Facultad de Ingeniería - Universidad de la República

Proyecto de grado

Construcción de routers inalámbricos para monitorizar calidad de servicio

Autores:
Johnatan Stanley Galli
Nicolás Santiago Puppo Perna

Supervisores:
Matías Richart Gutiérrez
Eduardo Grampín Castro

Instituto de Computación

9 de julio de 2015

"Confine yourself to observing and you always miss the point of your life. The object can be stated this way: Live the best life you can. Life is a game whose rules you learn if you leap into it and play it to the hilt. Otherwise, you are caught off balance, continually surprised by the shifting play. Non-players often whine and complain that luck always passes them by. They refuse to see that they can create some of their own luck. Darwi Odrade - Chapterhouse: Dune"

Frank Herbert

("Limítate a observar y siempre perderás el sentido de tu vida. Se puede poner de esta manera: Vive la mejor vida que puedas. La vida es un juego cuyas reglas aprendes si saltas a ella y la juegas a fondo. De lo contrario, te encontrarás en una inestabilidad constante, sorprendiéndote continuamente de los cambios del juego. Quienes no juegan, a menudo se quejan de que la suerte les pasa por al lado. Se niegan a ver que pueden crear un poco de su propia suerte. Darwi Odrade - Dune: Casa Capitular")

Resumen

Las redes inalámbricas 802.11, conocidas comúnmente como redes "Wi-Fi", se enfrentan a una gran variedad de problemas de señal y cobertura. En este proyecto se construye una herramienta que monitorea las redes inalámbricas, brindando diversa información de las mismas, teniendo como objetivos centrales entender sus problemas y analizar su funcionamiento, permitiendo lograr una mejor planificación.

A lo largo del proyecto se ha avanzado en la construcción de esta herramienta, comenzando con la interiorización en este tipo de redes y sus problemáticas, realizando luego diferentes estudios del estado del arte, posibles arquitecturas y parámetros de monitorización interesantes, para finalmente comenzar con el desarrollo de la herramienta.

El desarrollo se agrupa en diferentes etapas, desde el diseño y arquitectura del sistema, hasta su implementación completa utilizando C y C++. Luego de completado el desarrollo, se realizó como caso de estudio un despliegue en el Instituto de Computación con el objetivo de tomar mediciones de las redes inalámbricas dentro de un entorno real. Finalmente se analizaron los resultados obtenidos en el despliegue para hallar conclusiones sobre la utilidad de la herramienta, verificando que cumple con los objetivos propuestos.

Agradecimientos

A quienes nos acompañaron durante esta importante etapa. Se cierra un ciclo del cual nos llevamos muchísimo. Sigamos disfrutando el camino. Gracias!

A Shawn Patterson, Joshua Bartholomew, Lisa Harriton, The Lonely Island, Tegan y Sara, por alegrar nuestras jornadas de desarrollo. Everything is Awesome!

Contenido

Dedicado a los que sabían que este día iba a llegar...

Capítulo 1

Introducción

1.1. Problemática existente

Es común que en sitios públicos (como universidades, centros comerciales, etc.) se cuente con un servicio de red inalámbrica, el cual debe poseer una amplia cobertura para ser brindado en buenas condiciones a los usuarios dentro de la zona que se quiere alcanzar. En entornos de este tipo es complejo y difícil planificar la instalación de puntos de acceso (los cuales brindarán el acceso a la red) de forma que se disponga de buena cobertura y calidad de servicio.

En particular, en lo que refiere a la calidad, existen diversos problemas para brindar un buen nivel de servicio. Esto ocurre por las particularidades del medio que utilizan las redes inalámbricas para transportar los datos, es decir, el aire, donde existen diversas fuentes de problemas que pueden mermar la calidad del servicio ofrecido. Esto se debe en parte a la heterogeneidad del medio, en el cual coexisten diferentes tipos de señales junto a los datos transportados por la red inalámbrica, como son aquellas generadas por los teléfonos inalámbricos, celulares, radios, hornos microondas, entre otros, y son conocidas como interferencia.

Además, no solo existen estas otras señales que interfieren con la red inalámbrica, sino que, a diferencia de una red cableada, donde los datos se transfieren por un medio dedicado, en las redes inalámbricas los datos se encuentran y colisionan con los objetos físicos que existen en el espacio mientras viajan entre dos puntos, ya sea una pared, un mueble, una persona, etc. Estas interferencias reducen la calidad del servicio que brinda la red, generando problemas para el usuario final al querer utilizar la misma.

Más adelante, se analizan en profundidad los diferentes tipos de problemas existentes, planteando los escenarios más comunes a los que suelen enfrentarse las redes de este tipo.

1.2. Objetivos

En este proyecto se propone abordar los problemas existentes en redes inalámbricas densas desde el punto de vista de su detección y el análisis de sus causas. Se plantea implementar un sistema de monitoreo utilizando routers que sean capaces de obtener ciertos indicadores de la red para ayudar en su planificación, con el objetivo de ofrecer un buen nivel de servicio. Se tomarán mediciones de diversas variables del medio inalámbrico para obtener métricas de la calidad del servicio.

Para realizar dicha tarea se cuenta con placas de routers con capacidad de dos interfaces inalámbricas (hardware existente en la facultad). De esta forma es posible utilizar una para dar servicio y la otra para monitorizar (monitor fuera de banda). Como se explicará más adelante, además de estas placas, se disponen de routers TP-LINK que no poseen dos interfaces inalámbricas, por lo que pueden cumplir una única función a la vez.

Se realizará un estudio del estado del arte de diferentes temáticas que ayudarán a la elección de los distintos componentes del sistema a construir. Para profundizar en los dispositivos y sistemas operativos disponibles para los routers se realizará el estado del arte en hardware de routers inalámbricos y el estado del arte en firmware de código abierto. Por otro lado se estudiará el estado del arte en lenguajes de programación para sistemas embebidos para la construcción del software que se desarrollará en el sistema.

Junto a lo anterior se estudiarán diferentes arquitecturas de monitorización distribuida y se buscarán las variables relevantes a ser medidas para tener un buen indicador de la calidad del servicio. Finalmente, se analizarán los resultados obtenidos.

En resumen, los objetivos son implementar el sistema de monitoreo mencionado y obtener resultados a través de su utilización.

1.3. Composición del trabajo presentado

El trabajo desarrollado se presenta de la siguiente forma:

- En el capítulo 1 se presenta el problema y los objetivos buscados.
- En el capítulo 2 se realiza una introducción a la temática abordada.
- En el capítulo 3 se presenta el hardware utilizado.
- El capítulo 4 contiene el estudio de los diversos estados del arte mencionados anteriormente (en hardware de routers inalámbricos, en firmware de código abierto y en lenguajes

de programación para sistemas embebidos), además del análisis de arquitecturas de monitorización.

- En el capítulo 5 se presenta el estudio y la selección de los indicadores a utilizar para medir la calidad de la señal, los diferentes sistemas de monitorización existentes y una plataforma en concreto, propietaria, que coincide en algunos aspectos con lo buscado en este proyecto.
- El capítulo 6 presenta el desarrollo realizado sobre la infraestructura.
- En el capítulo 7 se muestran los resultados obtenidos.
- Finalmente, en el capítulo 8 se presentan las conclusiones y el trabajo a futuro.

Capítulo 2

Antecedentes

El presente capítulo está basado en el libro "Redes de Computadores: Un Enfoque Descendente" [1], en los documentos sobre redes inalámbricas [2], [3], [4] y [5], en la presentación "Wireless Security" [6] y en el artículo "Propagation Measurements and Models for Wireless Communications Channel" [7].

2.1. Redes inalámbricas

Una red inalámbrica es un conjunto de hosts, enlaces y estaciones base, mediante la cual los hosts consiguen comunicarse entre sí y con otras redes a las que las estaciones base tengan acceso. En una red inalámbrica tipo es posible identificar ciertos elementos:

- Hosts inalámbricos: los hosts son dispositivos que actúan como sistemas terminales. Pueden ser móviles o no, pero su conexión con el resto de la red se realiza a través del medio inalámbrico. Una notebook, un celular o una computadora de escritorio son posibles ejemplos.
- Enlaces inalámbricos: un host se conecta a la estación base o a otro host por medio de un enlace de comunicaciones inalámbrico. Dependiendo de qué tecnología se use para este enlace, se tendrá cierta velocidad y distancia máxima entre equipos. También las estaciones base u otros dispositivos de la red pueden conectarse entre sí por medio de enlaces inalámbricos.
- Estación base: es una parte clave de la red inalámbrica. Es responsable de enviar y recibir datos desde y hacia un host inalámbrico que esté asociado con ella. Generalmente se encarga de coordinar la transmisión de los múltiples hosts inalámbricos que tenga asociados. El estar asociado quiere decir que ese host se encuentra dentro del alcance de la estación base

