloque, el namespace de la clase y directivas de preprocesamiento, entre otros.

3. Modificar los archivos *CMakeLists.txt* que se encuentran dentro de las carpetas *include/ieee802-11* y *lib*, siendo estas rutas relativas respecto a a la carpeta de instalación de *gr-ieee802-11*. Esto es necesario para que la herramienta *CMake* compile los fuentes agregados en el primer paso.

4. Modificar el archivo *ieee802_11_swig.i* que se encuentra en la carpeta **swig** dentro de la instalación de *gr-ieee802-11*. Este archivo es el que utiliza swig como referencia para definir los wrappers de Python sobre las clases C++ que representan a cada bloque. Por lo tanto, es necesario definir aquí el bloque nuevo para que después pueda ser invocado desde los scripts Python generados por GNU Radio Companion y ejecutados por GNU Radio.

Anexo C

Glosario

A-MSDU (Aggregate MSDU), MSDU Agregado: Estructura que contiene múltiples MSDUs transportados en una única trama de capa MAC.

AP (Access Point), Punto de Acceso: Entidad que contiene una estación y provee acceso a los servicios de distribución para las estaciones asociadas.

BSS (Basic Service Set), Conjunto Básico de Servicio: Conjunto de estaciones que se han sincronizado de manera exitosa utilizando las primitivas correspondientes definidas en el estándar IEEE 802.11 y que están identificadas por un BSSID.

CFP (Contention-Free Period), Período Libre de Contención: Período de tiempo durante el cual el coordinador (PC) permite que los intercambios de tramas ocurran entre miembros del BSS sin que haya contención por el medio inalámbrico.

CRC (Cyclic Redundancy Check), Comprobación de Redundancia Cíclica: Código de detección de errores utilizado para detectar cambios accidentales en un conjunto de datos.

Decimador: Componente capaz de reducir la tasa de muestreo de una señal.

DA (Destination Address), Dirección de Destino: Dirección MAC individual o de grupo que identifica a las entidades MAC que deberían ser los receptores finales de la información transmitida en el cuerpo de la trama.

DS (Distribution System), Sistema de Distribución: Sistema utilizado para interconectar BSSs y redes de área local (LANs).

FCS (Frame Check Sequence), Secuencia de Comprobación de Trama: Código de detección de errores añadido a una trama en un protocolo de comunicaciones.

FEC (Forward Error Correction), Corrección de Errores Hacia Adelante: Técnica utilizada para controlar errores en la transmisión de datos sobre canales de comunicación no confiables o ruidosos. La idea central es que el transmisor envíe el mensaje en conjunto con información de redundancia utilizando un código de corrección de errores.

GPP (General Purpose Processor), Procesador de Propósito General: Microprocesador optimizado para su uso en un rango amplio de aplicaciones.

ICI (Inter Carrier Interference), Interferencia entre Carriers: Interferencia entre carriers o subcarriers vecinos de una señal transmitida.

IEEE 802.11: Conjunto de especificaciones de control de acceso al medio (MAC) y de capa física (PHY) para implementar redes de área local inalámbricas (WLAN). Dichas especificaciones, que constituyen un estándar, son creadas y mantenidas por el Instituto de Ingeniería Eléctrica y Electrónica (IEEE).

MAC (Medium Access Control), Control de Acceso al Medio: Subcapa inferior de la capa de enlace de datos en el modelo OSI que provee los mecanismos de direccionamiento y control de acceso al canal que hacen posible que múltiples nodos de una red puedan comunicarse en un medio compartido.

MPDU (MAC Protocol Data Unit), Unidad de Datos del Protocolo MAC: Unidad de datos intercambiada entre dos entidades MAC usando los servicios de la capa física. MPDU es el sinónimo de "trama" que utiliza el estándar Wi-Fi.

MMPDU (MAC Management Protocol Data Unit), PDU de Gestión de MAC: Unidad de datos intercambiada entre dos entidades MAC, utilizando servicios de la capa física, para implementar el protocolo de gestión de MAC.

Modelo OSI: Modelo conceptual que estandariza la estructura y funcionalidades de una red de computadoras con el objetivo de permitir la interoperabilidad de diversos sistemas con protocolos estándares.

MSDU (MAC Service Data Unit), Unidad de Datos del Servicio MAC: Información transmitida como una unidad entre puntos de acceso.

OFDM (Orthogonal Frequency Division Multiplexing), Multiplexación por División de Frecuencias Ortogonales: Tipo especializado de modulación que codifica la información en múltiples subcarriers, cada uno de ellos en frecuencias distintas.

Paquete: Unidad de datos con un formato definido intercambiada en una red de comunicaciones con el objetivo de transmitir algún tipo de información entre uno o más nodos de la red.

PC (Point Coordinator), Punto Coordinador: Entidad perteneciente a la estación (STA) de un punto de acceso (AP) que coordina la comunicación dentro de la red.

PDU (Protocol Data Unit), Unidad de Datos del Protocolo: Unidad de datos que es especificada por un protocolo de una capa determinada y puede contener tanto información de control propia del protocolo como datos de usuario.

PLCP (Physical Layer Convergence Procedure), Procedimiento de Convergencia de la Capa Física: Subcapa de la capa física que permite a la capa MAC IEEE 802.11 operar con la máxima independencia posible de la subcapa PMD de la capa física, simplificando la interfaz entre ambos.

PMD (Physical Medium Dependent), Dependiente del Medio Físico: Subcapa de la capa física que provee los medios para enviar y recibir datos entre dos o más estaciones (STA).

Portadora (Carrier): Onda sinusoidal que es modulada con una señal de entrada con el objetivo de enviar información.

Protocolo: Sistema de reglas que permite que dos entidades de una red puedan transmitir información entre ellas.

Qos (Quality Of Service), Calidad del Servicio: Rendimiento global de una red de computadoras, especialmente el rendimiento que es percibido por los usuarios de la red.

RA (Receiving STA Address), Dirección de la Estación Receptora: Dirección MAC individual que identifica la siguiente estación inmediata que debe ser la receptora de la información contenida en el cuerpo de la trama en el medio inalámbrico actual.

SA (Service Address), Dirección de Servicio: Dirección MAC individual que identifica la entidad MAC desde la cual fue comenzada la transferencia del MSDU o A-MSDU contenido en el cuerpo de la trama.

SDR (Software Defined Radio), Radio Definida por Software: Tipo de sistema de radiocomunicación en donde todos o algunos de los componentes que realizan el procesamiento de la señal han sido implementados mediante software que ejecuta en un procesador de propósito general.

STA (Station), Estación: Entidad lógica que constituye una unidad direccionable de manera única en el contexto de las capas de control de acceso al medio (MAC) y física.

Subcarrier: Subconjunto de un carrier modulado para transmitir información adicional.

TA (Transmitting STA Address), Dirección de la Estación Transmisora: Dirección MAC individual que identifica la estación que ha transmitido al medio inalámbrico la MPDU contenida en el cuerpo de la trama.

Trama: Paquete en el contexto de la capa de enlace de datos.

Transceptor: Dispositivo que incluye un transmisor y un receptor en la misma circuitería.

USRP (Universal Software Radio Peripheral), Periférico Universal de Software de Radio: Rango de dispositivos de radio definida por software (SDR) comercializados por la empresa Ettus Research.