UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Implementación del modelo OSI mediante radios definidos por software

TESIS

Que para obtener el título de

Ingeniero en Telecomunicaciones

P R E S E N T A

Ariel Cinco Solis

DIRECTOR DE TESIS

Dr. Luis Francisco García Jiménez

Ciudad Universitaria, Cd. Mx., 2020

Agradecimientos

Agradezco profundamente en primer lugar el apoyo que mi familia me ha brindado a través de estos años, porque sin ellos el camino hasta aquí hubiera sido más complicado. A mamá, papá, abuela, tíos y primos que siempre han estado presentes y me han enseñado que a pesar de las dificultades hay que ver el lado positivo de las cosas y no rendirse hasta lograr tus metas.

A los amigos y compañeros con quiénes pude compartir esta etapa de mi vida, que me inspiraron a ser mejor estudiante, con quiénes sufrimos para aprender lo que hemos aprendido hasta ahora, pero siempre con una sonrisa y actitud positiva. A quiénes estuvieron desde el inicio de esta travesía, pero también a aquellos que llegaron al final, cuando menos lo esperaba, e hicieron mejores los días de esta etapa.

Así mismo agradezco los consejos que mi tutor me ha brindado en todos los temas académicos y para el desarrollo de esta tesis, así como las recomendaciones de mis sinodales para mejorarla. Por otro lado agradezco el apoyo brindado por parte del proyecto DGAPA-PAPIIT IA105520.

Finalmente, infinitas gracias a la UNAM y a la Facultad de Ingeniería por ser mi casa durante estos años, darme la oportunidad de conocer personas excepcionales y brindarme esta formación que comprende una parte fundamental de mi desarrollo profesional.

Índice general

Índice de figuras

Acrónimos

ACK Acknowledgement

ADC Conversor analógico-digital

AM Amplitud modulada

ASCII American standard code for information interchange

BPSK Binary phase shift keying

CA Collision avoidence

CSMA Carrier sense multiple acces

CTS Clear to send

DAC Conversor digital-analógico

DC Direct current

DSP Procesador digital de señales

FCS Frame check sequence

FFT fast Fourier transform

FPGA Field-programmable gate array

ftp File transfer protocol

GHz Gigahertz

HPSDR High performance software defined radio

http Hypertext transfer protocol

ICMP Internet control message protocol

ID Identity document

IEEE Insitute of electrical and electronics engineers

IETF Internet engineering task force

IF Frecuencia intermedia

IP Internet protocol

ISO International organization for standardization

jpg Joint photographic experts group

LLC Logic link control

LTE Long term evolution

MAC Media access control

MDNS Multicast domain name system

MHz Megahertz

MIMO Multiple-input and multiple-output

MPDU MAC protocol data unit

MSDU MAC Service data unit

MTU Maximum transmission unit

OFDM Multiplexación por división de frecuencia ortogonal

OSI Open system interconnection

PDU Protocol data unit

ping Packet internet groper

PLCP Physical layer convergence protocol

png Portable network graphics

PPDU Physical layer protocol data unit

PSDU Physical layer service data unit

QAM Quadrature amplitude modulation

RF Radiofrecuencia

RTS Request to send

SDR Software defined radio

SDU Service data unit

SNR Relación señal a ruido

ssh Secure shell

SWIG Simplified wrapper and interface generator

TCP Transmission control process

telnet Telecommunication network

UDP User datargamm protocol

UHD USRP hardware driver

UIT Unión internacional de telecomunicaciones

USRP Universal software radio peripherial

VHDL VHSIC hardware description language

VHSIC Very high speed integrated circuits

WEP Wired equivalent privacy

Wifi Wireless fidelity

Resumen

El modelo OSI es un referente conceptual que permite establecer la comunicación entre múltiples sistemas. En general, el modelo OSI define 7 capas: aplicación, presentación, sesión, transporte, red, enlace y física. Gracias a esta abstracción, los protocolos de cada capa pueden ser analizados de manera independiente y a su vez interactuar entre ellas. Todas las capas del modelo son implementadas a través de software, a excepción de la capa de enlace de datos y capa física (capa 2 y 1, respectivamente). Estas últimas suelen ser implementadas a través de hardware propietario, por lo cual no es posible tener acceso para modificarlas.

En los últimos años se ha desarrollado una tecnología denominada *software defined radio* (SDR), la cual se compone de sistemas de radiocomunicación que pueden ser configurados o modificados mediante el uso de software en diferentes plataformas como son GNU Radio, CubicSDR, SDRangel, entre otras. Estos softwares permiten el procesamiento de señales y la implementación de protocolos de capa física. Sin embargo, GNU Radio ha tenido más aceptación en el mundo académico, ya que es un software amigable y de código abierto.

Aunque actualmente se pueden implementar las 7 capas del modelo OSI en software, no existe un módulo en GNU Radio que una las capas inferiores (2 y 1) con las capas superiores en una computadora convencional. Por esta razón, esta tesis propone una manera de unir las capas superiores, correspondientes a las capas 3, 4, 5, 6 y 7 (usualmente implementadas en software), con las capas inferiores 1 y 2 (usualmente implementadas en hardware) mediante el uso de GNU Radio. Gracias a esto, se puede tener un laboratorio donde las 7 capas del modelo OSI estén implementadas en software y de esta manera analizar la pila completa de protocolos. Además de poder probar e implementar nuevos protocolos. Para lograr este objetivo, se crea una interfaz virtual basada en un módulo del kernel de LINUX denominada TUN-TAP, y mediante el uso de un *socket* se pueden enlazar las capas superiores con las inferiores, esto con el fin de analizar los paquetes desde la capa de aplicación hasta la capa física. Específicamente, esta tesis hace uso del protocolo de comunicación IEEE 802.11, sin embargo, con este laboratorio se pueden estudiar múltiples protocolos como el IEEE 802.15.4, entre otros.

Capítulo 1

Introducción

Desde la creación del modelo OSI, por parte de la *International Organization for Standardization* (ISO) en 1984, las comunicaciones dentro de una red son explicadas y analizadas mediante 7 capas [1], las cuales modelan los diversos procesos para la transmisión y recepción de los datos. De esta forma, cada capa del modelo corresponde a un nivel de abstracción en la pila de protocolos y cada capa es dependiente de la capa anterior inmediata, ya que los datos que toma una capa, deben tener el formato adecuado para que la siguiente pueda procesarlos. Las capas que conforman el modelo OSI son la capa de aplicación (capa 7), seguida de la capa de presentación, sesión, transporte, red, enlace y finalmente la capa física (capa 1) y, de esta forma, en la literatura se habla que las capas del modelo se encuentran ordenadas de manera ascendente o descendente desde la capa 7 hasta la 1 o viceversa.

La capa de aplicación es la encargada de crear una interfaz directa con el usuario para que los datos, como son archivos de texto o mensajes de correo electrónico, puedan ser enviados dentro de la red. Algunos de los protocolos utilizados en esta capa son ftp, http, telnet y ssh.

La capa de presentación por otro lado, adapta los datos a transmitir a un formato adecuado para viajar por la red y ser recibidos en otro sistema. De esta manera, la capa de aplicación puede leer o entender los mensajes como son imágenes de tipo png, jpg o caracteres ASCII [2]. Estas dos capas comparten una relación muy directa con el usuario y con los mensajes a transmitir o recibir.

Por su parte, la capa de sesión establece la comunicación entre dos o más dispositivos para que pueda existir el intercambio de datos; por lo que aquí se define el comienzo, el control y la terminación de la comunicación entre los nodos. Esto se logra a través de diferentes servicios como son los avisos de datos recibidos, retransmisión de datos y sincronización.

La siguiente capa es conocida como la capa de transporte, la cual ofrece servicios de conexión mediante protocolos como *transmission control process* (TCP) o *user datagram protocol* (UDP) [3], que dependiendo del tipo de aplicación se utiliza uno u otro, puesto que algunos protocolos permiten verificar si los datos han llegado de manera correcta (se pueden detectar errores) e incluso en el orden deseado, además de controlar el flujo para evitar desbordamientos; es en esta capa donde se realiza la segmentación de los datos de las capas superiores para que viajen por la red.

Las capas inferiores del modelo OSI son la capa de red, enlace de datos y física. La capa de red es la encargada del direccionamiento lógico y permite que los paquetes de datos puedan ser encaminados dependiendo la prioridad de transmisión, el congestionamiento de la red, la ruta con el menor costo, entre otras métricas. En esta capa se utilizan equipos llamados *routers* que realizan la conexión de diversas subredes a dispositivos. La capa de enlace de datos se encarga por un lado del direccionamiento de manera física y por otro lado de la detección y corrección de errores de las tramas que se reciben. Finalmente, la capa física define la forma en que se hará la conexión entre los dispositivos, ya sea de manera cableada o inalámbrica; y dependiendo de lo anterior existen diversos tipos de protocolos, los cuales definen la interfaz o medio de transmisión para que los datos sean enviados en forma de *bits* y de esta manera puedan ser transmitidos y recibidos de manera correcta.

Últimamente se ha desarrollo una tecnología denominada SDR, la cual realiza procesos de

sistemas de radiocomunicaciones que suelen ser implementados en hardware mediante software, por ejemplo mezcladores, moduladores y filtros. En el área de la investigación científica recientemente se están utilizando unos radios denominados *universal software radio peripheral* (USRP), los cuales son una gama de radios diseñados con hardware capaz de transmitir y recibir datos en un amplio rango de frecuencias. GNU Radio por otro lado, es un software libre que interactúa con los USRP con la finalidad de analizar e implementar protocolos de comunicación mediante bloques programables. Estos dos agentes (USRP y GNU Radio) conforman un sistema completo de radiocomunicación, ya que son capaces de conjuntar las tareas de los protocolos de capa física y de enlace de datos del modelo OSI. Por un lado, el USRP cumple con las funciones de transmisión y recepción y por otro lado, GNU Radio con las funciones de procesamiento de señales.

Aunque el conjunto USRP-GNU Radio es capaz de desempeñar los protocolos de capa 2 y capa 1, no existe un módulo en GNU Radio que permita unir las capas superiores del modelo OSI, correspondientes a las capas 3, 4, 5, 6 y 7, con las capas 2 y 1; esto con el propósito de tener un laboratorio en software desde capa de aplicación hasta capa física. Por ello, en esta tesis se propone unir las capas superiores con las inferiores mediante una interfaz virtual basada en un módulo del kernel de LINUX denominada TUN-TAP, y así poder analizar toda la pila de comunicación y a su vez probar y crear nuevos protocolos. Específicamente, esta tesis utiliza el USRP N210 y el protocolo IEEE 802.11 como protocolo de capa física.

1.1. Definición del problema

El modelo OSI es un modelo de referencia de interconexión compuesto por 7 capas donde en el ámbito académico, cada una de las capas son estudiadas de manera separada por la facilidad que esto implica, pero desafortunadamente, no se tienen herramientas para probar toda la pila de protocolos de manera conjunta, dado que capa 2 y 1 suelen estar implementadas por hardware propietario al que no se tiene acceso. Esto último, en el ámbito académico y sobre todo de investigación, implica una limitante puesto que no se puede analizar o implementar nuevos protocolos en toda la pila de comunicación. Por otro lado, en los últimos años se ha desarrollado una tecnología denominada SDR, la cual se compone de sistemas de radiocomunicaciones programables mediante el uso de software de uso libre llamado GNU Radio, que permite la implementación de diversos protocolos en capa física. Estas herramientas pueden ser utilizadas para el desarrollo de nuevos protocolos en el ámbito de redes inalámbricas.

Sin embargo, aunque actualmente se pueden implementar las 7 capas en software, no existe un módulo en GNU Radio que conjunte las capas superiores con las inferiores. Esta unión permitirá manipular todos los procesos que suceden durante la comunicación. Además, permitirá analizar diversos tipos de protocolos existentes o implementar nuevos protocolos con el fin de mejorar los servicios de comunicación en un futuro.

1.2. Hipótesis

La creación de una interfaz virtual TUN-TAP como un módulo programable en GNU Radio permite conectar todas las capas del modelo OSI a través de software, eliminando el uso de hardware propietario.

1.3. Meta general.

Desarrollar un bloque en GNU Radio que permita conjuntar las capas superiores (aplicación, sesión, transporte y red) con las capas inferiores (enlace y física) con el objetivo de crear un laboratorio donde se puedan estudiar los diversos tipos de protocolos de comunicación.

1.4. Objetivos específicos.

- Instalar y configurar el módulo IEEE 802.11 en GNU Radio.
- Realizar el análisis del funcionamiento de una interfaz virtual utilizando el módulo del kernel de Linux TUN-TAP y así mismo, poder escribir paquetes por medio de la biblioteca Scapy de Python.
- Creación de un bloque en GNU Radio para conjuntar la interfaz virtual TUN-TAP con la capa física implementada en GNU Radio.
- Implementar un flowgraph en GNU Radio que permita la recepción y transmisión de señales a través del protocolo IEEE 802.11, donde sea posible modificar características como la modulación y ganancia de la señal.
- Realizar pruebas que muestren que los paquetes creados en capas superiores sean enviados mediante el protocolo IEEE 802.11.

1.5. Contribución

Esta tesis propone la creación de un bloque en GNU Radio que permita unir las capas superiores del modelo OSI, correspondientes a las capas 3, 4, 5, 6 y 7, con las capas inferiores (1 y 2), a través del uso de una interfaz virtual que está basada en un módulo del kernel de LINUX denominado TUN-TAP con el fin de tener un laboratorio en software de todas las capas que componen al modelo OSI. Específicamente, sobre el protocolo de comunicación IEEE 802.11. Sin embargo, puede probarse con cualquier otro tipo de protocolo, ya que el utilizar la tecnología SDR en combinación con una computadora personal brinda una gran flexibilidad y se espera que en un futuro se puedan desarrollar y probar nuevos protocolos de comunicaciones utilizando las herramientas de esta tesis.

1.6. Estructura de la tesis

- El capítulo 2, presenta una revisión del estándar IEEE 802.11, así como una descripción de la tecnología SDR. Además, se presentan algunos de los trabajos más relevantes con esta tecnología en el área de las comunicaciones inalámbricas.
- El capítulo 3, presenta las herramientas a utilizar para lograr la meta de esta tesis. Específicamente, se presenta la forma de instalar GNU Radio en una computadora personal, como crear y manejar una interfaz virtual del sistema operativo, así como la creación de *sockets* y generación de paquetes que permitan el enlace de las diferentes capas.
- El capítulo 4, describe el proceso de escritura de paquetes a la interfaz virtual (TAP) y se verifica que los paquetes lleguen a la interfaz a través de un *socket* UDP. Posteriormente son enviados a capa física del USRP. Además, se presentan algunas pruebas variando el número de paquetes. También se presenta una prueba que simula el protocolo IEEE 802.11 y otra que envía los paquetes a un transmisor real usando el USRP N210.
- El capítulo 5, presenta las conclusiones, la verificación de la hipótesis y las perspectivas de investigación.

Capítulo 2

Estado del Arte

2.1. SDR

Desde 1992 J. Mitola propuso el concepto de radio definido por software [5] y la Unión Internacional de Telecomunicaciones (UIT) define formalmente a la tecnología SDR [4] como:

> "Un transmisor y/o receptor de radio que emplea una tecnología que le permite operar con parámetros de radio frecuencia que incluyen, pero no se limitan, a rango de frecuencia, tipo de modulación o potencia de salida, que pueden ser configurados o modificados mediante software."

En un SDR, las señales se procesan de manera digital mediante un conversor analógico digital (ADC), el cual permite que las señales analógicas del circuito de radiofrecuencia (RF) (ver figura 2.1) se transformen en señales digitales para que se realicen funciones de modulación y demodulación, codficación de canal, codificación de fuente, entre otros. El tratamiento de las señales digitales, se realiza cuando la señal se encuentra en banda base, es decir, cuando la frecuencia de la señal no ha sufrido cambios por la modulación de portadora. Además, el sistema de RF cumple con bajar la frecuencia de la señal recibida a un frecuencia intermedia (IF) para que pueda ser procesada con software como GNU Radio.

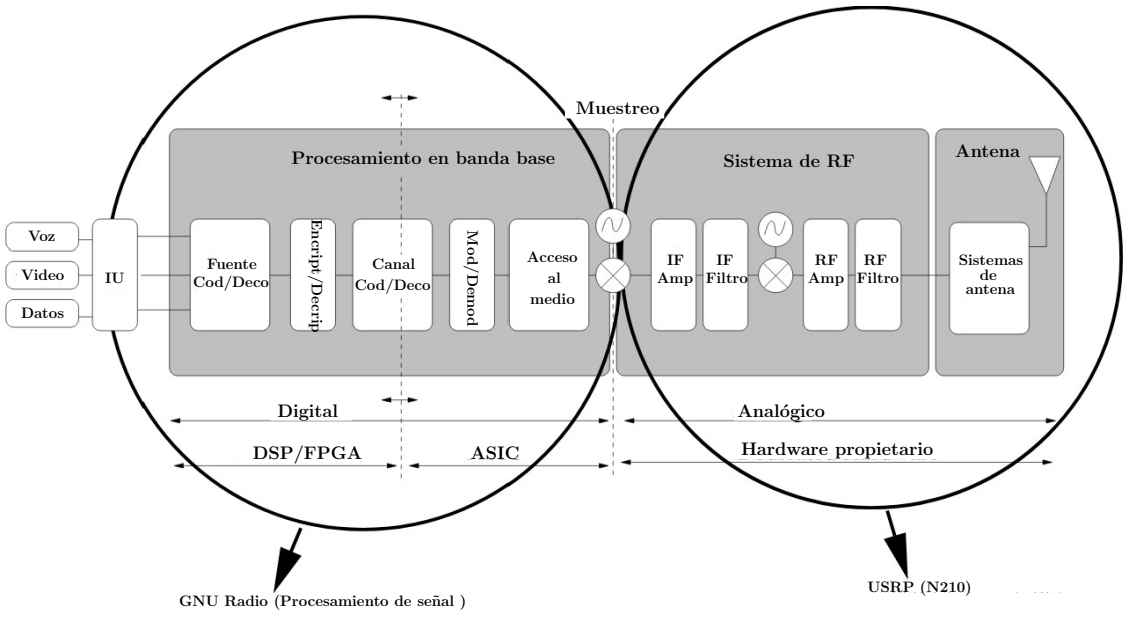

Figura 2.1: Estructura básica de un SDR [9].