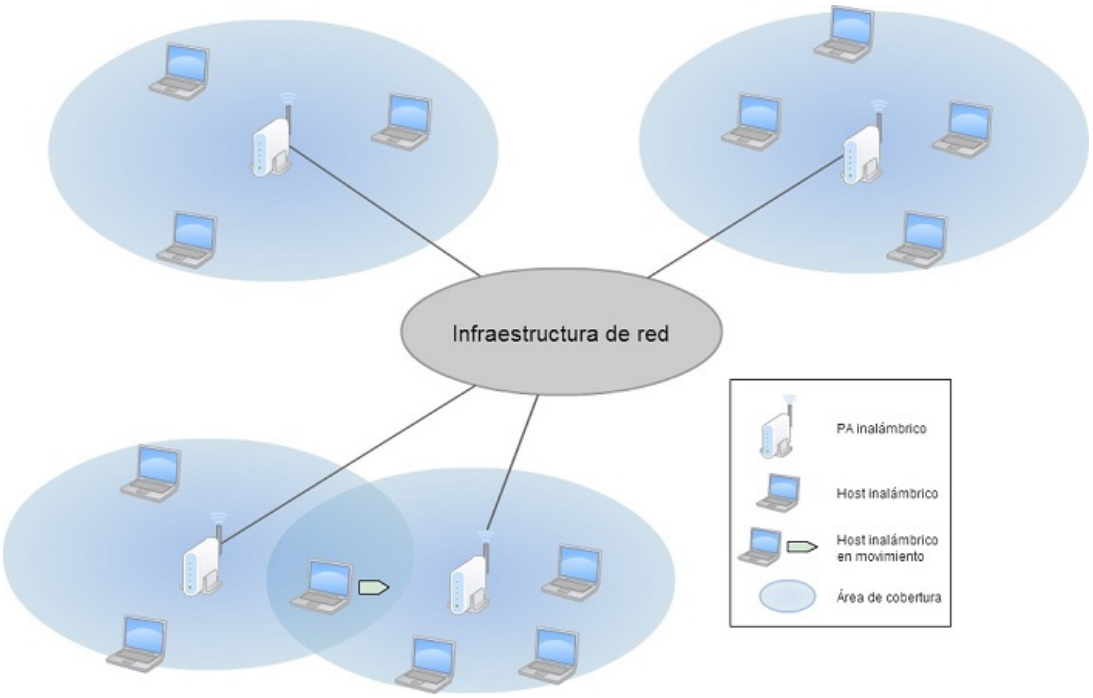

FIGURA 2.1: Escenario típico de una red inalámbrica.

y utiliza la estación base para re-enviar datos desde y hacia otra red u otro dispositivo. Particularmente en las redes LAN inalámbricas a las estaciones base se las conoce como puntos de accesos, AP, del inglés "Access Point".

Dichos elementos formando un escenario típico de una red inalámbrica se pueden apreciar en la figura 2.1.

Una estación base suele encontrarse conectada a otra red de mayor tamaño (generalmente Internet) y provee a la red inalámbrica la comunicación con esa otra red, permitiendo que los hosts inalámbricos asociados puedan enviar y recibir datos desde y hacia esa red.

Los hosts asociados a una estación base operan en modo de infraestructura. Lo cual implica que todos los servicios de red comúnmente usados (asignaciones de direcciones y enrutamiento, entre otros) son proporcionados por la propia red con la que el host se conecta por medio de la estación base. Esto se diferencia de las redes Ad Hoc, donde no existe infraestructura a la cual conectarse y los propios hosts deben realizar todas las funciones que generalmente brinda la red.

Cuando un host se mueve fuera del alcance de la estación base con la que estaba asociado y entra dentro del área de cobertura de otra estación base, cambia su estación asociada a la nueva. Esto se conoce con el nombre de handoff (transferencia).

De los diferentes elementos mencionados anteriormente se puede definir una clasificación de los diferentes tipos de redes según dos parámetros, si un paquete dentro de la red realiza un

	Con infraestructura	**Sin infraestructura**
Un único salto	Se tiene (al menos) una estación base conectada a una red cableada de mayor tamaño, por donde pasa toda la comunicación. Por ejemplo: redes 802.11, redes celulares y WiMAx.	No existe una estación base conectada a la red, pero los nodos pueden alcanzar sus objetivos de un solo salto. Por ejemplo: redes Bluetooth y redes 802.11 en modo ad hoc.
Múltiples saltos	Existe una estación base que se conecta con la red de mayor tamaño, pero algunos nodos inalámbricos pueden tener que retransmitir a través de otros para llegar a la estación base. Por ejemplo: algunas redes de sensores inalámbricos y las redes de malla inalámbricas.	No existe una estación base y los nodos pueden tener que retransmitir a través de otros nodos para alcanzar su objetivo. Por ejemplo: redes móviles ad hoc o redes vehiculares ad hoc.

TABLA 2.1: Clasificación de los tipos de redes.

único salto o múltiples saltos inalámbricos y si existe una infraestructura dentro de la red. Esta clasificación se puede apreciar en la tabla 2.1.

Como se puede ver, existen numerosos tipos de redes inalámbricas. En este proyecto nos enfocaremos en las redes LAN inalámbricas, del estándar 802.11, conocidas comúnmente como redes Wi-Fi.

2.2. Familia 802.11

Dentro del estándar 802.11 existen diversos tipos de especificaciones con diferentes usos. Las que definen tipos de redes LAN inalámbricas son cuatro: 802.11a, 802.11b, 802.11g y 802.11n.

En la tabla 2.2 se puede ver las características de las mismas.

La frecuencia de 5 GHz es menos usada, por lo que posiblemente posea menos interferencia que la 2.4 GHz, pero tiene menos alcance.

Como se puede ver en la tabla, los estándares mencionados presentan algunas diferencias entre sí, pero también poseen características comunes en cuanto a su funcionamiento, la estructura de trama para la capa de enlace [8], la capacidad de reducir la velocidad de transmisión para incrementar la distancia y los diferentes modos en los que pueden trabajar, entre otros. En la sección siguiente se detallarán estos tipos de redes.

	802.11a	**802.11b**	**802.11g**	**802.11n**
Año de lanzamiento	1999	1999	2003	2009
Frecuencia (GHz)	5	2.4	2.4	2.4 y 5
Área de cobertura	Área de cobertura reducida	Buen área de cobertura	Buen área de cobertura	Buen área de cobertura
Cantidad de canales que no se solapan	8	3	3	-
Máxima velocidad de transferencia por antena (Mbps)	54	11	54	+100
Rendimiento real (basado en el ambiente y los usuarios)	La mejor	Buena	Mejor que buena	La mejor
Compatibilidad	802.11a	802.11b	802.11b/g	802.11b/g/n
Penetrabilidad (en objetos sólidos)	Buena	Mejor que buena	Mejor que buena	-

Tabla 2.2: Caracterísiticas estándares 802.11 a/b/g/n.

2.3. Funcionamiento de una red LAN inalámbrica 802.11

Habiendo visto los diferentes estándares para la familia 802.11, se verá cómo es el funcionamiento de este tipo de redes. El conjunto de servicio básico, BSS (Basic Service Set) es un componente de arquitectura fundamental en una red de este tipo. Contiene uno o más hosts inalámbricos y una estación base central llamada AP (como se mencionó anteriormente). En la figura 2.2 puede verse un escenario típico de una arquitectura LAN inalámbrica 802.11.

Los APs se conectan generalmente a un switch o un router, el cual los conecta con Internet. En un hogar, típicamente se cuenta con un AP y un router, generalmente integrados en un mismo dispositivo que conecta el BSS con Internet. En cambio, la red utilizada en este proyecto estará formada por diferentes APs con su BSS, interconectados entre sí y/o con Internet por medio de switches o routers, conformando una red inalámbrica densa.

Como sucede en las redes cableadas, cada host, AP y router tendrán una dirección MAC por cada tarjeta de red que posean, la cual en teoría es única. Esta dirección MAC es muy importante para que exista comunicación en la red, como se verá mas adelante.

Para que un host pueda enviar o recibir datos de la red, debe primero asociarse con un AP. Asociarse implica crear una conexión entre el host y el AP, de forma tal, que solamente el AP sea quien envíe datos al host y el host solamente envíe datos al AP. Los APs poseen un Identificador

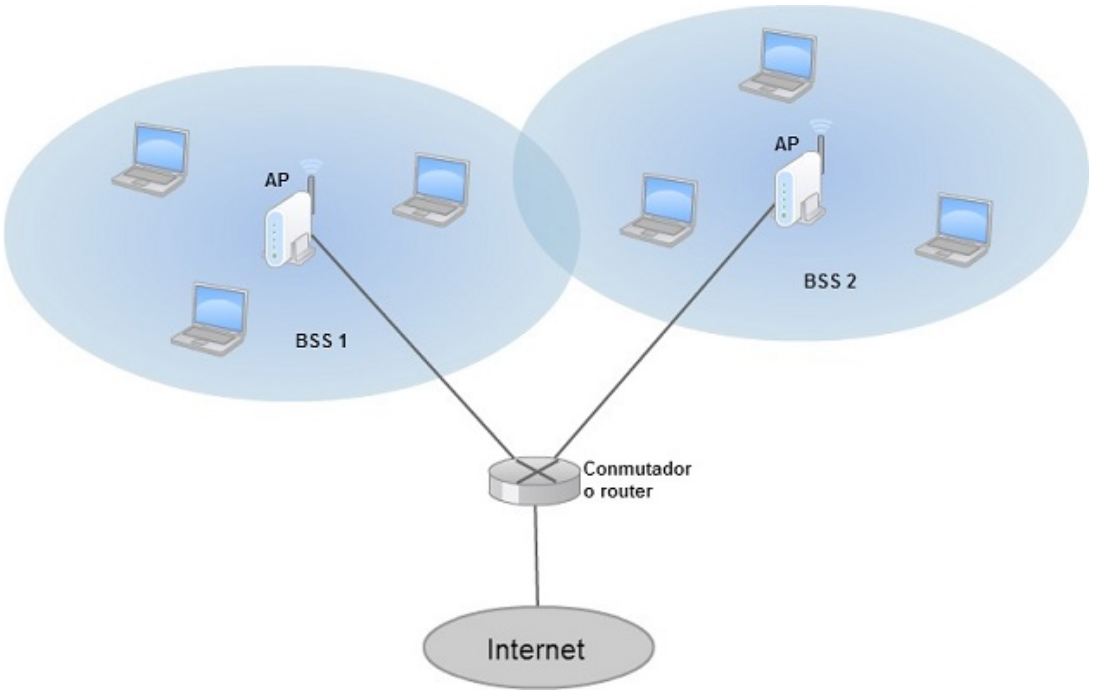

FIGURA 2.2: Arquitectura de una red IEEE 802.11.

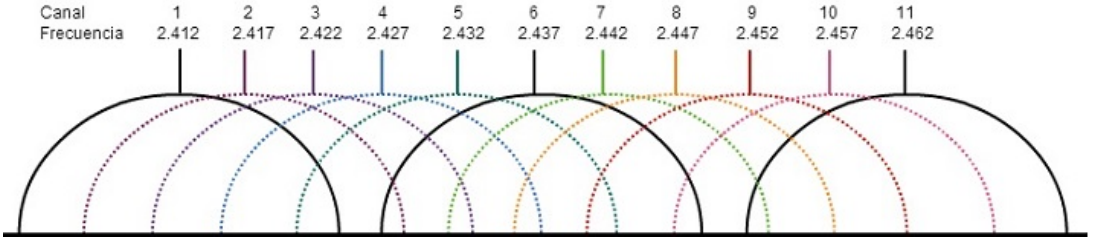

FIGURA 2.3: Solapamiento de los 11 canales.

de conjunto de servicio, SSID (Service Set Identifier) y un número de canal. El primero es la forma en que los hosts van a reconocer a ese AP, es decir, el nombre con el que será asociado, mientras que la segunda es el canal por el cual transmitirá y recibirá datos. En el capítulo 5 se explica en mayor nivel de detalle el uso de los canales. En nuestro país se utilizan 11 canales parcialmente solapados en frecuencia de 2.4GHz, como se puede ver en la figura 2.3. También se aprecia que en dichas condiciones existen hasta 3 canales que no se solapan entre sí, el conjunto formado por los canales 1, 6 y 11. Es importante destacar que el solapamiento implica que se genere ruido en los canales solapados, lo que provoca un deterioro en la calidad de la señal.

2.3.1. Asociación de los clientes con los APs

Los APs envían periódicamente tramas baliza (beacon frames) hacia la red, las cuales contienen su dirección MAC y su identificador SSID. Como los hosts esperan que los APs envíen estas tramas, exploran todos los canales en busca de las mismas, generando así un listado para que el

usuario o el propio host decida con cual AP asociarse. Esta exploración que realizan los hosts se denomina exploración pasiva.

Además de este tipo de exploración existe otro denominado exploración activa, en el cual los hosts difunden un paquete de sondeo que será recibido por los diferentes APs a los que llegue la señal. A continuación, los APs envían un paquete de respuesta, de esta forma el host puede identificarlos y elegir con cuál asociarse.

Cabe aclarar que el estándar no especifica la forma de seleccionar un AP para asociarse, sino que la selección queda a cargo del diseñador de software o del firmware del equipo. De todas formas, lo mas frecuente es que el host seleccione el AP cuya trama baliza se reciba con mayor intensidad de señal. Es importante destacar que esta elección no tiene por qué ser la mejor, ya que no solo es importante la intensidad que presente la señal, sino que también es importante la cantidad de hosts que ya se encuentren asociados a dicho AP. Cuanto cuanto mayor sea el número de hosts que esté atendiendo, menos tiempo y recursos podrá dedicarle a cada uno.

Luego de que un host decide con cuál AP asociarse le envía un paquete de solicitud de asociación al AP, el cual responde con un paquete de respuesta de asociación, quedando así asociado y pasando el host a formar parte de la subred a la que pertenece el AP. Esta asociación es un poco más compleja si el AP implementa un protocolo de seguridad, en cuyo caso el host deberá autenticarse para poder asociarse con el mismo.

Generalmente, luego de la asociación, el host buscará que se le asigne una IP, para lo cual enviará un mensaje de descubrimiento DHCP [9] hacia la subred a través del AP. DHCP es un protocolo mediante el cual se le asigna a un host una dirección IP libre perteneciente a la subred. Cuando el host obtiene una dirección IP, pasará a ser visible para el resto de la red y posiblemente el mundo (en caso de estar conectado a Internet).

2.3.2. Utilización del canal por diferentes clientes

En una situación “ideal” el host sería el único equipo asociado al AP, pero esto no es así, ya que generalmente existen diferentes hosts asociados, por lo que debe existir alguna forma de que los distintos emisores (hosts asociados y el AP) coordinen las transmisiones que realizan. Para esto existen diferentes formas de repartir el uso del canal: particionamiento del canal, acceso aleatorio y toma por turnos. En particular 802.11 utiliza y define un protocolo de acceso aleatorio, denominado CSMA/CA, acceso múltiple por sondeo de portadora con evitación de colisiones, por su sigla en inglés. Esto quiere decir que cada emisor sondea el canal antes de transmitir, y en caso de que detecte que el mismo está ocupado no transmite, y en su lugar espera un tiempo aleatorio para volver a intentarlo. Si detecta que el canal no está en uso, entonces comienza a transmitir y no se detiene hasta haber finalizado.

Este protocolo emplea un reconocimiento (ACK) de la capa de enlace, mediante el cual un emisor recibe una notificación de que su paquete ha llegado a destino. Esto es importante, ya que, como se verá más adelante, los paquetes no siempre logran llegar intactos a su destino. Este reconocimiento funciona de la siguiente manera: cuando el destinatario recibe un paquete, comprueba su estado mediante la comprobación CRC [10], espera un período de tiempo llamado "espacio corto entre tramas" (SIFS por sus siglas en inglés) y luego envía un paquete de reconocimiento a quien envió el paquete anterior. Si este no recibe la trama de reconocimiento dentro de un período de tiempo determinado, supone que no ha llegado correctamente y vuelve a enviarlo (siempre utilizando CSMA/CA para acceder al canal). Si alcanza cierto número de retransmisiones sin recibir el reconocimiento, descarta el paquete.

Es importante destacar que el sondeo de uso del canal solamente tiene lugar cuando se comienza una transmisión. Por lo tanto, si dos emisores comienzan a transmitir simultáneamente, ninguno sabrá que el canal se encuentra en uso y ambos transmitirán todo el paquete que pretendían enviar, lo cual puede generar una importante degradación del servicio, sobre todo ante paquetes largos. Por lo tanto, el protocolo presenta una forma de intentar evitar esta problemática y así reducir las colisiones, la cual se puede ver en el funcionamiento del mismo.

2.3.3. Detectando el estado del medio

Cuando una estación (ya sea un host o un AP) debe transmitir, primero comprueba que el canal (o medio) esté inactivo. Esta comprobación se realiza de forma física y virtual. Se denomina CCA, Clear Channel Assessment, al mecanismo para determinar si el medio está inactivo. Posee dos componentes: detección de portador (CS, Carrier Sense) y detección de energía. La primera de ellas a su vez está compuesta por una CS física y una CS virtual. La física es una medición directa de la intensidad de la señal recibida correspondiente al protocolo 802.11. Si se encuentra por encima de cierto nivel (82 dBm) el medio se considera ocupado. Mientras tanto, la virtual refiere al NAV (Network Allocation Vector), que funciona como un indicador de cuándo el canal estará inactivo. El NAV se actualiza cada vez que se recibe un paquete 802.11 válido que no tiene como objetivo la propia estación, de forma que una estación puede evitar transmitir aún cuando la CS física detecte inactividad. La segunda, detección de energía, revisa si el medio se encuentra ocupado, midiendo el total de energía recibida sin importar de dónde provenga. Si el mismo supera cierto nivel (62 dBm), entonces se considera que el medio está ocupado.

La comprobación de inactividad del canal mencionada es realizada durante un período de tiempo denominado DIFS, espacio distribuido entre tramas por sus siglas en inglés, para el caso en que no hayan existido errores de envío en la transmisión anterior, o uno llamado EIFS, espacio de error entre tramas, en caso de que haya ocurrido un error en una transmisión. En caso de que el medio se detecte inactivo durante el tiempo adecuado, se transmite el paquete completamente.

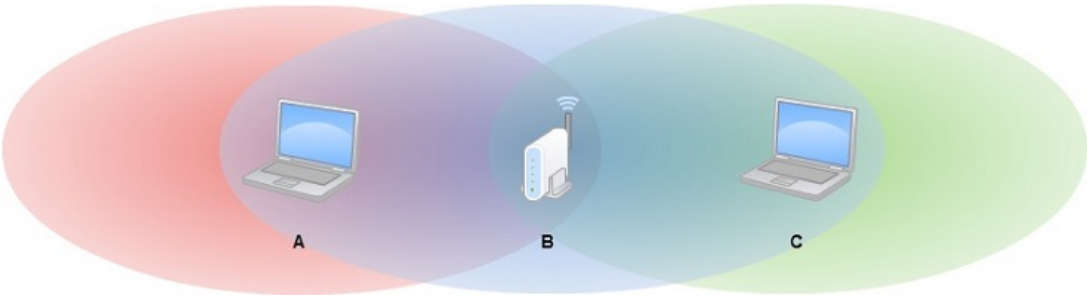

FIGURA 2.4: Terminal oculto por desvanecimiento de señal.