En la actualidad, existe una gran variedad de proyectos que se consideran como parte de la tecnología SDR, unos de ellos es denominado *high performance software defined radio* (HPSDR), el cual es un proyecto de hardware y software de código abierto, cuyo propósito es dividir el diseño general del radio en diversos módulos que estarán conectados entre sí mediante un bus, para permitir a los usuarios incorporar los módulos que desean, así como diseñar sus propias variantes. Hasta ahora, los módulos varían desde simples filtros pasa banda e interfaces de entrada/salida hasta funciones de un procesador digital de señales (DSP) completas [6]. Otro ejemplo es el proyecto denominado como WebSDR, el cual proporciona acceso a través de un navegador de Internet a múltiples receptores SDR en todo el mundo, es decir, cada usuario puede sintonizar de manera independiente para escuchar distintas señales mediante los servidores de WebSDR [7]. Pero en general, los más utilizados es la pareja GNU Radio y los USRP, los cuales se componen principalmente de una FPGA y una interfaz de RF de alta gama.

2.1.1. GNU Radio

De acuerdo con [8], GNU Radio es un software de desarrollo libre y código abierto que proporciona bloques de procesamiento de señales de radiofrecuencia. En general, el software es utilizado en conjunto con dispositivos externos de hardware de radiofrecuencia, pero también puede ser utilizado sin hardware debido a que permite realizar simulaciones de los mismos procesos e incluso simular adversidades del ambiente que afectan a las señales durante su propagación.

La estructura de GNU Radio se encuentra dividida en dos partes principales que hacen que a través de los denominados *flowgraphs* se implementen las tareas que normalmente en un radio común se implementan con hardware como por ejemplo moduladores, codificadores y filtros. La primera parte se encarga de realizar las tareas de procesamiento de las señales y es implementada en el lenguaje de programación *C++*. La segunda parte es programada en lenguaje *Python* y sirve para organizar, conectar y crear los *flowgraphs* (ver figura 2.2) que permiten realizar en software las tareas ya mencionadas. Adicionalmente a estas dos partes que sirven para el funcionamiento básico de los bloques del software, se utiliza la herramienta de desarrollo SWIG que funciona como un enlace entre *C++* y *Python* [10].

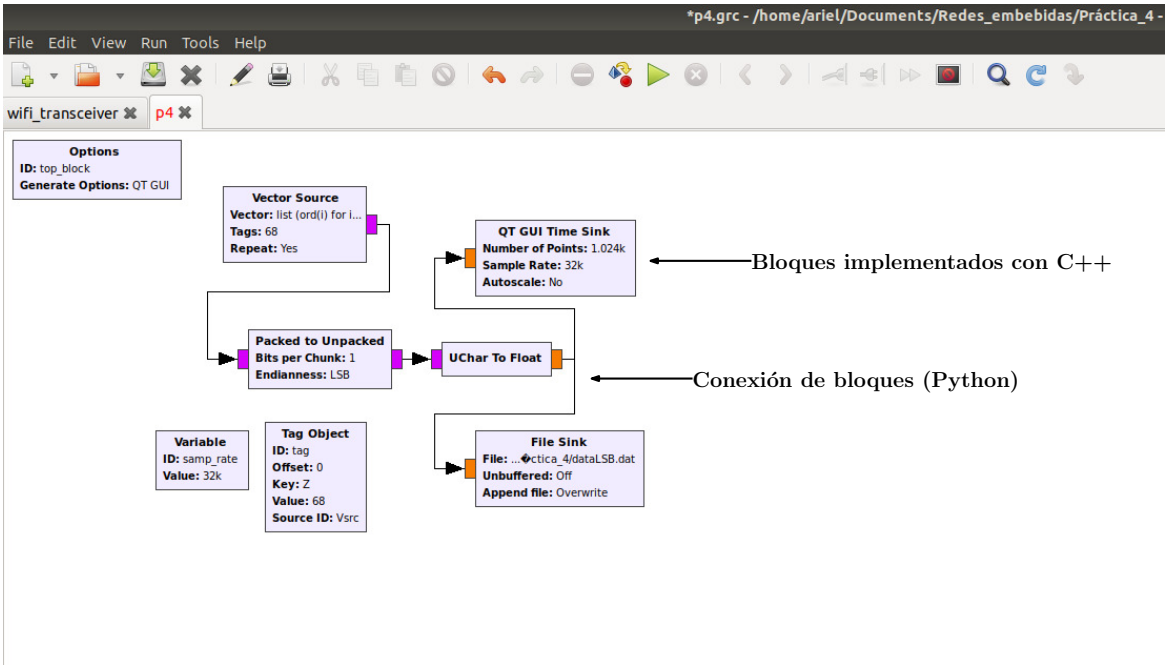

Figura 2.2: Ambiente gráfico de GNU Radio.

Cada programa en GNU Radio debe contar con los siguientes elementos para funcionar de manera correcta. El primero debe ser un bloque fuente único, que es el inicio del *flowgraph*, como por ejemplo un archivo que contenga los mensajes a transmitir, aunque también puede ser el USRP que esta recibiendo información del medio. El segundo elemento es denominado como sumidero, que se presenta al final de cada *flowgraph* y de igual forma que para la fuente, el sumidero puede ser un USRP, en este caso para transmitir o un archivo para escribir el mensaje recibido. Además de estos elementos, cada *flowgraph* cuenta con bloques intermedios por donde se va transfiriendo la información y a su vez recibe el tratamiento deseado para ser transformada en una señal de radio.

2.1.2. USRP.

Un USRP es un dispositivo cuyo hardware flexible permite que se pueda crear una conexión entre los sistemas de radiofrecuencia y el procesamiento digital de señales, ya que está constituido generalmente por una placa base con una FPGA para el procesamiento de señales a una alta velocidad y una o más placas secundarias con sistemas de radiofrecuencia que cubren diferentes gamas de frecuencias para aplicaciones y tecnologías como MIMO, LTE, Wifi y sistemas de radar [27]. Por lo que un USRP como el que se observa en la figura 2.3, puede trabajar en frecuencias desde el rango de AM, hasta las frecuencias de estándares como el IEEE 802.11ac.

Figura 2.3: USRP N210 [15].

De esta forma, el uso de los USRP y GNU Radio en conjunto han permitido desarrollar todo tipo de experimentos que han abierto la posibilidad de inspeccionar y mejorar los diversos protocolos existentes para capa física, en especial el estándar de Wifi de la IEEE 802.11 en varias de sus versiones. Por ejemplo en [11], los autores presentan un receptor de multiplexación por división de frecuencia ortogonal (OFDM) que fue implementado mediante GNU Radio para operar con un USRP N210. Es considerado como uno de los primeros prototipos que analizaba el encabezado MAC y extraía la carga útil en redes del estándar IEEE 802.11 a/g/p; y la contribución se dejó como un código abierto para que se pudiera experimentar nuevos algoritmos de procesamiento de señales.

Por otra parte, muchos de los estudios que se realizan con este tipo de tecnología sirven para analizar los efectos que el medio puede causar a las señales que se envían de manera inalámbrica como es la interferencia y el ruido. Un ejemplo de lo anterior se presenta en [13], donde los autores mediante simulaciones y experimentos reales observaron que la interferencia proveniente de dispositivos cercanos en el estándar 802.11p tiene el mismo efecto negativo que el ruido.

También se ha impulsado el desarrollo de nuevos avances en diversas áreas. En [12], por ejemplo, se evalúa el rendimiento del estándar 802.11 en un entorno acuático. Se demostró que la tasa de transmisión y el rango de alcance aumenta si se utilizan frecuencias por debajo de los GHz. Por otra parte, en [14], se propone la detección e identificación de gestos humanos a través de dispositivos Wifi.

2.2. Estándar 802.11

El estándar 802.11, más conocido como Wifi, describe la capa física y la capa de enlace de datos en las redes inalámbricas de área local. Ambas capas están a su vez divididas en dos subcapas como se muestra en la figura 2.4.

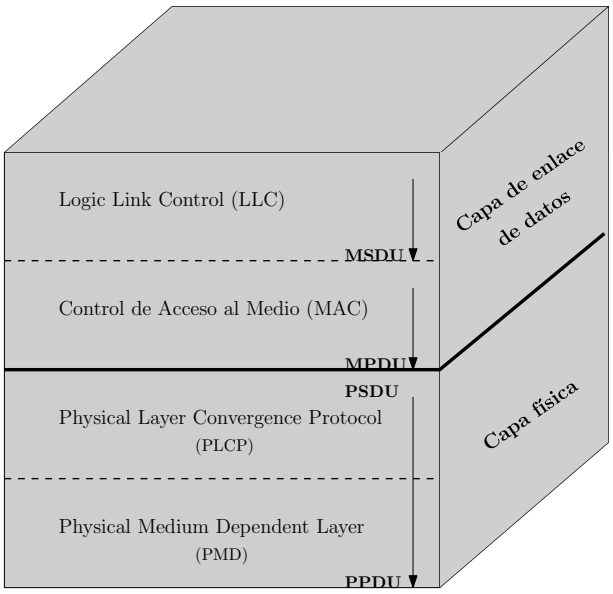

Figura 2.4: Protocolo IEEE 802.11[17].

De manera general, los mensajes recibidos desde capas superiores son encapsulados antes de transmitirse por el canal inalámbrico en la capa de enlace de datos y sus subcapas, así como en la capa física. Los datos inicialmente reciben el nombre de *service data unit* (SDU), dado a que aún no han sido encapsulados. Una vez que la SDU se transfiere de la subcapa LLC *(logic link control)* a la subcapa MAC, se le añaden 30 bytes de encabezado MAC y 4 bytes de frame check sequence (FCS), los cuales en conjunto forman la *unidad de datos del protocolo MAC* (MPDU). Posteriormente, esta unidad de datos se transfiere a la capa física, allí se le conoce como *unidad de datos de servicio de capa física* (PSDU). Finalmente, esta unidad de datos se traspasa a la subcapa *physical layer convergence protocol* (PLCP), la cual agrega un preámbulo y un encabezado para generar la *physical layer protocol data unit* (PPDU), que es la trama final envíada en capa física [17].

Por otra parte, es importante destacar que la capa MAC usa el protocolo *carrier sense multiple access* (CSMA) en conjunto con un sistema de prevención de colisiones denominado *collision avoidance* (CA), que consiste en escuchar el canal y transmitir únicamente cuando éste se encuentre libre de otras transmisiones; esto se realiza mediante mensajes como *request to send* (RTS), *clear to send* (CTS), *acknowledgement* (ACK) y *beacons*. Mientras que para la capa física, el estándar especifica tres técnicas de transmisión que son *frequency-hopping spread-spectrum*, *direct-sequence spread-spectrum* e *infrarred light*. Aunque, en la actualidad, la técnica más usada es conocida como *orthogonal frequency division multiplexing* (OFDM) [17].

OFDM es una técnica en la cual se divide el canal de comunicación en un número de bandas de frecuencias con igual espaciamiento y además ortogonales entre sí. Una subportadora lleva una parte de los datos de usuario y es transmitida en cada banda; por lo que al ser ortogonales se puede usar todo el espectro sin que las subportadoras se interfieran unas con otras. Lo anterior se puede apreciar de mejor forma en la figura 2.5, donde se observa que para cada máximo de una portadora, las demás valen cero y, las oscilaciones de cada subportadora se "contrarrestan" con las de las demás, por lo que prácticamente solo se podría considerar el lóbulo principal de cada portadora [17].

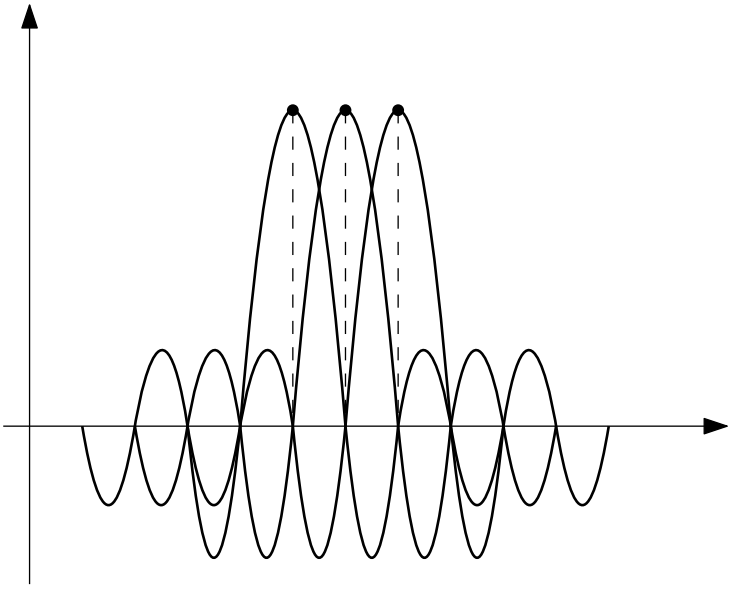

Figura 2.5: Espectro de OFDM. [17]

Por lo tanto, entre los beneficios de utilizar OFDM se encuentran la alta eficiencia espectral, ya que no se utilizan bandas de guarda entre portadoras; resistencia a la interferencia de otras señales de RF, además de una baja distorsión por multitrayecto [17].

2.3. Resumen del capítulo

En este capítulo se presentó una revisión del estado del arte respecto a las principales características de la tecnología SDR, el uso de los USRP y GNU Radio, los cuales brindan una gran flexibilidad, ya que permiten el procesamiento de señales digitales y la transmisión de señales de RF desde una computadora personal. Así mismo, se presentaron algunos ejemplos en los que se ha utilizado dicha tecnología para caracterizar diversos protocolos de capa física, buscando el desarrollo de nuevos avances en diversas áreas donde se utilice comunicación inalámbrica, puesto que como se ha mencionado, en general se utiliza hardware propietario para la implementación de los protocolos que corresponde a dicha capa. Además, se ofrece una pequeña descripción del estándar IEEE 802.11 y de la técnica principal para la transmisión de paquetes OFDM.

Capítulo 3

Herramientas de desarrollo

3.1. GNU Radio

A continuación se describe el proceso de instalación de GNU Radio en el sistema operativo Ubuntu 18.04. Como se observa posteriormente en esta tesis, se requiere modificar el código de los bloques, por lo que se debe instalar GNU Radio desde código fuente, es decir, se debe compila todo el proyecto. De la misma forma, es necesario instalar el *driver USRP hardware driver* (UHD) para poder utilizar el USRP N210 en conjunto con GNU Radio.

Antes de realizar la instalación, se verifica que todas las dependencias que corresponden al sistema operativo Ubuntu 18.04, mostradas en el apéndice A, se encuentren instaladas. De esta forma, se descarga el repositorio del *driver* (versión 3.13)[1] y se descomprime dentro de una carpeta de nombre *workarea* previamente creada. Posteriormente se compila el proyecto y se instala dentro de la carpeta con los comandos que se muestran a continuación:

3.1: Comandos instalación en Ubuntu.

```
$ mkdir build
$ cd build
$ cmake ../
$ make
$ sudo make install
$ sudo ldconfig
```

De manera similar, una vez instalado el *driver* del UHD, se descarga GNU Radio del repositorio github [2]. Para asegurar estabilidad del software, se descarga la versión 3.7.13.3, debido a que en ocasiones al utilizar la versión más actual causa problemas con algunos de los bloques y su funcionamiento. Después se descomprime en una carpeta llamada *workarea-gnuradio* y se siguen los comandos del cuadro 3.1 para su compilación e instalación.

La capa física 802.11 no se encuentra instalada por defecto con el software GNU Radio, por lo que es necesario realizar la instalación de este módulo (*gr-ieee802-11*) y, a su vez de un módulo llamado *gr-foo* que permite realizar la conexión entre el módulo de wifi y el software *Wireshark*, ya que el módulo permite la captura de paquetes en archivos *pcap*.

Para realizar la instalación del módulo *gr-ieee802-11*, es necesario descargar del repositorio de github[3] el branch que concuerde con la versión de GNU Radio, en este caso al ser la versión 3.7.13.3, se debe descargar el branch *maint*-3.7. Una vez descargada, se descomprime dentro de la misma carpeta donde se instaló GNU Radio y se escriben los comandos mostrados en el cuadro 3.1 para su compilación e instalación.

Por otro lado, para instalar *gr-foo* se debe verificar de la página de github[4] que el branch concuerde con la versión de GNU Radio, por lo que en este caso, se debe escoger el *maint*-3.7.

[1] Descarga del repositorio de UHD en la página: https://github.com/EttusResearch/uhd

[2] Repositorio de GNU Radio: https://github.com/gnuradio/gnuradio/tree/v3.7.13.4

[3] Descarga de módulo de wifi para GNU Radio: https://github.com/bastibl/gr-ieee802-11/tree/maint-3.7/

[4] Descarga de módulo de foo para GNU Radio: https://github.com/bastibl/gr-foo

Finalmente, se descomprime el archivo en la misma carpeta de instalación de GNU Radio y se siguen los comandos del cuadro 3.1 para instalar este módulo. Si ambos módulos fueron instalados de forma correcta, al abrir el software GNU Radio se deben mostrar los bloques pertenecientes a cada módulo como se observa en la figura 3.1.