Si no fuese así, se selecciona un valor de espera aleatorio llamado backoff, que representa el tiempo que la estación debe detectar al canal inactivo para poder transmitir. Este backoff se elige dentro de un rango posible, el cual crece al doble cada vez que se realiza una transmisión errónea y retorna a su mínimo cuando se realiza una transmisión satisfactoria. Mientras se detecte que el canal se encuentra inactivo durante un cierto período de tiempo, denominado "slot time", se decrementa el valor del backoff. Una vez que el backoff alcance el valor 0, la estación transmite la trama completa y espera a recibir un reconocimiento. En caso de recibirlo considera que se ha transmitido correctamente y vuelve a ejecutar el algoritmo para el siguiente paquete que posea. En caso de no recibir reconocimiento, el emisor vuelve a entrar en la fase de backoff para retransmitir el paquete.

Cabe aclarar que se espera que el backoff elegido por diferentes emisores sea distinto, lo cual ayuda a evitar las colisiones.

2.3.4. Problema del terminal oculto

De todas formas, existen ciertos problemas que este algoritmo no soluciona, como el problema del terminal oculto a causa del desvanecimiento de la señal. Esto ocurre cuando existen al menos 3 estaciones, dos de las cuales quieren transmitir hacia la tercera, pero están ubicadas de forma tal que no se encuentran dentro del alcance de la otra, por lo tanto no logran detectar lo que la otra emite. Este escenario se puede apreciar en la figura 2.4.

Otro problema que no contempla el algoritmo es el del terminal oculto a causa de un obstáculo, en el cual tanto A como C estarían dentro del radio de acción del otro, pero la existencia de un obstáculo entre ellos hace que detecten sus transmisiones, como se puede apreciar en la figura 2.5.

Por lo tanto, cuando A y C intenten transmitir hacia B, ambos verán el canal libre, por más que uno de los dos ya se encuentre transmitiendo, dado que la señal que envía cada uno de ellos no alcanza al otro. En cambio, B recibirá simultáneamente ambas transmisiones, y por lo tanto se provocará una colisión, lo que haría que el canal sea desperdiciado durante todo el tiempo de

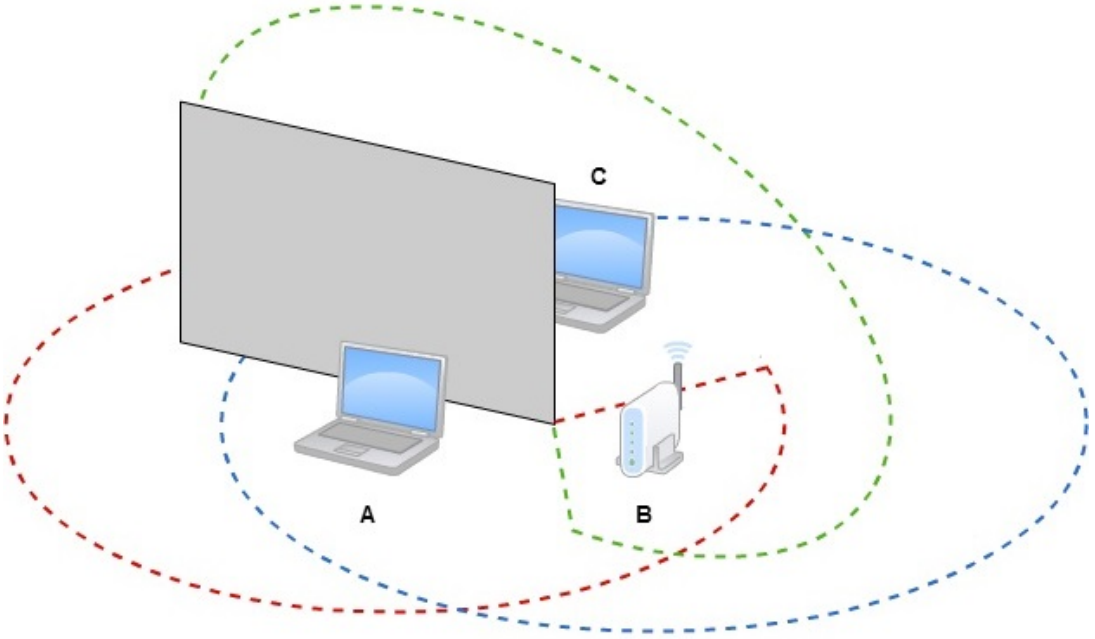

FIGURA 2.5: Terminal oculto por obstáculo.

transmisión del paquete más largo de los dos. Para solucionar este problema, el estándar presenta un esquema de reserva (el cual no es obligatorio implementar ni usar). Este esquema permite a las estaciones utilizar dos paquetes de tamaño pequeño denominados Solicitud de transmisión (RTS, por sus siglas en inglés) y Preparado para enviar (CTS, por sus siglas en inglés) para reservar el uso del canal. Cuando un emisor pretende comenzar a transmitir, primero envía un RTS al AP, indicando el tiempo necesario para completar la totalidad de la transferencia que desea realizar y para recibir el ACK. Cuando el AP recibe el RTS, responde difundiendo a toda la red un CTS, el cual tiene un doble propósito: le informa al emisor que puede transmitir y le avisa al resto de estaciones que no pueden transmitir durante el período de tiempo indicado. Este esquema se puede apreciar en la figura 2.6.

De esta forma se mitigaría el problema de la terminal oculta. Podría darse que existan colisiones cuando se envían RTS o CTS, pero dado que estos paquetes poseen un tamaño reducido, la colisión no es tan relevante. Como contrapartida, este método introduce retardos y consume recursos del canal, por lo tanto, si se utiliza, se reserva para paquetes de tamaño considerablemente grandes.

2.4. La trama IEEE 802.11

Una trama es la forma de nombrar a un paquete a nivel de capa de enlace. Es una agrupación de bits organizada de una forma específica. La trama 802.11 posee similitudes con la trama

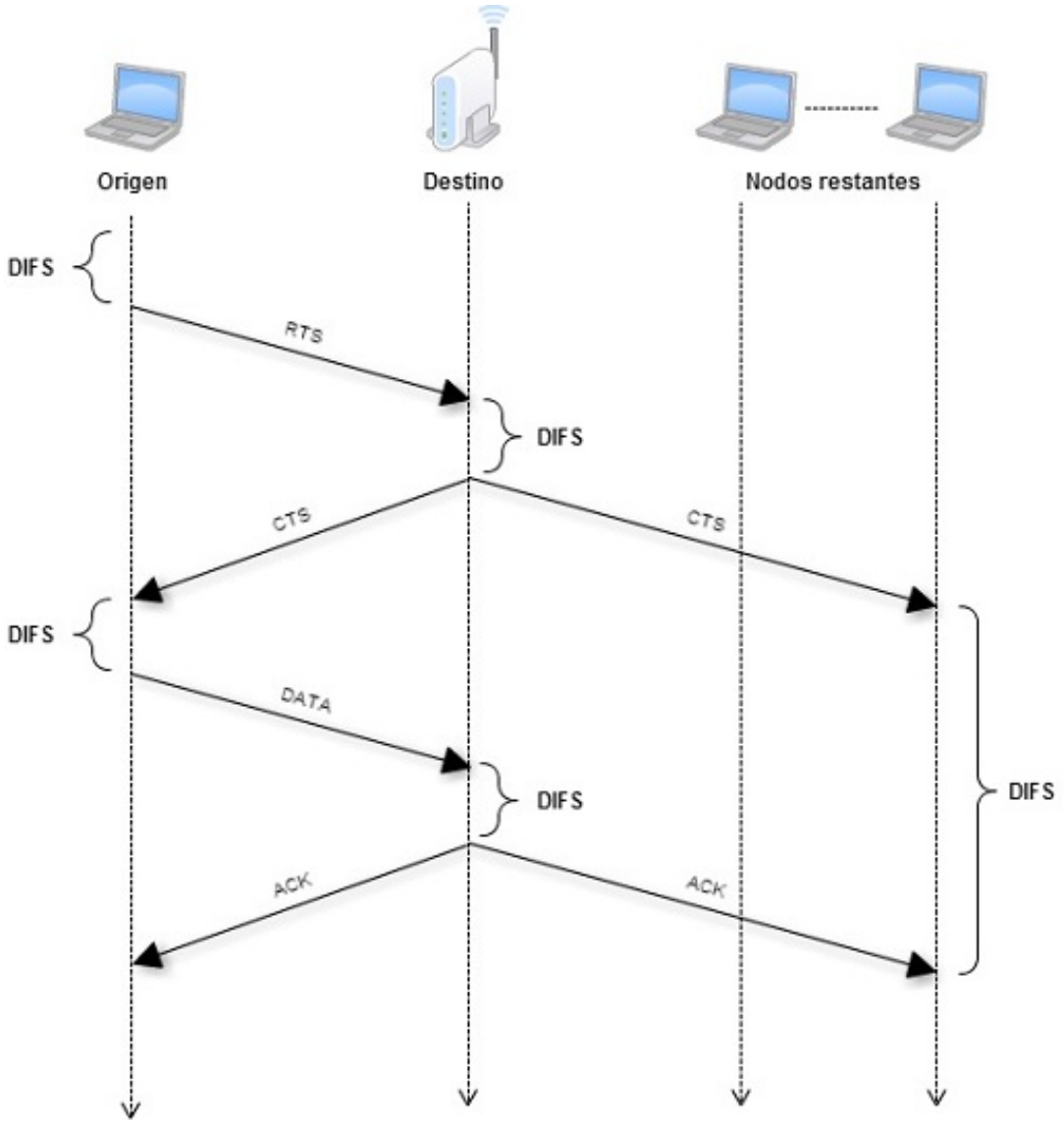

FIGURA 2.6: Evitación de colisiones usando RTS y CTS.