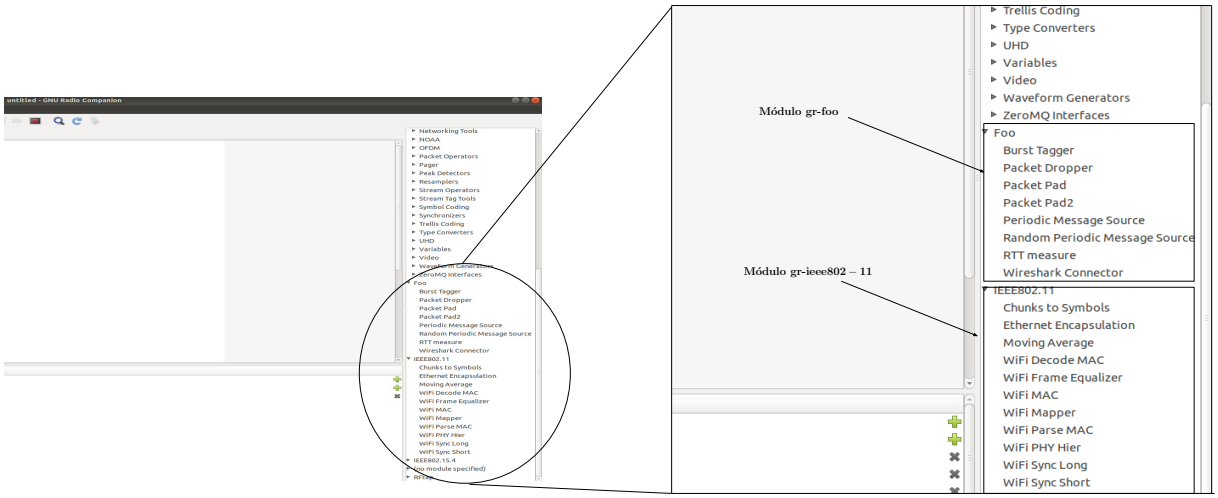

Figura 3.1: Módulos de gr-ieee802.11 y gr-foo instalados correctamente en GNU Radio.

3.2. TUN-TAP

TUN-TAP se refiere a una interfaz de red que solamente existe en el kernel del sistema operativo. A diferencia de las interfaces de red normal, por ejemplo la que tiene cualquier computadora, ésta no tiene componentes de hardware, por lo cual en lugar de utilizar un cable físico para su conexión, se requiere de un descriptor de archivo, el cual permite leer los datos que contiene, pero también escribir en ella como si la interfaz estuviera enviando o recibiendo datos de red por un cable [16].

Las interfaces TAP se utilizan para paquetes Ethernet, mientras que las interfaces TUN se utilizan para paquetes IP; esto se define al momento en que se crea la interfaz. Además, una interfaz TUN-TAP puede ser creada y eliminada (interfaz transitoria) en el momento que se requiera. También puede ser creada de forma permanente (interfaz persistente), pero ambas pueden ser utilizadas como cualquier interfaz, ya que se le puede asignar una dirección IP, analizar tráfico, crear reglas de firewall, entre otras cosas [16].

Existen diversas formas y comandos para crear una interfaz de red TAP en Ubuntu 18.04, pero en este caso se creará de manera transitoria con los comandos que se muestran en el cuadro 3.2.

3.2: Comandos para crear interfaz TAP.

```
$ ip tuntap add dev tap0 mode tap
$ ip link set tap0 up
$ ifconfig tap0 192.168.20.1/24 netmask \
    255.255.255.0 broadcast 192.168.20.255
```

Es importante destacar que para poder crear la interfaz se deben tener permisos de super usuario, que es equivalente en Ubuntu a usar el prefijo *sudo* en cada comando. El primer comando crea la interfaz de nombre *tap0* y modo TAP (paquetes Ethernet). Sin embargo, como se ve en la figura 3.2, la interfaz virtual no aparece, esto es debido a que se creó pero

aún no esta activa, es decir, es como si el puerto Ethernet no estuviera “conectado” a ningún cable.

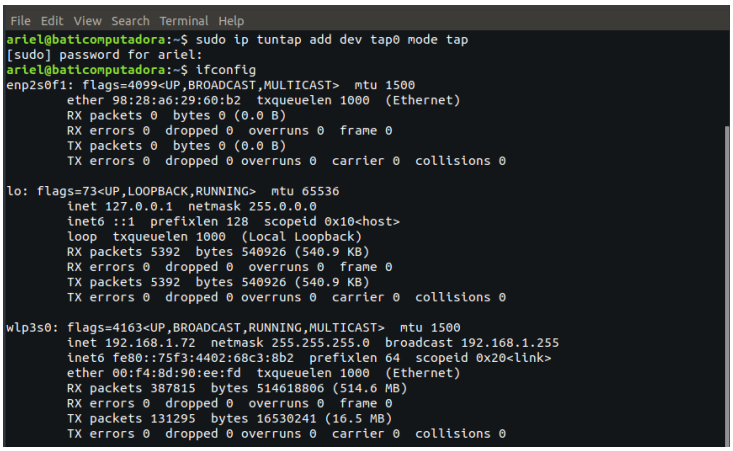

Figura 3.2: Interfaz TAP creada pero sin conectar.

Los siguientes comandos del cuadro 3.2 activan la interfaz. También se definen sus parámetros de red: dirección, netmask y broadcast. En la figura 3.3, se puede observar la interfaz con los parámetros configurados. Además, en la figura 3.4 se realiza un *ping (packet internet groper)* a la interfaz de manera que se comprueba que ésta tiene el mismo funcionamiento que cualquier otra interfaz del sistema.

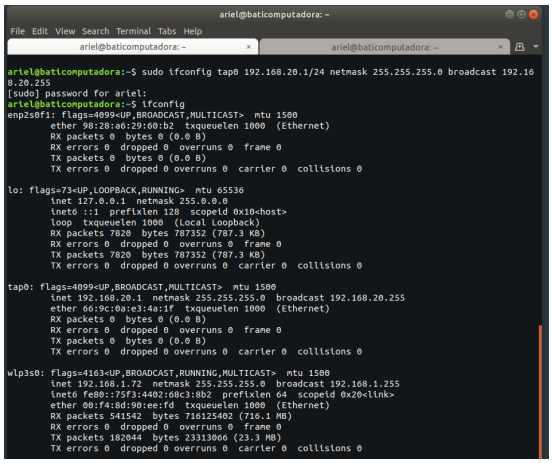

Figura 3.3: Interfaz TAP creada y lista para su funcionamiento.

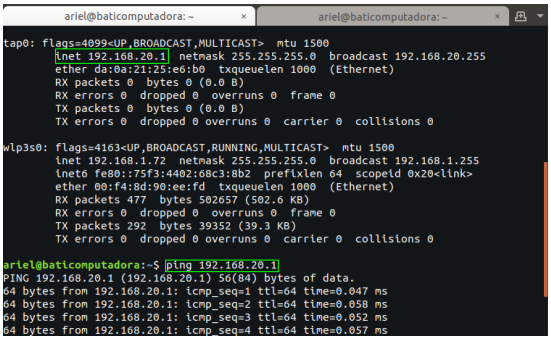

Figura 3.4: comando *ping* a la interfaz virtual con dirección 192.168.20.1.

3.3. Interfaz virtual en GNU Radio

GNU Radio cuenta con un bloque denominado TUNTAP, cuyo funcionamiento esta programado en lenguaje *C++* y permite crear una interfaz virtual de tipo TAP para paquetes Ethernet o tipo TUN para paquetes IP, de igual forma como si se estuviera creando a través de los comandos mostrados en el cuadro 3.2, a excepción de que no se definen los parámetros como dirección IP, netmask o broadcast; esto se debe hacer siempre a través de comandos en la terminal de Ubuntu. Este bloque es de vital importancia para la contribución de esta tesis, debido a que es el túnel para unir las capas 1 y 2 del modelo OSI con el resto.

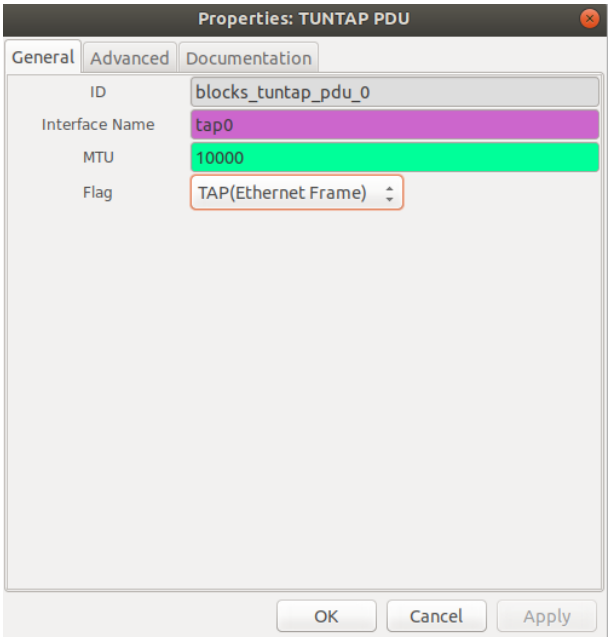

Figura 3.5: Propiedades del bloque TUNTAP en GNU Radio.

En la figura 3.5 se muestran las propiedades del bloque TUNTAP en GNU Radio como son el ID del bloque, el cual se nombra por defecto en el programa y es solamente para reconocer el nombre del archivo que se genera automáticamente al ejecutar el diagrama de flujo para ese bloque. Por otra parte, el campo correspondiente a *Interface Name* es para colocar el nombre de la interfaz virtual, aunque es importante mencionar que si la interfaz ya ha sido creada previamente mediante el sistema operativo, el bloque solamente ligará el nombre que se coloque en este campo con la interfaz del sistema correspondiente, aunque para poder manipular la interfaz desde GNU Radio es necesario manipular el código en *C++* de este bloque, debido a que se requiere trabajar con el descriptor de archivo perteneciente a la interfaz TUNTAP.

Otro campo importante para el bloque es la definición del *maximum transmission unit* (MTU), el cual es un término que se utiliza para definir el tamaño de la trama más grande que puede ser enviado, dependiendo del protocolo de comunicación que se utilice [19]. Por ejemplo, para el estándar de Ethernet se define un *MTU* de 1500 bytes, por lo que si la interfaz TAP se crea con los comandos del sistema, automáticamente es configurada con un *MTU* de 1500 bytes, como se muestra en la figura 3.3; y aunque este valor se cambie en el bloque de GNU Radio, se conservará el valor por defecto que le asigna el sistema operativo.

3.3.1. Creación de socket en la interfaz.

Como se mencionó anteriormente, para poder manipular la interfaz TUNTAP que se crea desde el sistema operativo a través del bloque de GNU Radio, es necesario identificar el descriptor de archivo de dicha interfaz. Para ello, se busca dentro de la carpeta de instalación de GNU Radio el código en *C++* del bloque; en el caso del sistema operativo *Ubuntu*, puede buscarse desde la carpeta de archivos del sistema, que es un modo más amigable debido a que es un ambiente gráfico, o también puede hacerse desde la terminal del sistema con el comando que se muestra en el cuadro 3.3.

3.3: Comando para buscar un archivo en Ubuntu.

```
$ sudo locate tuntap
```

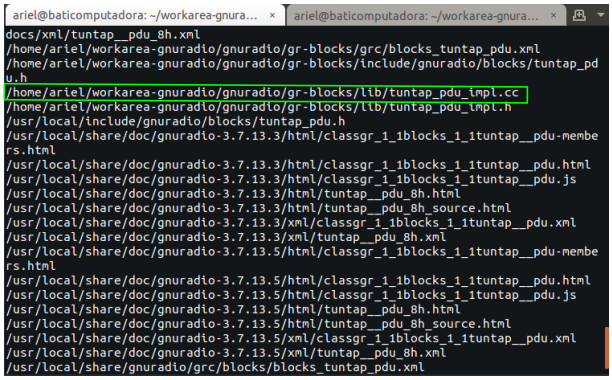

Figura 3.6: Archivos relacionados con bloque TUNTAP en GNU Radio.

La figura 3.6 muestra diversos archivos para el bloque TUNTAP, algunos sirven para visualizar la interfaz gráfica en el flowgraph, como son los archivos de extensión *xml*. El archivo *tuntap_pdu_impl.cc* es el archivo que se debe modificar para escribir los paquetes y así unir las capas superiores del modelo OSI con las capas 1 y 2.

Un *socket* es utilizado generalmente en ambientes de aplicación cliente-servidor y se puede definir como una interfaz programable que se utiliza como punto final para permitir la comunicación entre procesos de una computadora o de diversas computadoras en una red, es decir, es una forma de enviar y/o recibir mensajes utilizando descriptores de archivos, que como su nombre lo indica son los que permiten la administración del archivo donde se va a escribir o leer los datos de la comunicación. Existen tres tipos de sockets disponibles para usuarios, los cuales son [20]:

- *Stream sockets*: permiten la comunicación mediante el protocolo TCP, esto es, proporciona flujo de datos bidireccional que garantiza confiabilidad, no duplicado y que los datos sean secuenciales, es decir, si se envía por ejemplo "1, 2, 3", se recibirán el mismo orden y una vez recibidos se pueden escribir o leer en los *sockets* como una secuencia de bytes. Si existe un error durante la comunicación se envía un mensaje de error para que los datos se vuelvan a mandar.

- *Raw sockets*: no están destinados al usuario general; se usan principalmente en el desarrollo de nuevos protocolos de comunicación o para obtener acceso a algunas de las características más especiales de un protocolo existente. Solo los procesos de super usuario pueden usar sockets raw.

- *Datagram sockets*: al igual que los stream sockets, permiten el flujo bidireccional de datos, pero en este caso los mensajes pueden o no mantener la misma secuencia, es decir, los mensajes pueden llegar al destino en orden o mezclados e incluso pueden llegar a duplicarse. Permiten procesos de comunicación del protocolo UDP.

El protocolo UDP pertenece a un grupo de protocolos de internet en la capa de transporte del modelo OSI. Este protocolo ofrece a las aplicaciones de usuario enviar datragramas (paquetes de datos) IP encapsulados sin establecer una conexión. Las características de este protocolo se describen en el RFC768 de la IETF [21] y entre ellas se encuentra el ser considerado un protocolo del tipo *best-effort*, debido a que aunque no garantiza que los datos lleguen a su destino en orden, o sin duplicarse, hace lo posible para transmitir todos los datagramas que se quieran enviar. Una de las principales funcionalidades de este protocolo es el transporte de datos de manera más rápida que otros protocolos como por ejemplo TCP, donde se requiere establecer y aceptar la conexión primero para poder enviar los paquetes.

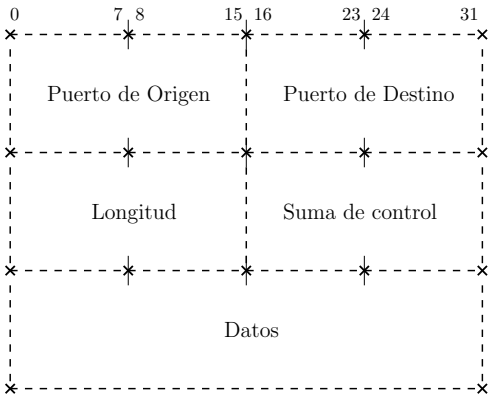

Figura 3.7: Formato de un datagrama UDP [21].

Un paquete UDP consiste en una cabecera de 8 bytes en conjunto con los datos (ver figura 3.7). Para la cabecera, se reservan 4 bytes para la asignación de puertos tanto de origen como de destino, además se reservan 2 bytes para la longitud del paquete UDP y 2 más para la suma de comprobación que se obtiene a partir de la información de la cabecera IP, UDP y los datos.

Con todo lo anterior se puede crear el túnel entre las capas 1 y 2 del modelo OSI con el resto de capas. Para ello se debe modificar el archivo que tiene por nombre *tuntap_pdu_impl.cc* (también mostrado en el apéndice B), el cual contiene el descriptor de archivo de la interfaz virtual TAP que se crea previamente con las instrucciones del cuadro 3.2. La modificación consiste en agregar el código que se muestra a continuación en la línea 136 del código del apéndice B, para realizar el *socket* UDP que permite enviar y recibir mensajes en la interfaz, así como la implementación de un hilo, para que el *socket* y la recepción de información en la interfaz TAP puedan trabajar concurrentemente y permita el flujo del *flowgraph* de GNU Radio.

```
1  //------------------------------------------------------------
2        //Se implementa el socket para poder meter paquetes al tap
3        int sockfd;  // socket
4        struct sockaddr_in servaddr; // Dirección del server
5
6        // Creación de socket
7        sockfd = socket(AF_INET, SOCK_DGRAM, 0);
8        //Verificar si se creo correctamente
9        if (sockfd == -1) {
10         printf("socket no creado...\n");
11         exit(0);
12       }
13       else
14         printf("socket creado..\n");
15
16       bzero(&servaddr, sizeof(servaddr));
17
18       // Asignación de puerto e IP
19       servaddr.sin_family = AF_INET;
20       servaddr.sin_addr.s_addr = inet_addr("192.168.20.1");
21       servaddr.sin_port = htons(7656);
22
23       // Verificación y enlace del socket con la IP dada
24       if ((bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)))
          ↪  != 0) {
25         printf("Falló el enlace con el socket...\n");
26         exit(0);
```

```
27        }
28        else
29          printf("Socket enlazado correctamente.\n");
30
31        std::thread t1(recibir,sockfd,fd);
32        t1.detach();
```

Como se observa en el código anterior, se especifica que el *socket* es de tipo UDP mediante el comando *SOCK_DGRAM* en la línea 7, además que se debe especificar la dirección y puerto en donde se realizará la conexión como se muestra en las líneas 20 y 21 (en este caso se asigna la dirección de la interfaz TAP). El resto del código se utiliza para verificar la correcta creación del socket y el enlace del mismo a través de la IP definida.