Ethernet IEEE 802.3 [11], pero a su vez posee ciertos campos que son particulares para el ámbito inalámbrico.

A continuación se presentan los diferentes campos, y en la imagen 2.7 puede apreciarse la composición de la misma.

Control de trama: este campo incluye varios sub-campos, los cuales se listan a continuación:

- Versión protocolo: indica la versión usada del protocolo 802.11.
- Tipo y subtipo: determinan la función de la trama. Existen 3 tipos diferentes: control, datos y administración. Para cada uno existen numerosos sub-tipos según la operación que se esté realizando.

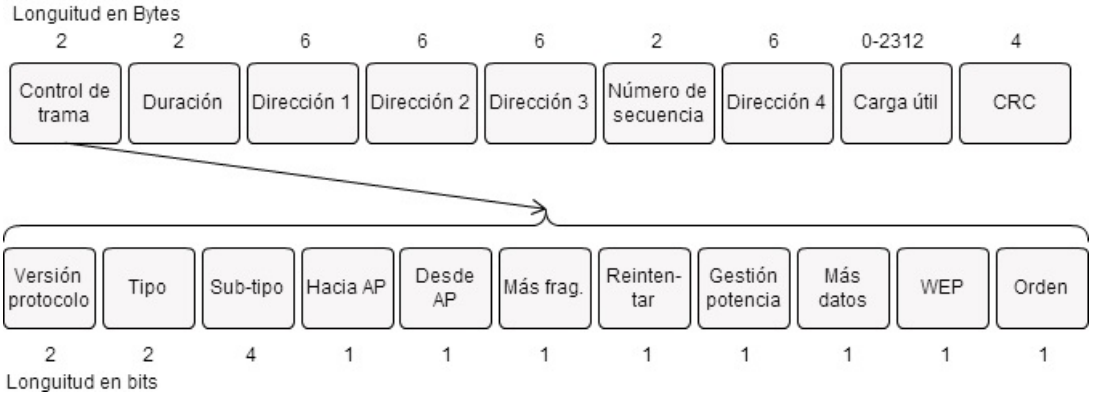

FIGURA 2.7: Trama 802.11.

- Hacia AP y desde AP: definen los significados de las direcciones. Son diferentes para modo ad hoc e infraestructura, y además, para éste último también cambian dependiendo de si el emisor de la trama es un AP o un host.
- Más fragmentos: indica si existen más fragmentos de la trama.
- Reintentar: indica si la trama está siendo retransmitida.
- Gestión de potencia: indica cuando la estación emisora está en modo activa o en ahorro de energía.
- Más datos: indica a un host en modo ahorro de energía que un AP tiene más paquetes para enviar. Tiene algunos otros usos particulares.
- WEP: indica si se está usando autenticación y cifrado.
- Orden: indica si las tramas recibidas deben ser procesadas en orden.

Campo duración: es utilizado para definir por cuánto tiempo se pretende reservar el medio. Como se vió anteriormente, esto refiere al uso de RTS y CTS.

Campos de dirección: la trama posee cuatro campos de este tipo, donde cada uno puede contener una dirección MAC de 6 bytes. Las primeras tres de estas direcciones son necesarias para la comunicación en la red. En particular, para mover el paquete desde una estación inalámbrica hasta una interfaz de router a través de un AP. La primera dirección contiene la MAC de la estación inalámbrica que tiene que recibir la trama, la segunda dirección contiene la MAC de la estación que transmite la trama, la tercera es un tanto especial, ya que contiene la dirección MAC de la interfaz del router a través de la cual el BSS se conecta con otras sub redes. El cuarto campo de dirección es usado solamente cuando los APs se re-envían tramas entre sí en modo ad hoc.

Campo número de secuencia: es utilizado para identificar una trama y poder distinguir tramas entre sí.

Carga útil y CRC: la carga útil generalmente estará compuesta por un paquete IP o ARP. La longitud máxima del campo es de 2312 bytes, pero generalmente no posee más de 1500 bytes. Luego de la carga útil se incluye el CRC (código de redundancia cíclica) de 32 bits, usado por el receptor para detectar errores en la trama recibida.

2.5. Problemas de calidad de la señal en las redes inalámbricas

Existen 3 factores principales que afectan la calidad de la señal en las redes inalámbricas:

- Intensidad decreciente de la señal: mientras la señal se propaga va perdiendo intensidad, atenuándose a medida que atraviesa diferentes objetos, incluido el aire. A esta pérdida se le denomina pérdida de propagación (path loss). Por lo tanto, cuanto más distanciados se encuentren el emisor y el receptor, y cuanto más objetos existan entre ellos, mayor será la pérdida.
- Interferencia de otros orígenes: los diferentes equipos que transmiten en una misma frecuencia generan interferencia entre sí, ya sean del mismo tipo (como más de una red inalámbrica en el mismo canal o en canales solapados) o de otros tipos que no tengan que ver con redes inalámbricas (como un teléfono inalámbrico transmitiendo en la frecuencia de 2.4 GHz con un AP transmitiendo en la misma frecuencia). Además de la interferencia entre equipos que son transmisores, existe también interferencia causada por ruido electromagnético, como el que produce un motor o un microondas.
- Propagación multicamino: este tipo de pérdida de señal sucede cuando partes de la onda electromagnética toman caminos de diferentes longitudes entre el emisor y el receptor. Esto puede deberse a que la onda se refleja en los objetos existentes y el suelo, por lo que la señal recibida en el receptor será menos limpia. Incluso si existen objetos que se están desplazando que se ubiquen entre el emisor y el receptor, la propagación multicamino variará a lo largo del tiempo.

La pérdida de señal que provocan estos factores puede comprobarse y medirse de diferentes formas, las cuales serán exploradas en el capítulo 5.

Capítulo 3

Hardware

3.1. Hardware utilizado

Este capítulo está basado en los sitios de Routerboard [12], [13], [14], "Linux Wireless" [15], [16], [17], TP-Link [18], [19], los manuales Routerboard 133 [20], Routerboard R52 [21], [22] y TP-Link [23] y el catálogo de Routerboard [24].

El hardware utilizado al comienzo de este proyecto estuvo compuesto por placas Routerboard 133 y tarjetas inalámbricas Routerboard R52.

Routerboard nace en el 2002, de la mano de la empresa MikroTik, como una gama de productos de hardware de routers. MikroTik fue fundada en 1995 enfocada al desarrollo de sistemas de routers y routers inalámbricos. Esta empresa apunta a un mercado que va desde el hogar hasta grandes corporaciones.

Se había elegido este hardware porque se encontraba disponible en el grupo de investigación "MINA" y ofrece una gran posibilidad de personalización.

Como se explicará más adelante, avanzado el proyecto se discontinuó el uso de este hardware y se optó por utilizar TP-Link TL-WDR4300 en su lugar.

3.2. Routerboard 133

En las figuras 3.1 y 3.2 se presentan la placa utilizada y un diagrama de la misma.

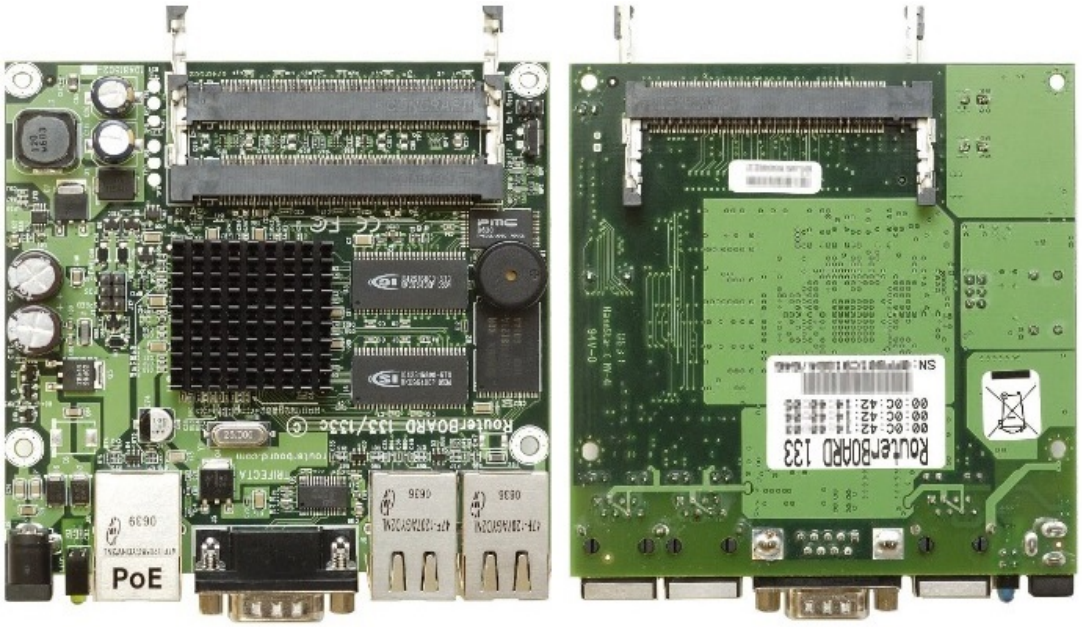

FIGURA 3.1: RouterBoard 133 vista superior e inferior.

Las características principales de este dispositivo se presentan en la tabla 3.1.

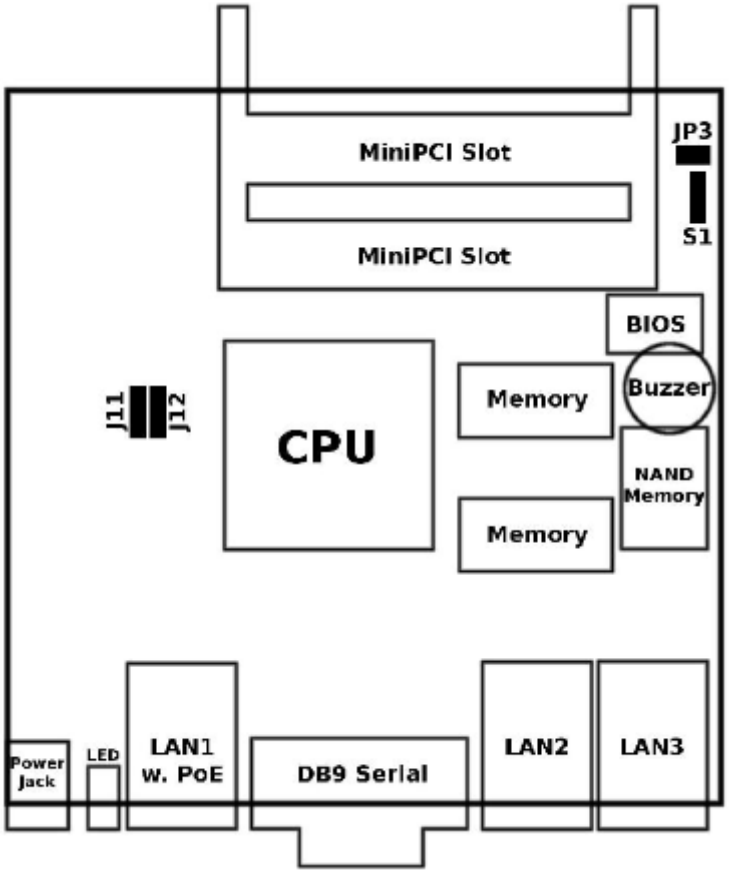

FIGURA 3.2: RouterBoard 133 diagrama.

Procesador	Procesador embebido MIPS32 4Kc 175MHz (little-endian), incluye TLB (Translation Lookaside Buffer) y no posee unidad de punto flotante (está optimizada para trabajar con enteros)
Memoria	32MB SDRAM onboard
Boot Loader	RouterBOOT, 1Mbit Flash chip
Almacenamiento	64 MB onboard NAND no volátil
Ethernet	3 puertos 10/100 Mbit/s Fast Ethernet
Slot MiniPCI	3 slots MiniPCI tipo IIIA/IIIB
Puerto serial	Un puerto serial DB9 RS232C asincrónico
LEDs	LED de encendido (indica cuando la placa está prendida) y de uso para el desarrollador (se enciende al iniciar y se apaga al levantar el kernel)
Parlante	Mini PC-Speaker
Energía	POE (Power over ethernet) 16 a 28V DC solamente en el puerto LAN1. Power jack: 9 a 28V DC
Consumo de energía	3-4W sin tarjetas conectadas, máximo 10W
Dimensiones y peso	11.7 cm x 10.5 cm (4.61 in x 4.13 in). 115 gramos
Temperatura y humedad	-20°C a +65°C, hasta 70 % de humedad relativa
Arquitectura	Completamente compatible con el estándar de arquitectura MIPS32 con bus PCI

Tabla 3.1: Caracterísiticas del RouterBoard 133.

3.3. Routerboard R52

La tarjeta R52 es una tarjeta miniPCI de doble banda. En la tabla 3.2 se presentan sus principales características, y en la figura 3.3 se puede ver una imagen de la misma.

Chipset	Atheros AR 5414
Estándares	802.11 a/b/g
Acceso al medio	CSMA/CA con ACK
Seguridad	64/128 bit WEP, Cifrado TKIP y AES-CCM, Autenticación WPA, WPA2, 802.1x
Conectores	Dos conectores U.fl
Certificaciones	FCC, EC
Energía	3.3V +/- 10 % DC
Temperatura y humedad soportada	-20°C a +70°C, hasta 90 %
Dimensiones y peso	6.0cm x 4.5cm, 20 gramos
Potencia y sensibilidad en 802.11 a	17dBm / -88dBm - 6Mbps 13dBm / -71dBm - 54Mbps
Potencia y sensibilidad en 802.11 b	19dBm / -95dBm - 1Mbps 19dBm / -90dBm - 11Mbps
Potencia y sensibilidad en 802.11 g	18dBm / -90dBm - 6Mbps 15dBm / -73dBm - 54Mbps
Velocidades de transferencia	802.11b:11,5.5,2,1 Mbps, auto-fallback 802.11g:54,48,36,24,18,12,9,6 Mbps, auto-fallback 802.11a:54,48,36,24,18,12,9,6 Mbps, auto-fallback En modo turbo: 802.11g:108,96,72,48,36,24,18,12 Mbps, auto-fallback 802.11a:108,96,72,48,36,24,18,12 Mbps, auto-fallback

Tabla 3.2: Caracterísiticas del RouterBoard R52.

Figura 3.3: RouterBoard R52.

Driver	AP	IBSS	Mesh	MON
ath5k	Si	Si	Si	Si

Tabla 3.3: Modos soportado por la tarjeta RouterBoard R52

Se investigó el chipset utilizado por la tarjeta para analizar los modos soportados por la misma. Dicho chipset es un Atheros AR5414, que a su vez utiliza el driver "ath5k", por lo que se investigó los modos soportados por este driver. Estos modos se pueden apreciar en la tabla 3.3.

El modo AP es el modo más común en los routers (el cual ha sido descrito en la introducción).

El modo MON (monitor) es un modo pasivo, en el que el equipo (generalmente) no transmite datos, pero en cambio obtiene todos los paquetes de la red sin importar su objetivo, sin aplicar ningún filtro. Es el modo comúnmente utilizado cuando se pretende analizar el tráfico de una red.

IBSS (Independent Basic Service Set), conocido también como modo Ad-Hoc, es usado para crear una red inalámbrica sin la necesidad de disponer de un AP en la misma. Es el modo punto a punto.

El modo Mesh, es usado para permitir a múltiples dispositivos comunicarse entre sí estableciendo rutas dinámicas entre ellos de modo inteligentes.

3.4. OpenWRT

Sobre el hardware mencionado se instaló el sistema OpenWRT, cuyo análisis se encuentra en el capítulo 3, en la sección de estado del arte de firmware de routers.

El trabajo realizado sobre la placa RouterBoard 133 para efectuar la instalación de OpenWRT se detalla en el capítulo 6.

3.5. TP-LINK

Como se detallará en el capítulo 5, se utilizó también un router TP-Link TL-WDR4300. Su característica más destacable, que lo hace relevante para este trabajo, es el soporte para OpenWRT en versiones más recientes, específicamente la versión utilizada es 12.09 "Attitude Adjustment".

Chipset	Atheros AR 9344 y Atheros AR 9580
Estándares	802.11 a/b/g/n
Antenas	3 antenas de banda dual 2.4GHz y 5GHz
Seguridad	Encriptación WEP,WPA / WPA2,WPA-PSK/ WPA2-PSK de 64/128-bit
Energía	12 VDC / 1.5A
Dimensiones	24.3 cm x 16.06 cm x 3.25 cm
Tasas de transferencia	5GHz hasta 450 Mbps 2.4GHz hasta 300 Mbps

Tabla 3.4: Caracterísiticas del TP-Link TL-WDR4300.

Figura 3.4: TP-Link TL-WDR4300.

El resto de las características del router, a pesar de ser interesantes, no son relevantes para el desarrollo de este proyecto, por lo que no fueron utilizadas.

Capítulo 4

Estado del arte

4.1. Estado del arte en hardware de routers inalámbricos

Esta sección está basada en el libro "Redes de Computadores: Un Enfoque Descendente" [1] y en los sitios [25], [26] y [27]. La subsección de cada fabricante y la comparación entre ellos se basa en: Qualcomm [28], [29] y [30], Intel, [31], [32] y [33], y Broadcom [34], [35] y [36].

4.1.1. Router Inalámbrico

Un router es un dispositivo de interconexión presente en una red, que cuenta con varios enlaces y toma los paquetes que llegan por cualquiera de ellos, reenviándolos por el enlace adecuado, de forma de hacer llegar el paquete al destino deseado. Para ello, el dispositivo posee capacidad de almacenamiento y de decisión para saber hacia dónde debe enviar el paquete. Este re-envío lo realiza utilizando direcciones de capa 3, capa de red [8]. La capa de red tiene como objetivo lograr que los datos lleguen desde un origen a un destino aún cuando ambos no se encuentren conectados directamente. En este nivel se realiza el direccionamiento y re-envío hasta su destino.