Finalmente se crea un hilo en la línea 31, que se liga a una función de nombre *recibir()*, la cual se usa para escribir los mensajes que se reciben en la interfaz TAP desde alguna aplicación de cliente; para ello, es necesario brindar los parámetros del *socket* y el descriptor de archivo de la interfaz TAP a la función *thread()*. Y como se muestra en el código siguiente (que se puede agregar en la línea 13 del código del apéndice B), para la escritura de paquetes en la interfaz, se puede utilizar la función *write()* (línea 19) proporcionando los parámetros del descriptor de archivo del TAP y los del *socket* UDP creado para transportar los paquetes que vienen desde capas 5, 6 y 7, correspondientes a la capas superiores del modelo OSI.

```
1  //-----------------------------------------------------------------
2      void recibir(int s, int descript){
3
4      struct sockaddr_in clientaddr; //Dirección del cliente
5      int clientlen; // tamaño en bytes de la dirección del cliente
6      int n; //tamaño del mensaje en bytes
7      char data[MAX]; //Mensaje
8
9      while(1){
10         //printf("Estoy recibiendo\n" );
11         memset(&data, 0, sizeof(data));
12         n = recvfrom(s, data, BUFSIZE, 0,
13                  (struct sockaddr *)&clientaddr, (socklen_t*)&clientlen);
14         if (n < 0){
15             printf("ERROR al recibir\n");
16             exit(0);
17         }
18         printf("Recibí %d bytes\n", n);
19         write(descript, data, sizeof(data));
20         sleep(1);
21     }
22 }
```

Es importante destacar que si algunas de las bibliotecas que aparecen al inicio del código del apéndice B no se encuentran agregadas, los códigos anteriores no funcionarán de forma adecuada, por lo que se recomienda revisar detalladamente y añadir las bibliotecas faltantes. Además para que se realicen los cambios en el código del bloque TUNTAP de manera permanente y éste pueda cumplir las nuevas funciones que se le agregan, es necesario volver a compilar el proyecto de GNU Radio (de ahí la importancia de la instalación desde código fuente). Para compilar de nuevo basta con ingresar desde la terminal a la carpeta donde se tiene instalado el software (en este ejemplo *workarea-gnuradio*) y específicamente a la carpeta *build* y escribir los comandos que se muestran a continuación.

3.4: Comandos para recompilar GNU Radio.

```
$ cmake ../
$ make
$ sudo make install
$ sudo ldconfig
```

3.4. Capa física en GNU Radio

Para la implementación de capa física, en particular del protocolo Wifi IEEE 802.11, se toma el módulo ya compilado (*gr-ieee*-802-11) que fue desarrollado por Bloessl [11]. La figura 3.8 muestra un transmisor de Wifi. Este flowgraph se encuentra dentro de la carpeta del módulo *gr-ieee*-802-11 con el nombre *Wifi_tx.grc*, el cual tiene como fuente un bloque llamado *Message Strobe* que envía como mensaje una cadena de "x" cada cierto tiempo. Además tiene conexión al bloque *Socket PDU (protocol data unit)*, el cual se utiliza para establecer un puerto UDP de destino para los paquetes enviados; pero ambos bloques deben de ser reemplazados con el bloque *TUNTAP PDU* descrito en la sección anterior, el cual será la fuente para el flowgraph, es decir, de donde se toman los mensajes para enviarlos por el medio, como se muestra en la figura 3.9. Además, se debe añadir el sumidero del *flowgraph*, que es el bloque llamado *UHD: USRP Sink*, el cual permite utilizar el hardware de radiofrecuencia, en este caso el USRP N210.

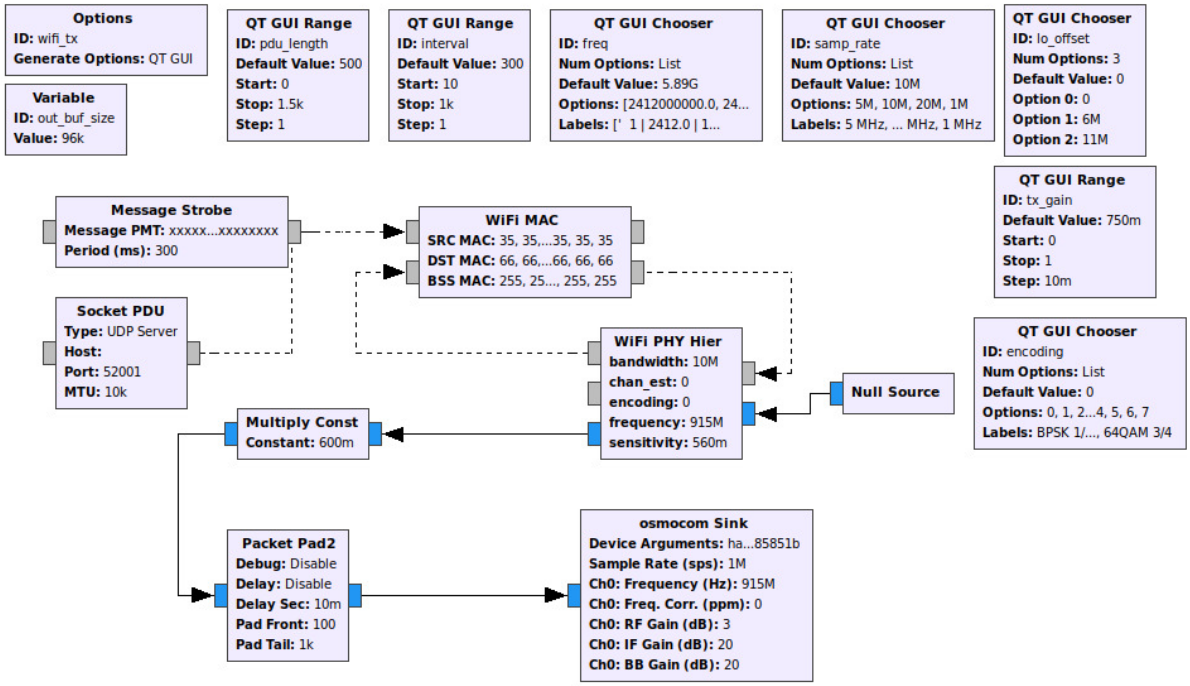

Figura 3.8: *Flowgraph* ejemplo para un transmisor del protocolo wifi (IEEE 802.11).

Además del bloque TUNTAP, es necesario agregar los bloques *Wireshark Connector* y *File Sink*, para realizar la captura de los paquetes antes de que sean transmitidos al medio.

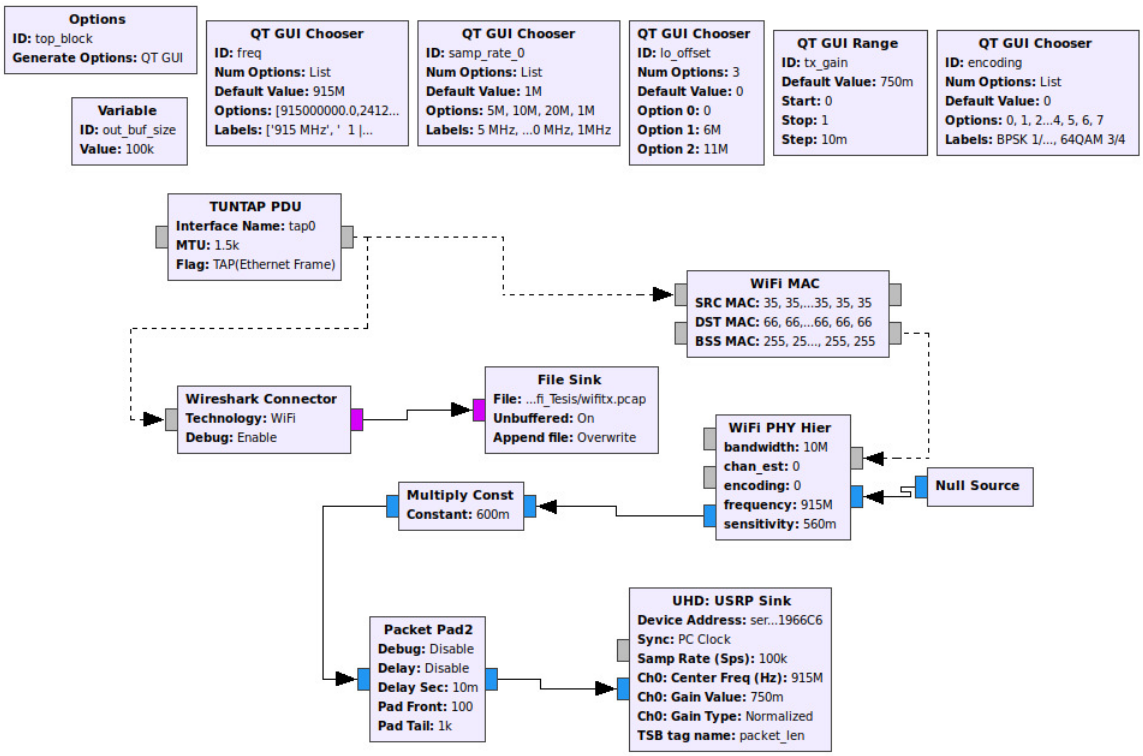

Figura 3.9: *Flowgraph* utilizado como transmisor del protocolo wifi (IEEE 802.11).

Cada bloque del flowgraph de la figura 3.9 realiza una tarea específica, comenzando la cadena de transmisión con el bloque *Wifi MAC*, el cual adiciona al mensaje proveniente de la TUNTAP las direcciones que corresponden a los distintos campos *Address* dentro de la trama MAC, con el objetivo de lograr una interoperabilidad con tarjetas 802.11 comerciales. Posteriormente, el mensaje se transfiere al bloque *Wifi PHY Hier*, el cual se encuentra definido como un flowgraph independiente que se puede observar en el apéndice C, por lo cual es denominado como bloque *jerárquico*. Finalmente, el paquete generado de este bloque es enviado al bloque *USRP Sink* donde se definen los parámetros del USRP N210, tales como la antena del transmisor, el reloj SDR y la razón de muestreo.

El bloque jerárquico *Wifi PHY Hier* (ver apéndice C) de manera general realiza las siguientes tareas. En primera instancia (grupo 1), se genera el encabezado del paquete, el mensaje se convierte en símbolos y se multiplexa. Posteriormente, se asigna la portadora de OFDM, se realiza la FFT y se agrega el prefijo cíclico, es decir, agrega el intervalo de guarda a cada símbolo del frame [23]. Mientras el grupo 2 corresponde al proceso de recepción. Este permite el desplazamiento de frecuencia y la estimación del desplazamiento del reloj de muestreo en el procesamiento de la señal mediante los bloques *Wifi Sync short* y *Wifi Sync long* [22].

Sin embargo, no se requiere de un USRP para simular el protocolo 802.11. Existe un flowgraph ejemplo que se encuentra en la carpeta del módulo *gr-ieee*-802-11 con nombre *wifi_loopback.grc*. Para este ejemplo, se agregaron los bloques TUNTAP, *Wireshark Connector* y *File Sink* como se observa en la figura 3.10, sustituyendo el bloque sumidero original *Message Strobe*.

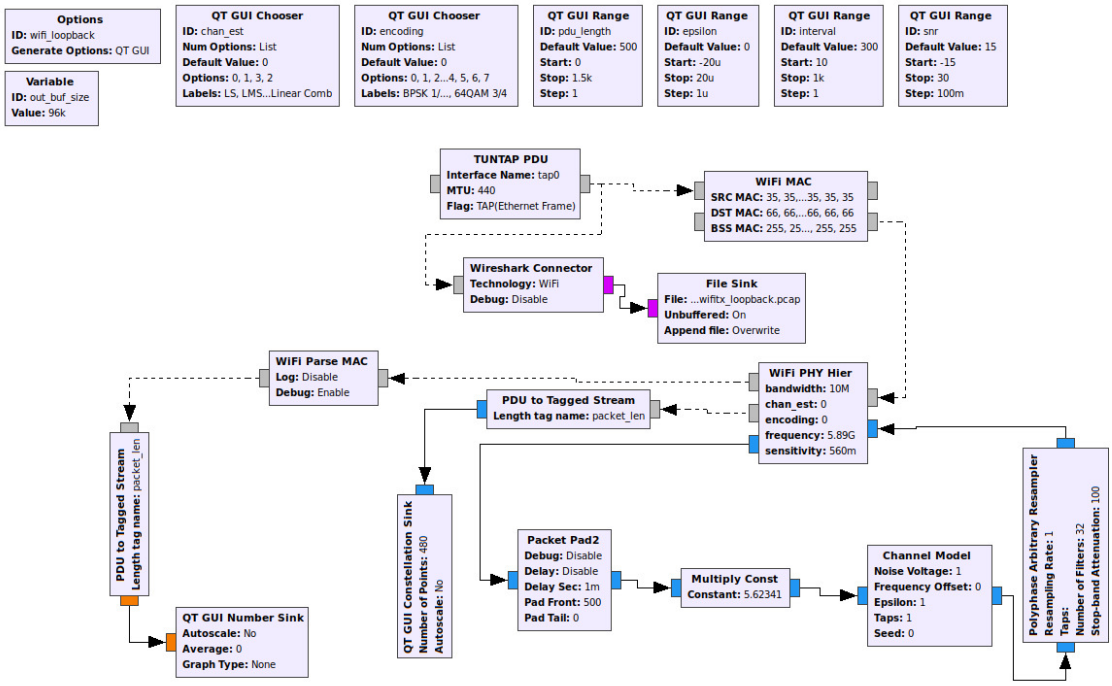

Figura 3.10: *Flowgraph* utilizado para la simulación del protocolo wifi (IEEE 802.11).

El flowgraph de la figura 3.10 corresponde a la simulación del protocolo, tanto a la transmisión como la recepción. Dado que en este ejemplo no se requiere el bloque USRP Sink. En su lugar se colocan los bloques para agregar ruido a la señal (Bloque *Channel Model*) y para muestrear la señal de manera arbitraria (bloque *Polyphase Arbitrary Resampler*) como si se estuviera transmitiendo por el medio. Posteriormente, se ocupa el grupo 2 del bloque jerárquico *Wifi PHY Hier*, para decodificar y mostrar las constelaciones que llegan al receptor dependiendo de la modulación elegida.

Finalmente, para terminar de describir cada uno de los flowgraphs que se utilizan para la experimentación de esta tesis, también se debe mencionar que en cada uno se tienen varios bloques *QT Range*, los cuales permiten seleccionar diferentes parámetros importantes para la transmisión como son ganancia o SNR (relación señal a ruido), así como bloques *QT Chooser*, que en este caso sirven para seleccionar la frecuencia de operación, frecuencia de muestreo, tipo de modulación, entre otras. Además cabe recalcar que para poder ejecutar cualquiera de los flowgraps descritos, primero se debe ejecutar el bloque *Wifi PHY Hier* y generar el archivo de extensión *py*, con el fin de evitar que exista error al ejecutar cualquiera de ellos, así como ajustar la memoria con la instrucción del cuadro 3.5, de acuerdo con lo recomendado en [11].

3.5: Comando para ajustar la memoria

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

3.5. USRP N210

El USRP elegido para la experimentación de este proyecto es el N210, el cual es una plataforma flexible de bajo costo en comparación con otros USRP, que opera con la ayuda de una computadora de propósito general, lo que facilita su programación y uso, ya que no se requiere de software o hardware extra especializado; además provee un ancho de banda amplio y alta capacidad de procesamiento. La arquitectura de este USRP, como se muestra en la figura 3.11 incluye una FPGA Xilinx Spartan® 3^a−DSP 3400, un ADC dual de 100 MS/s -

14 bits, DAC dual 400 Ms/s - 16 bit y conectividad *Gigabit Ethernet* para transportar el flujo de datos; además permite operar desde DC hasta 6 GHz [24].

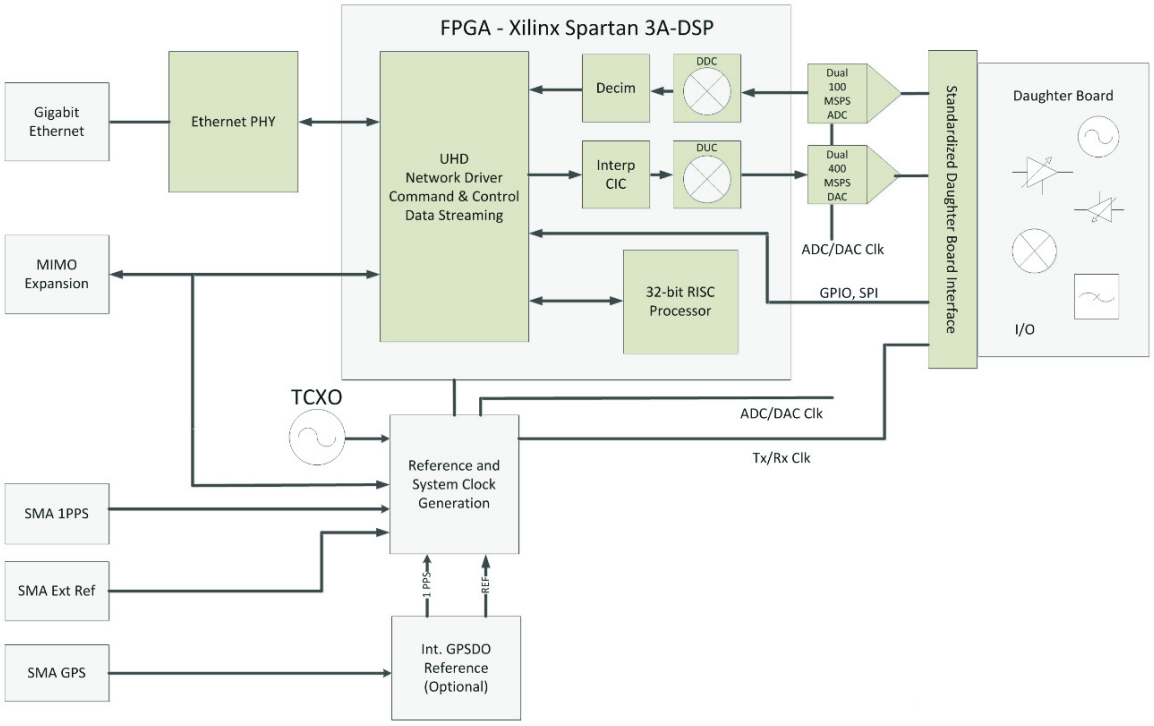

Figura 3.11: Arquitectura del USRP N210 [24].

Puede transportar un máximo de 50 MS/s desde y hacia la aplicación de software simultáneamente (modo full-dúplex), mediante el firmware *UHD*, el cual se encarga de establecer la configuración de la FPGA para direccionar el flujo de datos entre los módulos ADC y DAC hacia la interfaz *Gigabit Ethernet*. Si se implementan las funciones directamente en la FPGA, en lugar de utilizar una computadora personal, se pueden aumentar los recursos para el procesamiento. Por ejemplo, aumentar la frecuencia de muestreo y ejecutar tareas en paralelo mediante el uso de *VHDL*.

Otras características del USRP se muestran en la siguiente tabla:

Especificación	**Valor**
Potencia	
Alimentación DC	$6\,[V]$
Consumo de corriente	$1.3\,[A]$
Desempeño de conversión	
Razón de muestreo ADC	$100\,[MS/s]$
Resolución ADC	$14\,[bits]$
Razón de muestreo DAC	$400\,[MS/s]$
Resolución DAC	$16\,[bits]$
Física	
Temperatura operación	0 a $55\,[^oC]$
Dimensiones (l x w x h)	$22\,x\,16\,x\,5\,[cm]$
Peso	$1.2\,[kg]$

Tabla 3.1: Especificaciones del USRP N210 [24].

Para utilizarlo, es necesario comprobar la conexión del USRP a la computadora y verificar que existe conexión escribiendo la instrucción del cuadro 3.6 en la terminal de Ubuntu. Si el USRP tienen instaladas todas las bibliotecas para su uso como se indica en la sección 3.1, aparecerán sus principales características como se muestra en la figura 3.12.

3.6: Instrucción para verificar conexión de USRP.

```
$ uhd_find_devices
```

Figura 3.12: Características del USRP.

3.6. Generación de paquetes mediante Scapy

Scapy es una biblioteca de *Python* que le permite al usuario crear una gran variedad de paquetes para ser enviados dentro de una o varias redes. Esta capacidad de construcción de paquetes hace posible no solamente el envío de datos, sino que también pueden fabricarse paquetes para sondear, escanear o incluso atacar una red; teniendo la capacidad de decodificar y hasta falsificar paquetes de una gran cantidad de protocolos, enviarlos por cable, capturarlos, entre otras cosas. El principal objetivo de *Scapy* es tener un modelo flexible, donde los paquetes creados puedan tener las características que cada usuario le quiera agregar según sus intereses. Además esta biblioteca representa una simplicidad para el usuario debido a que para construir los paquetes solamente basta con ligar algunos comandos [18].

Para poder utilizar esta herramienta, es necesario tener instalado *Python* (preferentemente la versión 3). *Scapy* se instala con el comando que se muestra en el cuadro 3.7. Además, para saber que el módulo se instaló correctamente sobre *Python* y que se importaron todas las bibliotecas, se escribe en la terminal el comando *scapy* como se muestra en la figura 3.13.

3.7: Comando para instalar *Scapy*.

```
$ sudo apt install python-scapy
```

Figura 3.13: Instalación de Scapy correcta.

Scapy pueden generar paquetes de capa 2 (Ethernet) y capa 3 (IP), que son necesarios para la comunicación dentro de una red; así como añadir en el mismo paquete distintos tipos de protocolos que se necesiten [26].

Uno de los protocolo que se puede añadir es *internet control message protocol* (ICMP), el cual está diseñado para proporcionar retroalimentación sobre problemas en el entorno de comunicación [25]. ICMP sirve para diagnosticar problemas de capa 3 y el flujo de tráfico; éste tipo de mensajes se envían en muchas circunstancias para identificar un destino inalcanzable o control de congestión general en la red.

El código *Python* que se muestra a continuación genera un paquete de tipo ICMP y lo envía a la interfaz virtual TAP creada en el *kernel* del sistema operativo. Este paquete tiene diferentes campos, donde éstos pueden ser llenados de manera manual o en su defecto, Scapy los rellena de manera automática.

```
1  import socket
2  from scapy.all import Ether, IP, ICMP, srp1, sendp, UDP, Raw, RadioTap
3  import struct
4  import time
5  from hexdump import *
6
7  i = 0
8  while(1):
9          i = i+1
10         A = Ether()/ IP( src="127.0.0.1" , dst="192.168.20.1") /
           ↪  ICMP()/"HelloWorld"
11         A.show2() # rellena los campos que deben autogenerarse al enviarse
12         print "----------------------------------------------"
13         pkt = str(A)
14         print "----------------------------------------------"
15         hexdump(pkt)
16         print "----------------------------------------------"
17         print "Paquetes transmitidos: ", i
18         print "----------------------------------------------"
19         sendp(A, iface='tap0')
20         time.sleep(5)
```

El campo Ethernet consta principalmente de las direcciones MAC de origen y destino, que para este ejemplo Scapy rellenó de manera automática junto con el parámetro *type*, como se observa en el cuadro 3.8. El cuadro 3.9 muestra el encabezado IP, como es la dirección IP de destino y la fuente. Para esta tesis se utilizó la dirección *loopback* del sistema y la dirección de la interfaz TAP; los demás campos son necesarios para enviar la trama y *Scapy* los rellena por defecto. Además, como se utiliza el protocolo ICMP al crear el paquete, existe un campo dentro de la cabecera IP.

3.8: Parámetros de Ethernet para la creación de paquetes en Scapy.

```
###[  Ethernet  ]###
  dst        =  18:4a:6f:77:aa:7c
  src        =  00:f4:8d:90:ee:fd
  type       =  0x800
```

3.9: Parámetros de IP para la creación de paquetes en Scapy.

```
###[  IP  ]###
     version    =  4L
     ihl        =  5L
```

```
tos         =  0x0
len         =  38
flags       =
frag        =  0L
ttl         =  64
proto       =  icmp
chksum      =  0x272c
src         =  127.0.0.1
dst         =  192.168.20.1
```

Finalmente se crea el paquete ICMP, que aunque se define en capa IP, tiene sus propios parámetros en Scapy. En este caso, como se observa en el código anterior y en el cuadro 3.10, todos los parámetros del protocolo se generan por defecto. También se agrega una carga al paquete (cuadro 3.11) la cual representa los datos que vienen directamente del usuario desde capas superiores. De esta forma, se genera el paquete que se envía a la interfaz TAP mediante los comandos $iface = 'tap0'$ y *sendp()*.

3.10: Parámetros de ICMP para la creación de paquetes en Scapy.

```
###[  ICMP  ]###
          type       =  echo-request
          code       =  0
          chksum     =  0xf7ff
          id         =  0x0
          seq        =  0x0
```

3.11: Parámetros extra para la creación de paquetes en Scapy.

```
###[  Raw  ]###
          load       =  'HelloWorld'
```

3.7. Generación de paquetes del IEEE 802.11

Para la generación de tramas Wifi (IEEE 802.11), Scapy cuenta con un módulo llamado *RadioTap* y al igual que sucede con las funciones *Ethernet()* o *IP()* mencionadas en la sección 3.6, RadioTap puede realizar la modificación de diversos parámetros que contiene un paquete del estándar Wifi (ver cuadro 3.13). Por lo tanto, para mandar paquetes Wifi, se remplaza la línea 10 del código de la sección 3.6, por el código que se muestra en el cuadro 3.12.

3.12: Código para crear paquete de Wifi.

```
A = RadioTap()/Ether()/IP( src="127.0.0.1" , dst="192.168.20.1")/ \
    ICMP()/"ThisIsAMessage"
```

Por lo tanto el código para generar paquetes de wifi es el siguiente:

```
1 import socket
2 from scapy.all import Ether, IP, ICMP, srp1, sendp, UDP, Raw, RadioTap
3 import struct
4 import time
5 from hexdump import *
6
7 i = 0
```

```
8  while(1):
9          i = i+1
10         A = RadioTap()/Ether()/IP( src="127.0.0.1" , dst="192.168.20.1")/
           ↪  ICMP()/"ThisIsAMessage"
11         A.show2()
12         print "----------------------------------------------"
13         pkt = str(A)
14         print "----------------------------------------------"
15         hexdump(pkt)
16         print "----------------------------------------------"
17         print "Paquetes transmitidos: ", i
18         print "----------------------------------------------"
19         sendp(A, iface='tap0')
20         time.sleep(5)
```

Como se observa en el cuadro 3.13, en general el mensaje creado es el mismo, solo que ahora se agrega el encabezado Wifi, como direcciones MAC, tipo de paquetes e incluso se agrega por defecto un cifrado WEP.

3.13: Parámetros en la creación de paquetes 802.11.

```
###[ RadioTap dummy ]###
  version   = 0
  pad       = 0
  len       = 8
  present   =
  notdecoded= ''
###[ 802.11 ]###
     subtype   = 1L
     type      = Data
     proto     = 0L
     FCfield   = from-DS+retry+wep
     ID        = 28535
     addr1     = aa:7c:00:f4:8d:90
     addr2     = ee:fd:08:00:45:00
     addr3     = 00:2a:00:01:00:00
     SC        = 320
     addr4     = None
###[ 802.11 WEP packet ]###
        iv        = "'(\x7f"
        keyid     = 0
        wepdata   = '\x00\x01\xc0\xa8\x14\x01\x08\x00o)\
           \x00\x00\x00\x00ThisIsAMes'
        icv       = 1935763301
```

3.8. Resumen del capítulo

En este capítulo se presentan las herramientas para el desarrollo de esta tesis. En primer lugar, se indica la forma en que se instala GNU Radio para utilizarlo en conjunto con el USRP N210. Posteriormente, se describe la forma de crear y manipular una interfaz virtual TAP en Ubuntu y como acceder a ella por medio de su descriptor, así como la instalación y creación de paquetes de red ICMP y 802.11 en Scapy. Finalmente, se especifican las tareas que realizan los flowgraphs diseñados en GNU Radio del protocolo IEEE 802.11.

Capítulo 4

Experimentación

4.1. Paquetes en la interfaz TAP

Con la interfaz TAP levantada mediante los comandos del cuadro 3.2, se puede corroborar la escritura de paquetes mediante el código que se muestra en el Apéndice D, el cual abre un *socket* de tipo UDP y toma el descriptor de archivo de la TAP. La conexión UDP se establece en el puerto 4000. Para su ejecución es necesario utilizar el comando del cuadro 4.1. Además, como se observa en la figura 4.1, en un inicio (antes de ejecutar el programa) el número de paquetes recibidos en la TAP es igual a cero. Lo anterior se puede observar mediante el comando *ifconfig*.

4.1: Comandos para ejecución del archivo del apéndice D.

```
$ sudo python <nombre de archivo .py> <nombre de \
    interfaz> <direccion IP de interfaz>
```

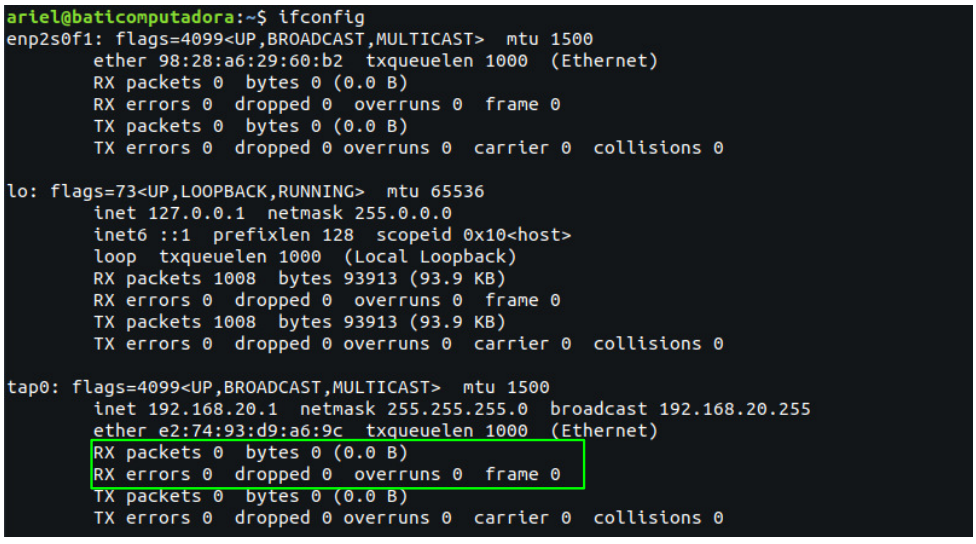

Figura 4.1: Número de paquetes recibidos en la TAP antes de enviar paquetes.

La figura 4.2 muestra los campos del paquete que se está enviando, desde los parámetros Ethernet, la capa IP y el tipo de protocolo de red que se utiliza, así como la carga de datos. Este mismo paquete también se muestra como una cadena hexadecimal con el propósito de compararlo con el analizador de paquetes *Wireshark*.

```
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 00:00:00:00:00:00
  type      = 0x800
###[ IP ]###
     version   = 4L
     ihl       = 5L
     tos       = 0x0
     len       = 38
     id        = 1
     flags     =
     frag      = 0L
     ttl       = 64
     proto     = icmp
     chksum    = 0x272c
     src       = 127.0.0.1
     dst       = 192.168.20.1
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xf7ff
        id        = 0x0
        seq       = 0x0
###[ Raw ]###
           load      = 'HelloWorld'

---------------------------------------------
---------------------------------------------
00000000: FF FF FF FF FF FF 00 00  00 00 00 00 08 00 45 00  ..............E.
00000010: 00 26 00 01 00 00 40 01  27 2C 7F 00 00 01 C0 A8  .&....@.',......
00000020: 14 01 08 00 F7 FF 00 00  00 00 48 65 6C 6C 6F 57  ..........HelloW
00000030: 6F 72 6C 64                                      orld
```

Figura 4.2: Formato de paquete generado en Scapy.

Se transmiten 17 paquetes ICMP a la interfaz TAP, como se observa en la figura 4.3. Sin embargo, se escriben más de 17 paquetes (ver figura 4.4), ya que el kernel genera paquetes del tipo ICMPv6 y *multicast domain name system* (MDNS). Es importante observar en la figura 4.4 que los paquetes generados en *Scapy* son exactamente iguales a los que llegan a la interfaz *tap0*.

```
---------------------------------------------
---------------------------------------------
00000000: FF FF FF FF FF FF 00 00  00 00 00 00 08 00 45 00  ..............E.
00000010: 00 26 00 01 00 00 40 01  27 2C 7F 00 00 01 C0 A8  .&....@.',......
00000020: 14 01 08 00 F7 FF 00 00  00 00 48 65 6C 6C 6F 57  ..........HelloW
00000030: 6F 72 6C 64                                      orld
---------------------------------------------
Paquetes transmitidos:  17
---------------------------------------------
Begin emission:
Finished to send 1 packets.

Received 0 packets, got 0 answers, remaining 1 packets
^Z
[1]+  Stopped                 sudo python tap-socket.py tap0 192.168.20.1
ariel@baticomputadora:~/Documents/Wifi_Tesis$
```

Figura 4.3: Paquetes transmitidos hacia la TAP.

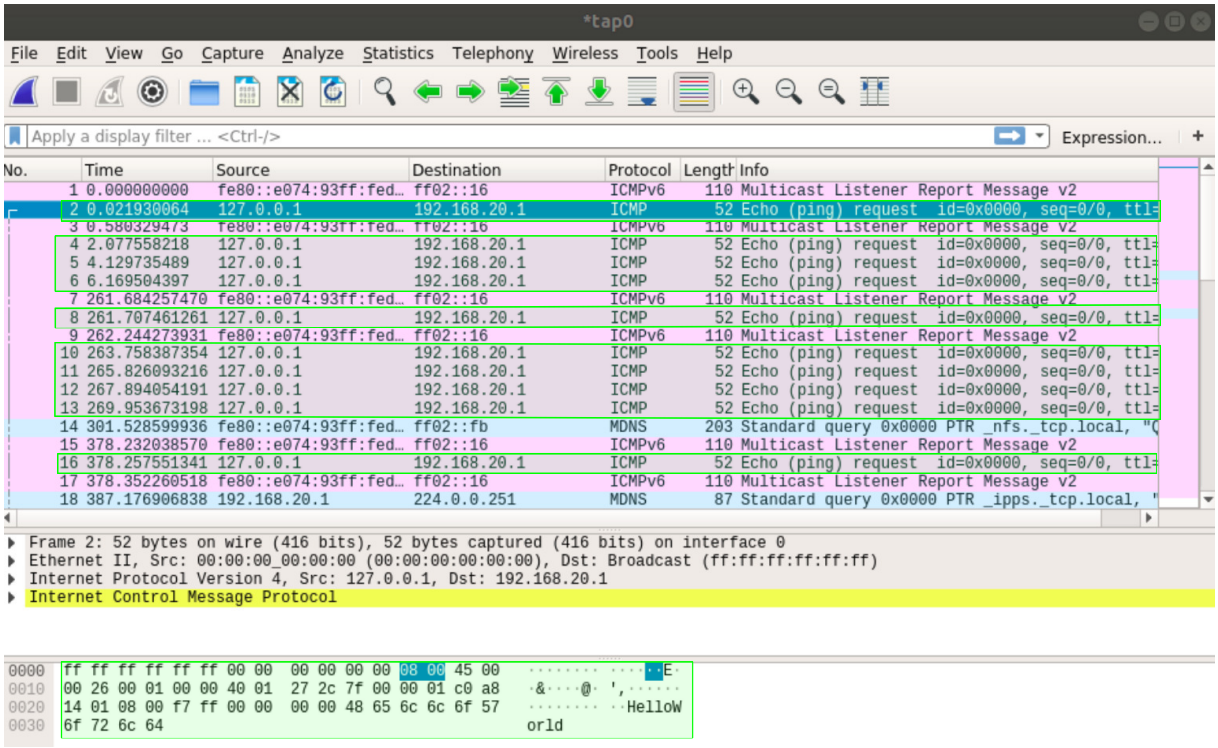

Figura 4.4: Paquetes recibidos en la interfaz TAP mostrados en Wireshark.

Una segunda evidencia de que los paquetes generados en *Scapy* son los mismos que se escriben en la interfaz TAP se puede encontrar al comparar las figuras 4.4 y 4.3, puesto que las cadenas hexadecimales que aparecen (recuadros verdes) son idénticas y muestran que el dato de mensaje dentro del frame que se envía y recibe es el texto *HelloWorld*. Finalmente, en la figura 4.5 se observa que el número de paquetes que se reciben en la interfaz TAP ha cambiado a 25.