Un punto de acceso inalámbrico es un dispositivo que integra una red inalámbrica, al cual los usuarios inalámbricos de dicha red transmiten paquetes (y reciben) de forma inalámbrica, utilizando ondas de radio, que generalmente está conectado al resto de la red de forma cableada.

Dadas estas dos definiciones, se define un router inalámbrico como un dispositivo que realiza tanto las funciones de un router como de un punto de acceso inalámbrico.

Cuando una red posee al menos un router inalámbrico, se dice que es una "red de infraestructura inalámbrica". Los dispositivos inalámbricos que deseen conectarse a la red (notebooks, celulares, impresoras, etc.) lo harán a través del router inalámbrico, que permite que los dispositivos conectados interactúen entre sí y con el resto de los dispositivos que sean alcanzables

desde la red (en el caso de un hogar con conexión de datos, sería a toda la internet). Como se mencionó anteriormente, estos routers inalámbricos pueden estar conectados al resto de la red por un cable Ethernet, y a su vez pueden permitir que otros dispositivos se conecten al router de forma cableada, por lo tanto una red de infraestructura inalámbrica también puede poseer conexiones por cable, tanto para dispositivos como para conectarse al resto de la red. Sin importar la forma en que los dispositivos se conecten al router, todos tendrán acceso a la misma red (o redes) y podrán comunicarse entre sí (siempre que así se permita).

4.1.2. Funcionamiento de un router inalámbrico

Como se explicó en el punto anterior, los routers reenvían la información (paquetes) que reciben tanto de los dispositivos conectados al mismo como del resto de la red. Cada vez que un router reenvía un paquete verifica que el mismo se haya enviado correctamente. De la misma forma, cada vez que recibe un paquete envía una verificación de recibido al emisor del paquete, sin importar si la transmisión del paquete es a través de un cable o de forma inalámbrica.

Para que el router pueda efectuar este reenvío de paquetes necesita saber hacia dónde debe enviar cada paquete, es decir, debe conocer el destinatario del mismo. La forma de identificar al destinatario es mediante la dirección IP. Esta dirección es asignada por el router a cada dispositivo conectado en el momento en que dicho dispositivo se conecta (pudiendo ser renovada por el router), asignación para la cual se usa el protocolo DHCP [9]. Cabe aclarar que las IP que tendrán asignadas los dispositivos conectados al router son IPs de la red local y no de Internet (en el caso que el router posea conexión a Internet). Si el router posee conexión a Internet, el mismo tendrá al menos una dirección IP única en Internet, asignada por su ISP [37] y otra local en la red que él mismo gestiona.

Cuando el router recibe un paquete que debe reenviar entre Internet y la red local, necesita realizar una conversión de direcciones. Para esto posee una tabla NAT [38].

Se ha realizado un estudio de las diferentes piezas de hardware en materia de tarjetas inalámbricas existentes actualmente en el mercado. En las subsecciones siguientes, se presentan los resultados obtenidos.

4.1.3. Qualcomm Atheros

Qualcomm Atheros (anteriormente Atheros) es un desarrollador de semiconductores para redes y comunicaciones. Se especializa en chipsets inalámbricos. Fue fundada en 1998 bajo el nombre de Atheros por expertos en procesamiento de señales de la Universidad de Stanford, la Universidad de California en Berkeley y la industria privada. Posteriormente, en 2011, fue comprada por Qualcomm pasando a llamarse Qualcomm Atheros. Esta empresa tiene un importante interés

en el estudio, ya que los chipsets Atheros para el estándar IEEE 802.11 son usados por unos 30 fabricantes de dispositivos inalámbricos.

4.1.4. Intel

Intel es uno de los mayores fabricantes de circuitos integrados del mundo. Fundada en California el 18 de julio de 1968 como Integrated Electronics Corporation por Robert Noyce y Gordon Moore. A pesar de ser mayormente conocida por la creación de procesadores (fundamentalmente de la serie x86), también desarrollan chipsets inalámbricos.

4.1.5. Broadcom

Broadcom Corporation es uno de los principales fabricantes de circuitos integrados para comunicaciones de banda ancha inalámbricas y cableadas a nivel mundial. Fue fundada en 1991 por Henry Samueli y Henry Nicholas. Sus principales clientes son grandes compañías, como Amazon, Dell y Cisco, entre otras, pero también posee un gran mercado en los hogares.

4.1.6. Comparativa

En las tablas 4.1, 4.2 y 4.3 se puede observar una comparativa de las principales características de los últimos 4 modelos lanzados por cada compañía (a Octubre del 2014).

Se apuntó a relevar ciertas características de cada uno: modelo, interfaz, MiMo, estándares que implementa, fecha de salida, rango de frecuencia, parámetros de monitorización, parámetros accesibles por hardware y soporte de OpenWRT. Como no ha sido posible encontrar toda la información necesaria para todos los modelos, la comparativa no se ha podido realizar completamente. Entendemos que en algunas características el problema se debe a que son datos reservados de cada fabricante o no son datos que se encuentren públicamente. En particular, el soporte a OpenWRT no depende solo de esta pieza de hardware, sino que depende de todo el router en su conjunto, por lo que no es posible determinar qué versión de OpenWRT soporta cada tarjeta.

MiMo [39] refiere a “Multiple-input multiple-output”, múltiples antenas usadas de forma simultánea para transmitir y múltiples antenas utilizadas simultáneamente para recibir. Se denota de la forma ExR, donde E es el número de antenas usadas para emitir y R el número de antenas usadas para recibir.

	QCA9862	QCA9880	QCA9890	QCA6174
Interfaz	PCIe	PCIe	PCIe	PCIe USB3 SDIO
MiMo	2x2	3x3	3x3	2x2
Estándares	a/b/g/n/ac	a/b/g/n/ac	a/b/g/n/ac	a/b/g/n/ac
Fecha de salida	22/07/2013	15/07/2013	12/09/2013	05/06/2014

Tabla 4.1: Características diferentes hardware Qualcomm.

	AC7260	N 7260	N 7260 Plus	N 7260 Plus + BT
Interfaz	PCIe	PCIe	PCIe	PCIe
MiMo	2x2	2x2	2x2	2x2
Estándares	a/b/g/n/ac	a/b/g/n	a/b/g/n	b/g/n
Fecha de salida	17/04/2013	17/04/2013	17/04/2013	17/04/2013

Tabla 4.2: Características diferentes hardware Intel.

	BCM43162	BCM43602	BCM43241	BCM4339
Interfaz	PCIe	PCIe	SDIO	SDIO
MiMo	1x1	3x3	2x2	1x1
Estándares	a/b/g/n/ac	a/b/g/n/ac	a/b/g/n	a/b/g/n/ac
Fecha de salida	13/02/2014	06/02/2014	05/04/2013	05/09/2013

Tabla 4.3: Características diferentes hardware Broadcom.

4.2. Estado del arte en firmware de código abierto

Esta sección se basa en los artículos "Network performance evaluation based on three processes" [40], "Implementación openwrt del protocolo de comunicaciones w2lan" [41], "Plataforma móvil para aula virtual basada en wrt160nl-openwrtusb" [42] y "The openwrt embedded development framework" [43], y en el sitio [44].

Las diferentes secciones de cada firmware se basan en los sitios: Alchemy y Talisman [45], [46], [47] y [48], HyperWRT y Tomato [49], DD-WRT [50], [51], [52] y [53], OpenWRT [54], X-WRT [55], y Tarifa [56].

4.2.1. Introducción

Los firmwares libres tienen su origen en la empresa Linksys de Cisco (fabricante de hardware para redes domésticas y pequeñas empresas), particularmente con el modelo WRT54G; ya que

poco después de su lanzamiento se descubrió que su firmware se basaba en componentes del kernel de linux, y al regirse éste por la GNU GPL la empresa se vio obligada a liberar el código fuente del firmware. De esta manera, los desarrolladores pudieron aprender con exactitud cómo comunicarse con el hardware, y así nacieron varios proyectos open source como Alchemy, DD-WRT, OpenWRT, X-WRT, HyperWRT, Tomato, etc; orientados no sólo a arreglar los bugs de WRT54G y optimizar su rendimiento, sino también a extender sus funcionalidades.

Algunas de las funcionalidades agregadas que se pueden encontrar en estos firmwares son soporte para sistemas de distribución inalámbrica (WDS), puentes inalámbricos (Wireless Bridge), repetidores (Repeaters), montar un servidor VPN o VoIP, controlar el uso del ancho de banda por protocolo, alterar los tamaños del tráfico, incrementar la potencia de la antena, acceder a los logs del router de forma remota, ejecutar aplicaciones linux, etc.

Existen firmwares que ofrecen aún más funcionalidades, mientras otros se centran en las funcionalidades específicas de los routers, y otros buscan proveer una interfaz de usuario más amigable; pero un gran atractivo que ofrecen todos, además de las funcionalidades que traen de serie, es la posibilidad de desarrollar nuevas funcionalidades o ajustar las existentes según la necesidad.