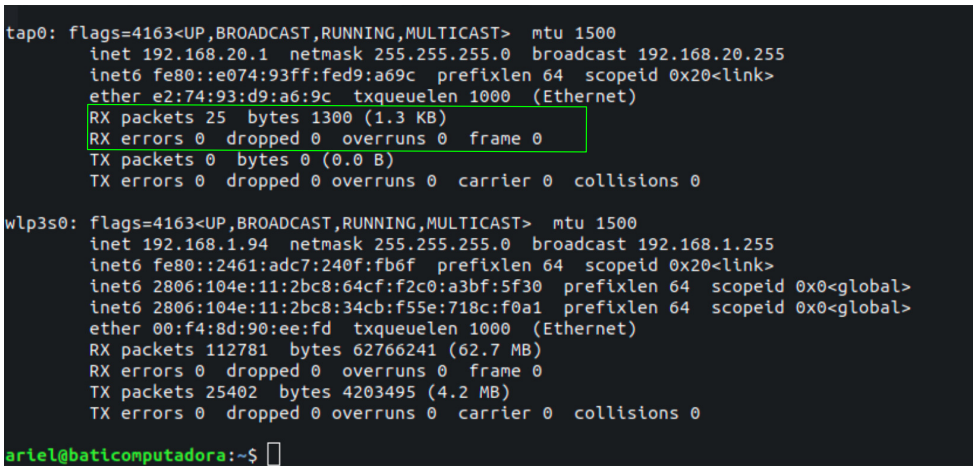

Figura 4.5: Paquetes recibidos en la interfaz TAP.

4.2. Implementación del modelo OSI simulando capa física

4.2.1. Paquetes ICMP

Para la simulación del protocolo Wifi IEEE 802.11, primero se debe restablecer la interfaz para que no tenga paquetes registrados, como se muestra en la figura 4.1 de la sección anterior. Esto se puede realizar de distintas formas, por ejemplo eliminando la interfaz con el comando del cuadro 4.2 y volviéndola a levantar con las instrucciones del cuadro 3.2 del capítulo anterior.

4.2: Instrucción para eliminar interfaz virtual TAP.

```
$ sudo ip link delete <nombre de la interfaz>
```

Posteriormente, se abre el *flowgraph* de la figura 3.10 y se ejecuta, además se ejecuta el código de la sección 3.6 con la instrucción del cuadro 4.3, para poder escribir mensajes en la TAP y que a su vez éstos puedan ser enviados a través de capa física por medio del protocolo Wifi.

4.3: Instrucción para ejecutar programa para generación de paquetes.

```
$ sudo python sendpacks.py
```

En este ejemplo se envían a la interfaz TAP un total de 22 paquetes generados en *Scapy* de longitud igual a 52 bytes con el mismo formato mencionado en la sección 3.6 y a través de un socket UDP perteneciente a la capa de transporte (ver figura 4.6). La figura 4.8 muestra tanto los paquetes enviados por Scapy como los paquetes generados por el kernel, teniendo entonces para este ejemplo, un total de 63 paquetes (ver figura 4.7).

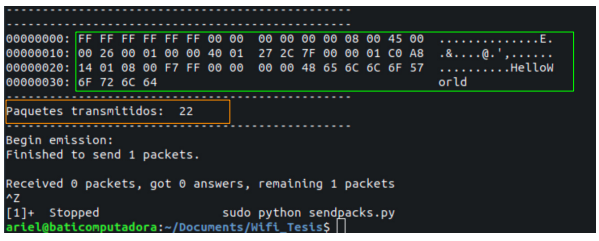

Figura 4.6: Paquetes enviados a la interfaz TAP y creados en *Scapy*.

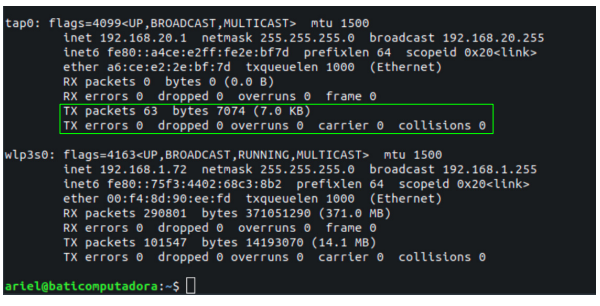

Figura 4.7: Paquetes transmitidos desde la interfaz TAP.

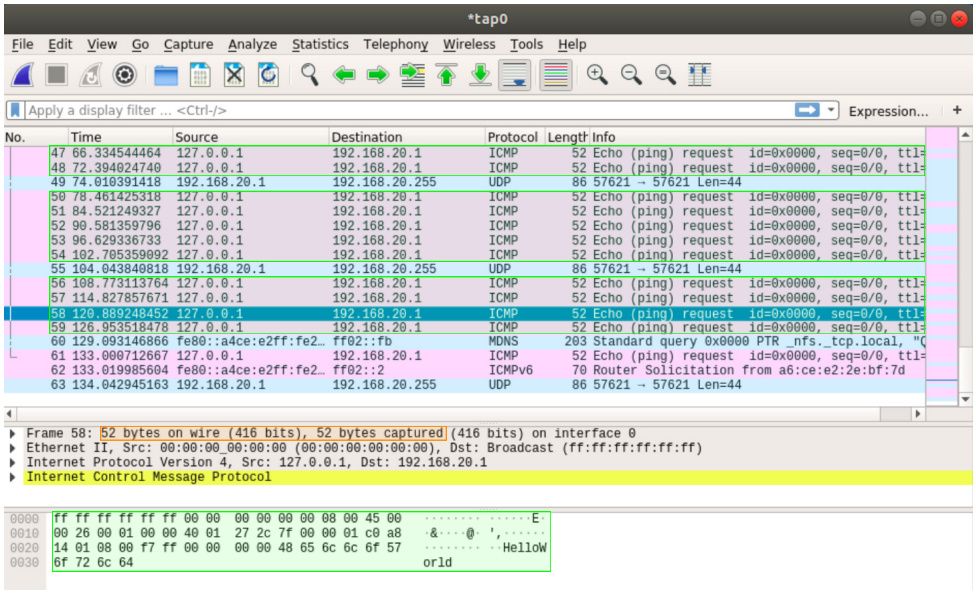

Figura 4.8: Paquetes recibidos en la interfaz TAP.

Finalmente, en lo referente a la capa física del sistema desarrollado en GNU Radio, el usuario tiene la capacidad de modificar entre otras cosas, el tipo de modulación con que se envían las tramas provenientes de la interfaz TAP, así como el SNR de las señales, lo que conlleva a tener una mejor recepción de la señal y menos pérdida de datos, sobre todo si se tiene un tipo de modulación como por ejemplo 16QAM, donde se envían más símbolos que en una modulación BPSK.

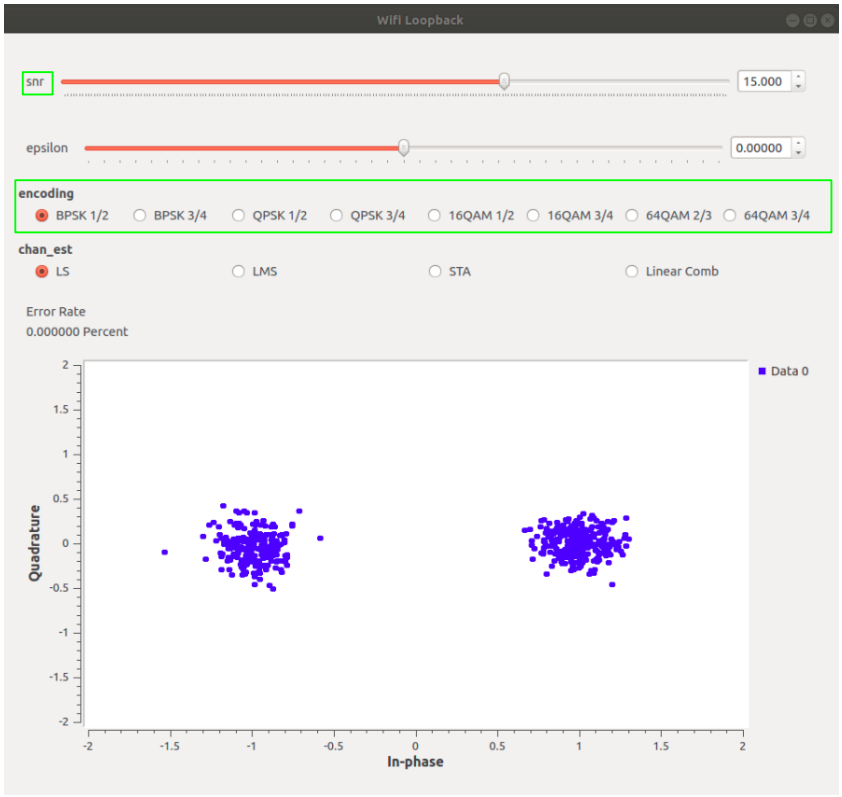

Figura 4.9: Modulación BPSK con que se reciben los mensajes que se envían de la interfaz TAP.

En las figuras 4.9 y 4.10 se puede observar la simulación de recepción de paquetes con modulación BPSK y 16QAM respectivamente, además de que en la figura 4.10 también se observa (de la terminal donde se ejecuta GNU Radio) el mensaje recibido decodificado y nos indica que el error de frame (fer) instantáneo es igual a 0, es decir, no hay error al recibir y decodificar los paquetes que se envían de la TAP y por ello el mensaje recibido que se muestra es igual al que se observa en la figura 4.8.

Figura 4.10: Modulación 16QAM con que se reciben los mensajes que se envían de la interfaz TAP y su correcta decodificación.

4.2.2. Paquetes IEEE 802.11

Para esta prueba, se crean paquetes 802.11 mediante el código de la sección 3.7, cuyos encabezados pertenecen a Wifi. Las figuras 4.11 y 4.12 muestran los campos del paquete, así como su mapeo a hexadecimal, respectivamente. Estos paquetes, al igual que los de secciones anteriores, se encuentran realizados como ICMP (con parámetros IP y Ethernet), pero con la diferencia de que se agregaron encabezados de Wifi, con distintos campos de banderas correspondientes al protocolo y hasta parámetros para encriptar y desencriptar dicho paquete. Además, se observa que para la prueba se envía un total de 10 paquetes con el mensaje *ThisIsAMessage*.

```
###[ RadioTap dummy ]###
  version   = 0
  pad       = 0
  len       = 8
  present   =
  notdecoded= ''
###[ 802.11 ]###
     subtype   = 15L
     type      = Reserved
     proto     = 3L
     FCfield   = to-DS+from-DS+MF+retry+pw-mgt+MD+wep+order
     ID        = 65535
     addr1     = ff:ff:00:00:00:00
     addr2     = 00:00:08:00:45:00
     addr3     = None
     SC        = 10752
     addr4     = None
###[ 802.11 WEP packet ]###
        iv        = '\x00\x01'
        keyid     = 0
        wepdata   = "@\x01'(\x7f\x00\x00\x01\xc0\xa8\x14\x01\x08\x00o)\x00\x00\x0
0\x00ThisIsAMes"
        icv       = 1935763301
```

Figura 4.11: Parámetros del paquete generado en Scapy del protocolo 802.11.

```
----------------------------------------------
----------------------------------------------
00000000: 00 00 08 00 00 00 00 00  FF FF FF FF FF FF 00 00  ................
00000010: 00 00 00 00 08 00 45 00  00 2A 00 01 00 00 40 01  ......E..*....@.
00000020: 27 28 7F 00 00 01 C0 A8  14 01 08 00 6F 29 00 00  '(..........o)..
00000030: 00 00 54 68 69 73 49 73  41 4D 65 73 73 61 67 65  ..ThisIsAMessage
----------------------------------------------
Paquetes transmitidos:  10
----------------------------------------------
.
Sent 1 packets.
^Z
[2]+  Stopped                 sudo python sendpacks.py
ariel@baticomputadora:~/Documents/Wifi_Tesis$
```

Figura 4.12: Representación en hexadecimal del paquete y total de paquetes enviados.

En la figura 4.14, se observan todos los paquetes que llegan a la interfaz TAP, aunque como ya se mencionó, muchos de ellos son generados por el kernel de Linux (35 paquetes). Estos paquetes llegan a GNU Radio para la simulación de su transmisión al medio y como se muestra en la figura 4.15 los paquetes enviados al medio son reconocidos por *Wireshark* como tramas del protocolo IEEE 802.11. Finalmente, se puede observar en la figura 4.13, que el número de paquetes transmitidos de la interfaz TAP es igual al total de paquetes en la figura 4.14, con lo que se confirma que todos los paquetes que recibe la TAP son enviados a capa física.

```
tap0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.20.1  netmask 255.255.255.0  broadcast 192.168.20.255
        inet6 fe80::cc78:caff:fe38:cc9  prefixlen 64  scopeid 0x20<link>
        ether ce:78:ca:38:0c:c9  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 45  bytes 5181 (5.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.72  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::75f3:4402:68c3:8b2  prefixlen 64  scopeid 0x20<link>
        ether 00:f4:8d:90:ee:fd  txqueuelen 1000  (Ethernet)
        RX packets 82799  bytes 104739706 (104.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 31740  bytes 4255594 (4.2 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ariel@baticomputadora:~$
```

Figura 4.13: Paquetes transmitidos de la interfaz TAP a capa física.

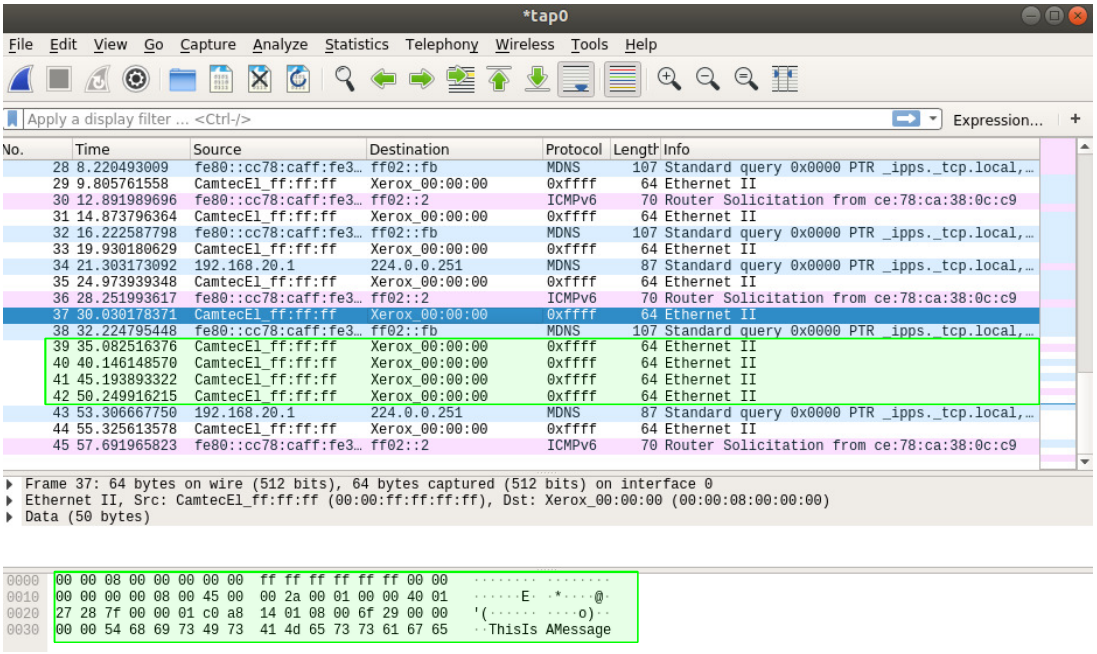

Figura 4.14: Paquetes que llegan a la interfaz TAP.

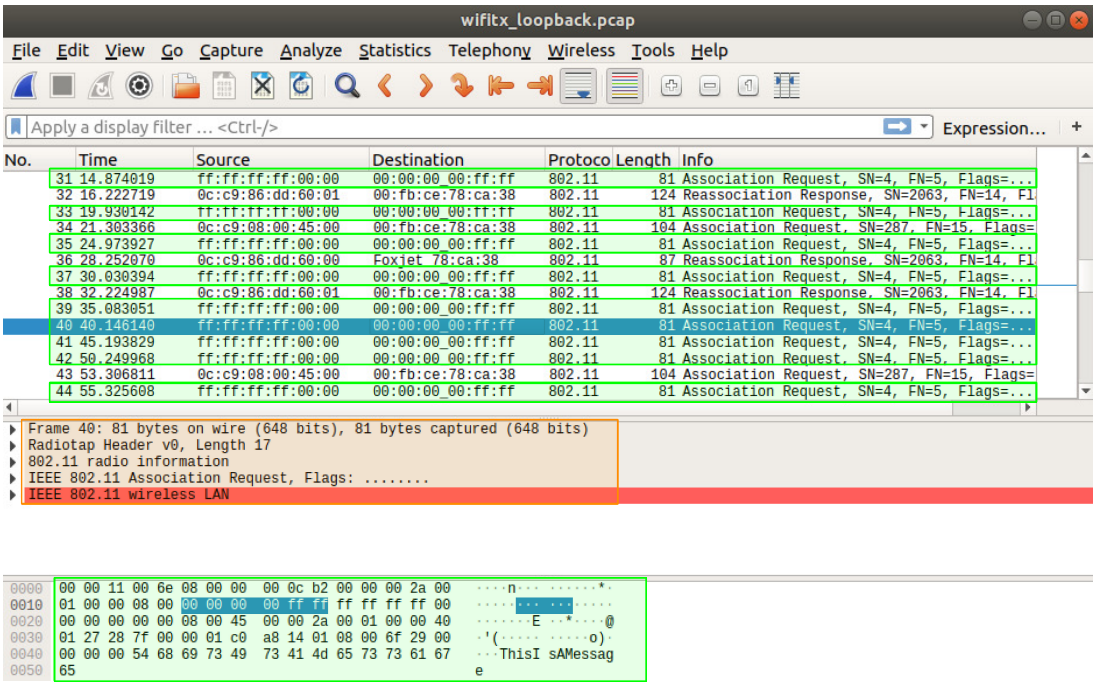

Figura 4.15: Paquetes que llegan a GNU Radio para la simulación de su transmisión.

Los detalles para la simulación de la capa física son idénticos a los mostrados en las figuras 4.9 y 4.10 de la sección anterior.

4.3. Implementación del modelo OSI utilizando USRP

Esta prueba utiliza el USRP N210 con el fin de transmitir paquetes de manera física. Una vez que se ha verificado que el USRP N210 está conectado (siguiendo las instrucciones de la sección 3.5), y que la interfaz TAP se restableció con las instrucciones de los cuadros 4.2 y 3.2, se abre en GNU Radio el *flowgraph* de la figura 3.9, así como el código en *Python* de la sección 3.7.

Para esta ejecución se enviaron a la interfaz TAP 10 paquetes de acuerdo con el contador del programa realizado en *Python* de la sección 3.7 como se observa en la figura 4.16. Aunque igual que en las secciones anteriores, no solamente se reciben estos paquetes en la interfaz, sino que se tienen los generados por el kernel; así que como se observa en la figura 4.18 en total se recibieron 43 paquetes, de los cuales 10 fueron generados por *Scapy* y tienen una longitud de 64 bytes con el mensaje *ThisIsAMessage* y los parámetros definidos en la sección 3.7.

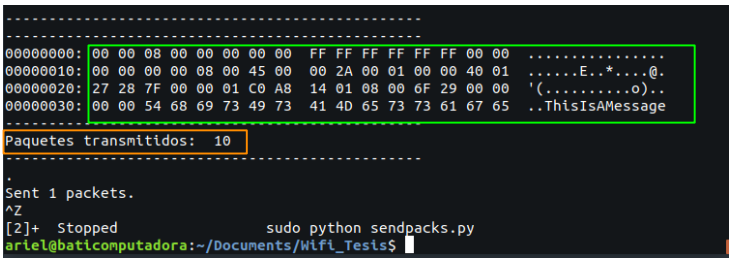

Figura 4.16: Paquetes enviados a la interfaz TAP, creados en *Scapy*.

Para corroborar el total de paquetes en la interfaz TAP, se utiliza el comando *ifconfig* que se muestra en la figura 4.17, donde se puede ver que el número de paquetes transmitidos en la interfaz al final de la ejecución coincide con el número de paquetes de las figuras 4.18 y 4.19 que en primera instancia ingresan a la TAP y posteriormente a la capa física para ser enviados utilizando el USRP.

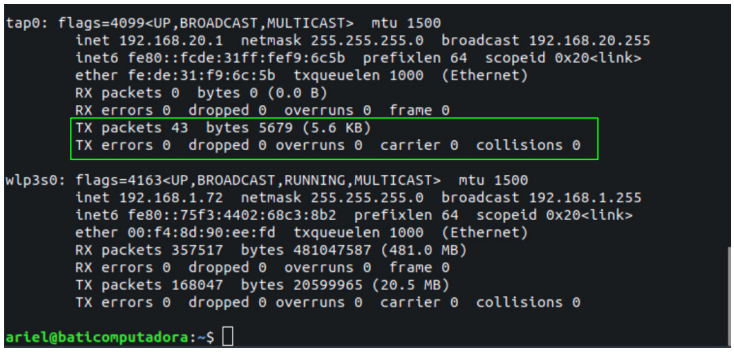

Figura 4.17: Paquetes transmitidos desde la interfaz TAP.

Al igual que ocurre en la implementación con la capa física simulada, en este caso, el *flowgraph* de la figura 3.9 cuenta con el bloque *File Sink* que genera un archivo de extensión *pcap* para mostrar los paquetes que se envían a la capa física desde la interfaz TAP, los cuales son identificados como paquetes del protocolo Wifi, con longitud de 81 bytes (ver figura 4.19). Y una vez más es importante observar en las figuras 4.16, 4.18 y 4.19, que los mensajes generados por *Scapy* son los mismos que llegan a GNU Radio para ser enviados por medio del USRP N210.

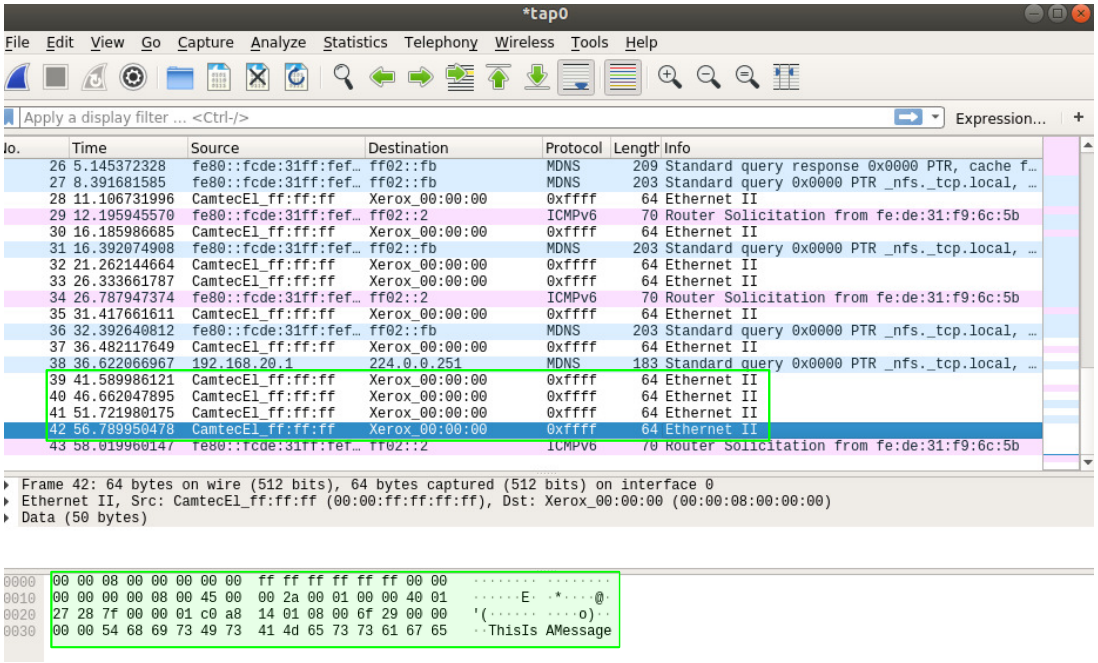

Figura 4.18: Paquetes recibidos en la interfaz TAP.

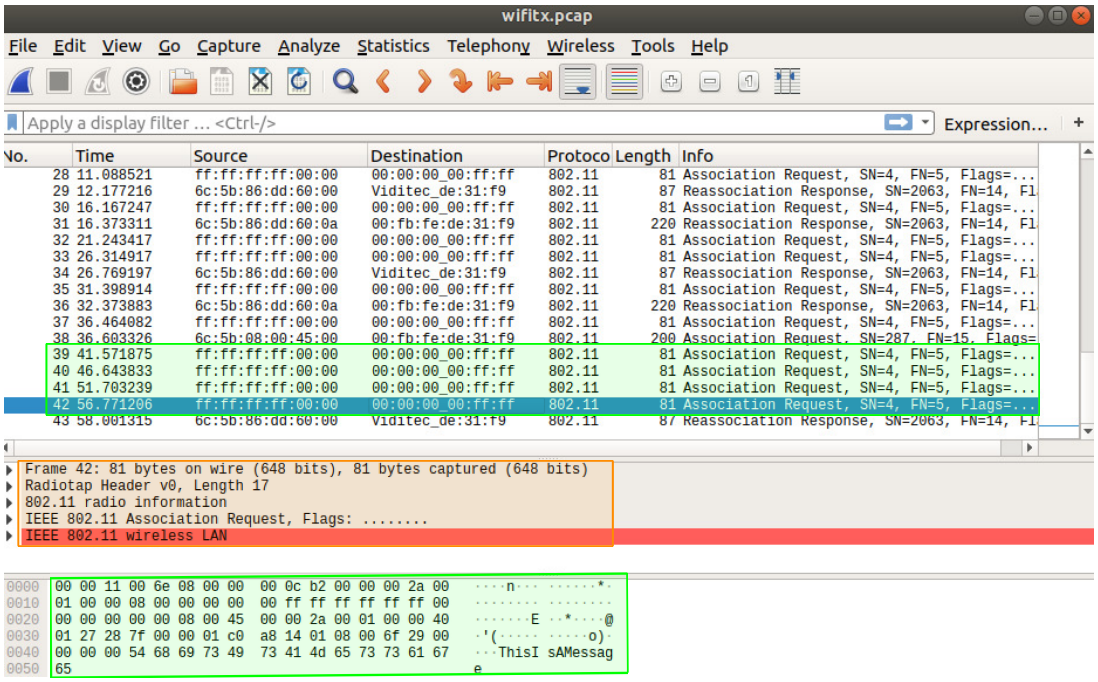

Figura 4.19: Paquetes que llegan a GNU Radio para ser enviados con el USRP.

Respecto a la capa física para este caso en general, no se utiliza un receptor para tener un sistema completo, solamente se realiza el proceso hasta la transmisión, debido a que no es un objetivo en esta tesis y puesto que el proceso de transmisión y recepción de datos utilizando los mismos *flowgraphs* de GNU Radio, ya han sido probados por ejemplo en [11] o en [28], donde sí se abordan parámetros exclusivos de capa física para el protocolo de Wifi.

En este transmisor, como se observa en la figura 4.20, se pueden variar diversos paráme-

tros como la ganancia de transmisión, la frecuencia de trabajo (que en este caso fue la banda libre de 915 MHz, aunque se pueden elegir bandas propias de Wifi en sus diferentes versiones), el tipo de modulación y la frecuencia de muestreo, entre otros. Estos parámetros automáticamente modifican el USRP para realizar el proceso de RF necesario para la transmisión de datos.

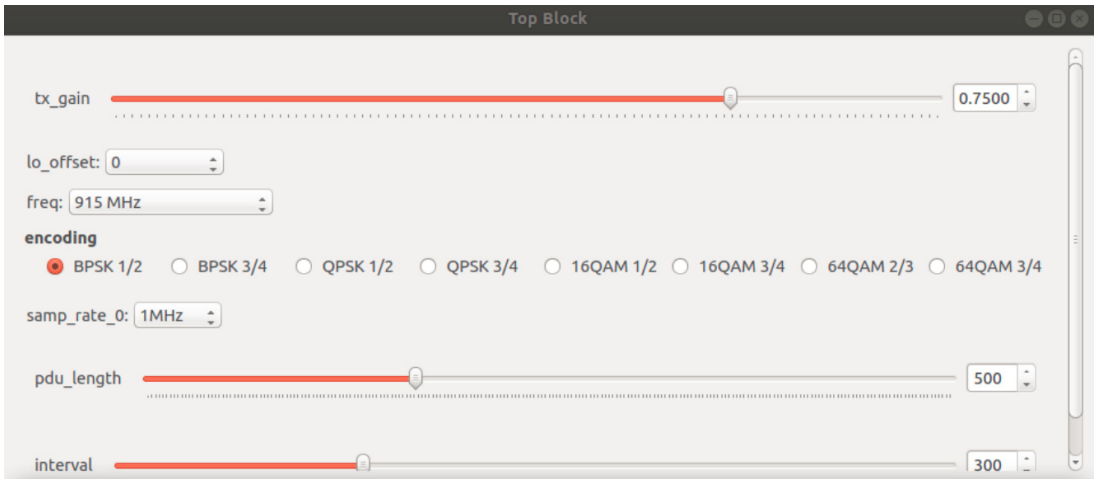

Figura 4.20: Parámetros para la transmisión de datos con el USRP N210.

4.4. Resumen del capítulo

Este capítulo divide las pruebas en dos fases. Primero, se simula un medio de transmisión y posteriormente se utiliza un USRP. Para las pruebas simuladas se envían paquetes ICMP generados en *Scapy* con el mensaje *HelloWorld*. Posteriormente se envían paquetes IEEE 802.11. En ambas pruebas se utiliza el software *Wireshark* y se observa que los paquetes se transmiten hacia el medio empleando diferentes tipos de modulación y éstos son recibidos con el mismo formato original con el que fueron creados. Finalmente, se realiza las mismas pruebas, pero en esta ocasión se transmiten paquetes 802.11 al medio a través del USRP N210, donde se puede modificar parámetros de RF como la ganancia y frecuencia de transmisión.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

En los últimos años, los radios definidos por software han tomado una relevancia muy importante en el área de investigación, ya que han permitido el estudio de protocolos de capa física y enlace de datos que antes no se tenía acceso; ya que estas capas solían ser implementadas en hardware propietario. Más aún, el uso de tecnologías como los USRP y GNU Radio han permitido analizar características de capa física que antes era muy difícil replicar. Esta tesis hace uso de estas tecnologías para unir la capa física modelada en GNU Radio con las capas superiores del modelo OSI mediante herramientas propias de los sistemas UNIX como son las interfaces virtuales para conformar un sistema que contenga todas las capas del modelo OSI en software. Esto con el propósito de tener un laboratorio para analizar toda la pila de protocolos existentes y crear nuevos protocolos.

Para lograr este objetivo, se desarrollaron diversas tareas que aunque parecieran simples en realidad no lo son debido a que la mayoría de las herramientas utilizadas como son GNU Radio, USRP y la misma interfaz virtual TAP, no existe información clara ni precisa de cómo están construidos internamente. Inclusive, las páginas oficiales carecen de información concreta para la modificación de estas herramientas. Por lo que la escritura de esta tesis se desarrolló como un manual que enseñe a otros usuarios a utilizar estas herramientas, y especialmente a modificarlas, aunque no fuera un objetivo planteado inicialmente.

Entre las tareas destacan la creación y estudio del protocolo IEEE 802.11 implementado en GNU Radio. También se realizaron estudios en cuanto a la creación de paquetes en Scapy y se buscó que fuera lo más simple posible con propósitos ilustrativos. Finalmente, se destaca la creación y manipulación de la interfaz virtual TAP en Ubuntu, así como el manejo de los descriptores para poder escribir paquetes en la interfaz por medio de sockets.

Las pruebas mostradas en el Capítulo 4 indican que es posible escribir paquetes en la interfaz TAP, con los mismos parámetros con los que se crearon en Scapy y éstos a su vez pueden ser tomados de la interfaz para que sean procesados en GNU Radio y puedan ser enviados al medio mediante el USRP. Para esta tesis, se crearon solo paquetes 802.11 e ICMP. Sin embargo, cualquier tipo de paquete puede ser creado. Esto muestra la flexibilidad del sistema implementado y la capacidad para configurar y mezclar diferentes tipos de protocolos en todos los niveles del modelo OSI.

5.2. Verificación de la hipótesis

Retomando la hipótesis que se presenta en la sección 1.2:

"*La creación de una interfaz virtual TUN-TAP como un módulo programable en GNU Radio permite conectar todas las capas del modelo OSI a través de software, eliminando el uso de hardware propietario.*"

Para verificar la validez de la hipótesis, la experimentación se dividió en 2 fases. En la primera, se generaron paquetes a través de Scapy con carga útil como si se tratara de un mensaje que viene de capa de aplicación, presentación y sesión del modelo OSI. También se agrego el encabezado de capa IP. De esta forma, por medio del protocolo UDP de la capa de transporte son enviados a la interfaz TAP. En las figuras 4.3 y 4.4 se puede observa que los paquetes que son recibidos en la TAP, son iguales a los generados en Scapy (figura 4.2). Por lo anterior, se demuestra que se tienen conectadas las capas 3, 4, 5, 6 y 7 del modelo OSI.

Para la segunda fase, se modificó el bloque TUN-TAP (ver Sección 3.3), lo cual permitió que se uniera la interfaz virtual TAP con la capa física desarrollada en GNU Radio. De esta manera, se unen todas las capas del modelo OSI. Como se muestra en las figuras 4.8 y 4.7, donde los paquetes que llegan a la interfaz TAP, son los mismos que son enviados al medio con el simulador de GNU Radio (ver figura 3.10). O también se puede corroborar en las figuras 4.18 y 4.19, que los paquetes 802.11 son enviados por el USRP N210 (ver Sección 4.3). Por lo que la hipótesis se cumple.

5.3. Trabajo futuro

En la sección 4.3, se utilizó el USRP N210 como transmisor de paquetes al medio. Se mostró que los paquetes enviados pertenecen al protocolo 802.11 y que Wireshark es capaz de leer e identificarlos todos los campos del encabezado. Sin embargo, solo se experimentó con el transmisor de dicho protocolo, por lo que en un futuro se planea implementar un receptor en GNU Radio para poder tener toda la comunicación perteneciente a este protocolo.

Además, se planea implementar una capa MAC que escuche la portadora de la señal y utilice una técnica de prevención de colisiones como *exponential backoff*, ya que actualmente el modelo tiene una capa MAC ALOHA.

Apéndice A

Dependencias para instalación de GNU Radio

Antes de instalar GNU Radio es necesario verificar que se tengan las siguientes bibliotecas instaladas en Ubuntu 18.04.

A.1: Dependencias para instalar GNU Radio.

```
sudo apt-get -y install git swig cmake doxygen build-essential \
   libboost-all-dev libtool libusb-1.0-0 libusb-1.0-0-dev libudev-dev\
    libncurses5-dev libfftw3-bin libfftw3-dev libfftw3-doc libcppunit\
   -1.14-0 libcppunit-dev libcppunit-doc ncurses-bin cpufrequtils \
   python-numpy python-numpy-doc python-numpy-dbg python-scipy python\
   -docutils qt4-bin-dbg qt4-default qt4-doc libqt4-dev libqt4-dev-\
   bin python-qt4 python-qt4-dbg python-qt4-dev python-qt4-doc python\
   -qt4-doc libqwt6abi1 libfftw3-bin libfftw3-dev libfftw3-doc \
   ncurses-bin libncurses5 libncurses5-dev libncurses5-dbg \
   libfontconfig1-dev libxrender-dev libpulse-dev swig g++ automake \
   autoconf libtool python-dev libfftw3-dev libcppunit-dev libboost-\
   all-dev libusb-dev libusb-1.0-0-dev fort77 libsdl1.2-dev python-\
   wxgtk3.0 git libqt4-dev python-numpy ccache python-opengl libgsl-\
   dev python-cheetah python-mako python-lxml doxygen qt4-default qt4\
   -dev-tools libusb-1.0-0-dev libqwtplot3d-qt5-dev pyqt4-dev-tools \
   python-qwt5-qt4 cmake git wget libxi-dev gtk2-engines-pixbuf r-\
   base-dev python-tk liborc-0.4-0 liborc-0.4-dev libasound2-dev \
   python-gtk2 libzmq3-dev libzmq5 python-requests python-sphinx \
   libcomedi-dev python-zmq libqwt-dev libqwt6abi1 python-six libgps-\
   dev libgps23 gpsd gpsd-clients python-gps python-setuptools
```

Apéndice B

Código en C++ del bloque TUNTAP en GNU Radio.

```
    /* -*- c++ -*- */
/*
 * Copyright 2013 Free Software Foundation, Inc.
 *
 * This file is part of GNU Radio
 *
 * GNU Radio is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 3, or (at your option)
 * any later version.
 *
 * GNU Radio is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GNU Radio; see the file COPYING.  If not, write to
 * the Free Software Foundation, Inc., 51 Franklin Street,
 * Boston, MA 02110-1301, USA.
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include "tuntap_pdu_impl.h"
#include <gnuradio/io_signature.h>
#include <gnuradio/blocks/pdu.h>
#include <boost/format.hpp>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/sendfile.h>
#include <netinet/in.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
39 #include <fcntl.h>
40 #include <netdb.h>
41 #include <thread>
42 #include <pthread.h>
43 #include <iostream>
44 #define BUFSIZE 1440
45 #define MAX 1546
46
47 #if (defined(linux) || defined(__linux) || defined(__linux__))
48 #include <sys/ioctl.h>
49 #include <arpa/inet.h>
50 #include <linux/if.h>
51 #endif
52
53 namespace gr {
54   namespace blocks {
55
56     tuntap_pdu::sptr
57     tuntap_pdu::make(std::string dev, int MTU, bool istunflag)
58     {
59 #if (defined(linux) || defined(__linux) || defined(__linux__))
60       return gnuradio::get_initial_sptr(new tuntap_pdu_impl(dev, MTU,
        ↪  istunflag));
61 #else
62       throw std::runtime_error("tuntap_pdu not implemented on this
        ↪  platform");
63 #endif
64     }
65
66 #if (defined(linux) || defined(__linux) || defined(__linux__))
67     tuntap_pdu_impl::tuntap_pdu_impl(std::string dev, int MTU, bool
      ↪  istunflag)
68       :        block("tuntap_pdu",
69                  io_signature::make (0, 0, 0),
70                  io_signature::make (0, 0, 0)),
71         stream_pdu_base(istunflag ? MTU : MTU + 14),
72         d_dev(dev),
73         d_istunflag(istunflag)
74     {
75       // make the tuntap
76       char dev_cstr[1024];
77       memset(dev_cstr, 0x00, 1024);
78       strncpy(dev_cstr, dev.c_str(), std::min(sizeof(dev_cstr),
        ↪  dev.size()));
79
80       bool istun = d_istunflag;
81       if(istun){
82         d_fd = tun_alloc(dev_cstr, (IFF_TUN | IFF_NO_PI));
83       } else {
84         d_fd = tun_alloc(dev_cstr, (IFF_TAP | IFF_NO_PI));
85       }
86
87       if (d_fd <= 0)
88         throw std::runtime_error("gr::tuntap_pdu::make: tun_alloc failed
          ↪  (are you running as root?)");
```

```
89
90      int err = set_mtu(dev_cstr, MTU);
91      if(err < 0)
92        std::cerr << boost::format(
93          "gr::tuntap_pdu: failed to set MTU to %d.\n"
94          "You should use ifconfig to set the MTU. E.g.,\n"
95          "  $ sudo ifconfig %s mtu %d\n"
96          ) % MTU % dev % MTU << std::endl;
97
98      std::cout << boost::format(
99        "Allocated virtual ethernet interface: %s\n"
100       "You must now use ifconfig to set its IP address. E.g.,\n"
101       "  $ sudo ifconfig %s 192.168.200.1\n"
102       "Be sure to use a different address in the same subnet for each
          ↪  machine.\n"
103       ) % dev % dev << std::endl;
104
105     // set up output message port
106     message_port_register_out(pdu::pdu_port_id());
107     start_rxthread(this, pdu::pdu_port_id());
108
109     // set up input message port
110     message_port_register_in(pdu::pdu_port_id());
111     set_msg_handler(pdu::pdu_port_id(),
          ↪  boost::bind(&tuntap_pdu_impl::send, this, _1));
112   }
113
114   int
115   tuntap_pdu_impl::tun_alloc(char *dev, int flags)
116   {
117     struct ifreq ifr;
118     int fd, err;
119     const char *clonedev = "/dev/net/tun";
120
121     /* Arguments taken by the function:
122      *
123      * char *dev: the name of an interface (or '\0'). MUST have enough
124      *   space to hold the interface name if '\0' is passed
125      * int flags: interface flags (eg, IFF_TUN etc.)
126      */
127
128     /* open the clone device */
129     if ((fd = open(clonedev, O_RDWR)) < 0)
130       return fd;
131
132     /* preparation of the struct ifr, of type "struct ifreq" */
133     memset(&ifr, 0, sizeof(ifr));
134
135     ifr.ifr_flags = flags;   /* IFF_TUN or IFF_TAP, plus maybe IFF_NO_PI
          ↪  */
136
137
138     /* if a device name was specified, put it in the structure;
          ↪  otherwise,
139      * the kernel will try to allocate the "next" device of the
```

```
140         * specified type
141         */
142       if (*dev)
143         strncpy(ifr.ifr_name, dev, IFNAMSIZ - 1);
144 
145       /* try to create the device */
146       if ((err = ioctl(fd, TUNSETIFF, (void *) &ifr)) < 0) {
147         close(fd);
148         return err;
149       }
150 
151       /* if the operation was successful, write back the name of the
152        * interface to the variable "dev", so the caller can know
153        * it. Note that the caller MUST reserve space in *dev (see calling
154        * code below)
155        */
156       strcpy(dev, ifr.ifr_name);
157 
158       /* this is the special file descriptor that the caller will use to
        ↪  talk
159        * with the virtual interface
160        */
161       return fd;
162     }
163 
164     int
165     tuntap_pdu_impl::set_mtu(const char *dev, int MTU)
166     {
167       struct ifreq ifr;
168       int sfd, err;
169 
170       /* MTU must be set by passing a socket fd to ioctl;
171        * create an arbitrary socket for this purpose
172        */
173       if ((sfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
174           return sfd;
175 
176       /* preparation of the struct ifr, of type "struct ifreq" */
177       memset(&ifr, 0, sizeof(ifr));
178       strncpy(ifr.ifr_name, dev, IFNAMSIZ);
179       ifr.ifr_addr.sa_family = AF_INET; /* address family */
180       ifr.ifr_mtu = MTU;
181 
182       /* try to set MTU */
183       if ((err = ioctl(sfd, SIOCSIFMTU, (void *) &ifr)) < 0)
184         return err;
185 
186       return MTU;
187     }
188 #endif
189 
190   } /* namespace blocks */
191 }/* namespace gr */
```

Apéndice C

Bloque jerárquico para el funcionamiento de capa física en GNU Radio

En la siguiente figura se muestra el *flowgraph* del bloque *Wifi PHY Hier* dividido en los 2 grupos principales de funcionamiento.

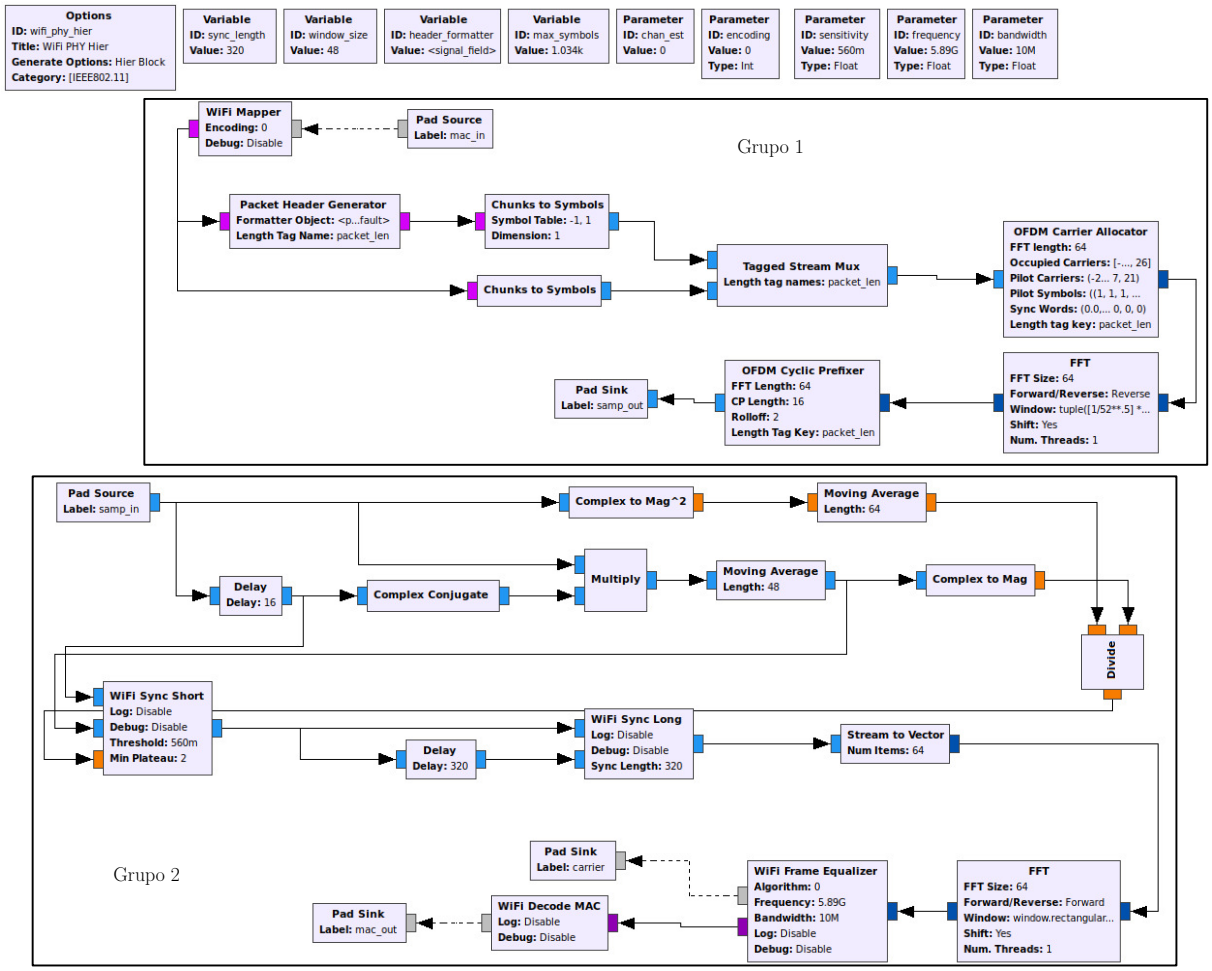

Figura C.1: *Flowgraph* jerárquico del protocolo Wifi (IEEE 802.11).

Apéndice D

Programa en Python para corroborar entrada de paquetes a la TAP

```
import fcntl
import struct
import os
import socket
import threading
import sys
import time
from scapy.all import Ether, IP, ICMP, srp1, sendp, UDP, Raw, RadioTap
from hexdump import *

TUNSETIFF = 0x400454ca
TUNSETOWNER = TUNSETIFF + 2
IFF_TUN = 0x0001
IFF_TAP = 0x0002
IFF_NO_PI = 0x1000

def recv():
     print "recibiendo"
     ss = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
     ss.bind(("192.168.20.1", 4000))
     while True:
         data, addr = ss.recvfrom(1526)
         print data
         os.write(ntun.fileno(), data)

if __name__ == "__main__":

    if len(sys.argv) < 3:
        print "Usage: tap-linux.py <tap_interface>
        ↪  <dst_address_of_tunnel>"
        sys.exit(1)
    iface = sys.argv[1]
    dst = sys.argv[2]
    print "Working on %s inteface, destination address %s:4000 udp" %
    ↪  (iface, dst)
```

```
34      tun = open('/dev/net/tun', 'r+b')
35      ifr = struct.pack('16sH', iface, IFF_TAP | IFF_NO_PI)
36      fcntl.ioctl(tun, TUNSETIFF, ifr)
37      fcntl.ioctl(tun, TUNSETOWNER, 1000)
38      t = threading.Thread(target=recv)
39      i = 0
40      try:
41          t.start()
42          while True:
43              i = i+1
44              A = Ether()/ IP( src="127.0.0.1" , dst="192.168.20.1") /
                ↪  ICMP()/"HelloWorld"
45              A.show2() # rellena los campos que deben autogenerarse al
                ↪  enviarse
46              print "----------------------------------------------------"
47              pkt = str(A)
48              print "----------------------------------------------------"
49              hexdump(pkt)
50              print "----------------------------------------------------"
51              print "Paquetes transmitidos: ", i
52              print "----------------------------------------------------"
53              sendp(A, iface='tap0')
54              os.write(tun.fileno(), pkt)
55              time.sleep(1)
56
57
58      except KeyboardInterrupt:
59          print "Terminating ..."
60          os._exit(0)
```

Bibliografía

[1] Barceló Miquel. Una historia de la informática. Editorial UOC. 2008

[2] Li Yadong, Cui Wenqiang, et al. Research based on OSI model. Northeastern University. Shenyang, China. IEEE, 2011.

[3] Global Knowledge. The OSI Model: Understanding the Seven Layers of Computer Networks. 2006

[4] Definitions of Software Defined Radio (SDR) and Cognitive Radio System (CRS). ITU-R. 2009. URL: https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-SM.2152-2009-PDF-E.pdf

[5] Mitola J. Software Radio: Wireless Architecture for the 21st Century. Mitola's STATISfaction, ISBN 0-9671233-0-5

[6] HPSDR. Recuperado de: https://openhpsdr.org/

[7] WebSDR. Recuperado de: http://www.websdr.org/

[8] GNU Radio official website. Recuperado de: https://www.gnuradio.org/

[9] Risset Tanguy. Introduction to GNU Radio. City Laboratory, INSA de Lyon. 2016. Recuperado de: https://gnuradio-fr-18.sciencesconf.org/data/pages/gnuRadioArchitecture.pdf

[10] Lei Zhang. Implementation of wireless communication based on software defined radio. Universidad Politécnica de Madrid. Julio de 2013.

[11] B. Bloessl, M. Segata, C. Sommer, and F. Dressler. An IEEE802.11a/g/p OFDM Receiver for GNU Radio. inACM SIGCOMM2013, 2nd ACM SIGCOMM Workshop of Software Radio ImplementationForum (SRIF 2013).Hong Kong, China: ACM, Agosto. 2013, pp.9–16.

[12] Teixeira F, Santos J, et al. Evaluation of Underwater IEEE 802.11 Networks at VHF and UHF Frequency Bands using Software Defined Radios. ACM. 10ma Conferencia de sistemas y redes bajo el agua. Washington DC. Octubre de 2015.

[13] Bloessl B, Klingler F, et al. A Systematic Study on the Impact of Noise and OFDM Interference on IEEE 802.11p. Conferencia de redes vehiculares de la IEEE. 2017.

[14] Zhang T, SOng T, Chen D, et al. WiGrus: A wifi based gesture recognition system using SDR. University of Electronic Science and Technology of China. Septiembre de 2019.

[15] USRP $N210$ product. Recuperado de: https://www.ettus.com/all-products/un210-kit/

[16] Tun/Tap interface tutorial. Marzo de 2010. Recuperado de: https://backreference.org/2010/03/26/tuntap-interface-tutorial/

[17] Soto Sánchez Ofelia. Comparación de la eficiencia volumétrica entre redes inalámbricas Wifi y Wimax. Facultad de Ingeniería, México. 2011

[18] Biondi Philippe et al. Scapy Documentation. Enero de 2020. Recuperado de: https://readthedocs.org/projects/scapy/downloads/pdf/latest/

[19] Lynn, et al. Transmission of IPv6 over Master-Slave/Token-Passing (MS/TP) Networks. Internet Engineering Task Force (IETF). Mayo de 2017.

[20] Oracle Corporation and/or its affiliates. Transport Interfaces Programming Guide. 2010. Recuperado de: https://docs.oracle.com/cd/E19504-01/802-5886/6i9k5sgsg/index.html

[21] Postel, J. Protocolo de Datagramas de Usuario. RFC768. Internet Engineering Task Force (IETF). 28 de Agosto de 1980.

[22] Bloessl B. GNU Radio WiFi Updates. Junio de 2016. Recuperado de: https://www.bastibl.net/wifi-updates/

[23] Remírez Moreno Berta. A study of IEEE 802.11ah and its SDR implementation. Escuela Técnica Superior de Ingenieros Industriales y de Telecomunicación. París, Francia. Junio de 2016.

[24] E. Research. USRP N210. 10 de Noviembre de 2016. Recuperado de: https://www.ettus.com/product/details/UN210-KIT

[25] Postel, J. Internet Control Message Protocol. RFC 792. InternetDraft Submission Tool. 30 de abril, 2009.

[26] Biondi Philippe. Scapy, Packet crafting for Python2 and Python3. Official Webpage. Recuperado de: https://cursos.clavijero.edu.mx/cursos/069_cIII/modulo1/contenidos/tema1.1.7.html

[27] Dispositivo de Radio Definido por Software USRP. National Instruments. Recuperado de: https://www.ni.com/es-mx/shop/select/usrp-software-defined-radio-device

[28] Abrahantes Huratdo Yilena. Implementación del protocolo 802.11p basado en SDR para los sistemas ITS. Universidad Nacional Autónoma de México. Noviembre de 2017