4.2.2. Alchemy y Talisman - SVEASOFT

Alchemy es el primer firmware de routers de la empresa Sveasoft (y uno de los primeros firmwares de routers desarrollados por terceros), basado en el código fuente del WRT54G liberado por Linksys. En un principio ofrecían el firmware con algunas funcionalidades nuevas y arreglos de bugs respecto a la versión de Linksys, pero luego comenzaron a cobrar una membresía para tener acceso a su comunidad (que ofrecía los binarios de los productos y soporte para los mismos), provocando una batalla con los defensores de la GPL. A partir de ese momento comienzan a circular versiones del firmware en las redes P2P y sitios web, a lo que la empresa Sveasoft comunica que dichas versiones son pirateadas y contienen backdoors, mientras que los responsables de hacer público el firmware acusan recibir amenazas por parte del dueño de la empresa.

A raíz de esto la empresa discontinuó el desarrollo de Alchemy, y comenzó a ofrecer un nuevo firmware, Talisman, el cual agrega más de cien funcionalidades nuevas y arregla bugs encontrados comúnmente en las versiones de serie de los firmwares. Sin embargo, no se encontró información sobre qué funcionalidades incorporan tanto Alchemy como Talisman, lo que se presume que se debe a la turbiedad de la historia de la empresa y el secretismo con el que se manejó por fuera de su comunidad con respecto a sus productos.

Actualmente, la empresa parece haber desaparecido, ya que la última actualización de su página data del año 2010, y el último hilo en su foro (el único con menos de un año de antigüedad)

consta de notificaciones de sus clientes de que los enlaces de descarga no funcionan, para las cuales no tuvieron respuesta.

4.2.3. HyperWRT y Tomato

Al igual que en el caso de Alchemy, HyperWRT se basa en el código fuente del firmware de Lynksys, pero sus desarrolladores centraron su foco en optimizar dicho firmware, y no tanto en extender sus funcionalidades. Por esta razón es considerado el firmware desarrollado por terceros más recomendado para principiantes, ya que su conjunto de funcionalidades se encuentra limitado a aquellas más básicas de los routers, siendo más fácil configurarlo y adiminstrarlo que otros firmwares de su clase. Algunas de sus prioridades en cuanto a las funcionalidades a agregar eran incrementar la potencia de la antena, triggers de puertos, conexión por telnet, y scripts de configuración.

Sin embargo, esta versión se encuentra discontinuada desde el 2006. Su sucesor directo es Tomato, en desarrollo desde el 2008 a la fecha, y soporta una variedad mucho más amplia de equipos. Entre sus funcionalidades se destacan una interfaz gráfica mejorada, basada en AJAX y SVG, clasificación del tráfico en 10 tipos, almacenando y mostrando estadísticas, y permitiendo controlar el ancho de banda de los clientes a partir de estos datos, soporte para múltiples modos de red inalámbrica, ajuste de potencia de transmisión, selección de antenas y canales, y permite mostrar la intensidad de la señal de los dispositivos inalámbricos conectados.

4.2.4. DD-WRT

DD-WRT nace como alternativa gratuita a Alchemy, cuando sus desarrolladores deciden cobrar por el producto. Ofrece un conjunto de funcionalidades más amplio que HyperWRT/Tomato, y existe una comunidad dedicada a extenderlo y actualizarlo; pero no es tan flexible como OpenWRT, ya que las modificaciones implican volver a compilar el firmware, por lo que se lo considera un firmware para usuarios de nivel medio para su uso y nivel avanzado para modificarlo. Se basaba en el propio código fuente de Alchemy, tomado del repositorio público de la última versión gratuita, pero a partir de la versión 23 comenzó a basarse en OpenWRT. Al día de la fecha se encuentra vigente, y es mantenido por un desarrollador que se denomina Brain Slayer.

Contiene funcionalidades como WDS, controles avanzados de calidad de servicio para la asignación de ancho de banda, control de potencia, etc., pero, si bien ofrece soporte para más de 200 dispositivos, no se encontró que soporte el Mikrotik RouterBoard 133 en su wiki ni en la base de datos de routers de su web.

4.2.5. OpenWRT

Como en el caso de HyperWRT y Alchemy, OpenWRT nace como un firmware alternativo para el router WRT54G de Linksys, basado en su código fuente, pero que rápidamente comenzó a soportar otros dispositivos, y a diferencia del resto de los proyectos, luego se comenzó a desarrollar el producto desde cero, si bien se seguía utilizando el código original de Linksys como referencia.

Todos sus componentes fueron optimizados en cuanto a tamaño y uso de recursos para poder ejecutar en hardware limitado, e incluso varias de sus funcionalidades han sido agregadas a la línea base del kernel de linux. Dispone de cerca de tres mil quinientos paquetes de instalación de software mediante el manejador de paquetes opkg, y es el que permite mayor nivel de customización, por lo que se lo considera para usuarios con nivel de expertise medio a alto, aunque a la hora de extenderlo puede resultar más accesible que otros de su tipo, ya que no necesariamente requiere compilar el firmware completo para agregar una funcionalidad.

Entre sus funcionalidades destacadas se encuentran el ajuste del tamaño del tráfico para mejorar la calidad del servicio, configuración extensible de todos los drivers de hardware, un conjunto de scripts para unificar y simplificar la configuración del sistema, balanceo de carga, AQM (Active Queue Management) a través del scheduler de red del kernel de linux, monitorización y estadísticas de la red en tiempo real, extensibles a través de herramientas como RRDtool, collectd, nagios, etc., y la implementación de un file system con posibilidad de escritura. Esta última funcionalidad sobresale frente a los otros firmwares del mismo tipo, ya que en caso de querer modificar los contenidos de su memoria no será necesario flashear el dispositivo, y a su vez habilita el uso del manejador de paquetes para instalar nuevas herramientas.

Dadas sus características, OpenWrt resulta la mejor opción para construir una aplicación sobre un router inalámbrico sin tener que construir un firmware para ello.

4.2.6. X-WRT

El proyecto X-WRT es un conjunto de paquetes y parches para OpenWRT que ofrecen acceso a las herramientas de configuración del router mediante una interfaz gráfica web, en lugar de la interfaz por consola que traía OpenWRT originalmente, o la interfaz web actual (LuCi) basada en el lenguaje de programación Lua.

4.2.7. Tarifa

Tarifa es un proyecto basado en la versión de serie del WRT54GL de Linksys, con un alcance menor en comparación con los anteriores, ya que se presenta como un reemplazo para el firmware

de dicho router con mejoras de velocidad, alcance y arreglos de bugs. Permite aumentar el rango de la señal, y agrega otras funcionalidades, como aumentar la cantidad de conexiones simultáneas soportadas y mejoras de performance en el uso de la NAT-Firewall.

4.2.8. Elección del firmware

Se eligió OpenWRT como firmware a utilizar en el proyecto debido a las siguientes razones:

- El gran apoyo que tiene de la comunidad, con una gran cantidad de paquetes de diferentes tipos de funcionalidades, información y guías tanto para la instalación como para el desarrollo sobre el mismo.
- Posee file system, por lo cual no hay necesidad de trabajar sobre el firmware y re-compilarlo cuando se quieren agregar nuevas funcionalidades, sino que se pueden agregar paquetes directamente como en cualquier distribución de Linux.
- Es abierto y gratuito.
- Es frecuentemente utilizado en el grupo de investigación "MINA", lo que simplifica la extensión futura de la herramienta a realizar, y puede ser de ayuda en la elaboración de la misma.
- El hardware con el que se cuenta soporta OpenWRT.

4.3. Estado del arte en lenguajes de programación para sistemas embebidos

La presente sección se basa en los libros "Programming Embedded Systems" [57], "Embedded Systems Design" [58], "Programming Embedded Systems in C and C++" [59], "Embedded Systems Specication and Design Languages" [60], en los artículos "Design languages for embedded systems" [61] y "Models of computation and languages for embedded system design" [62], y en los sitios [63], [64], [65], [66] y [67].

4.3.1. Definición de sistema embebido

Un sistema embebido es un conjunto de hardware y software diseñado para realizar una función determinada. Este tipo de sistemas se encuentra presente continuamente en nuestras vidas. La mayoría de las veces son utilizados sin percatarnos de su existencia, sin saber que algo que se

utiliza o algo que se realiza sucede gracias a que detrás existe un conjunto de piezas de hardware y software que lo hacen posible.

Por la propia definición, este tipo de sistemas se distingue de las computadoras (en el término común de la palabra), las cuales no están diseñadas para realizar una función específica, por lo cual es claro que la forma de diseñar estos sistemas no es igual a la forma de diseñar una computadora.

Es común que estos sistemas formen parte de un sistema más grande. Un ejemplo claro es un auto moderno, que posee diferentes sistemas embebidos (radio, control de frenos, etc.), pero también este sistema mayor, al cual pueden pertenecer, podría ser una computadora, en los cuales tanto un teclado, joystick o mouse son sistemas embebidos (poseen un procesador y un software para realizar ciertas funciones específicas).

En la mayoría de estos sistemas es muy importante el enlace con el mundo físico. Es decir, que toman datos, censan, o interactúan de alguna forma con el mundo físico que los rodea.

4.3.2. Evolución de los sistemas embebidos

En 1971 Intel presentó el primer microprocesador de un solo chip (es decir, integrado físicamente en una misma unidad [68]), el 4004, diseñado para ser usado en calculadoras de la empresa Busicom. Intel creó este chip de propósito general en lugar de diseñar un hardware específico para cada tipo de calculadora, diseñado para leer y ejecutar una serie de instrucciones, lo que sería el software, guardadas en una unidad de memoria. A partir de ese momento se comenzó a utilizar cada vez más este tipo de microprocesadores.

En los 80 y 90 los microprocesadores comenzaron a estar presentes cada vez más en la vida cotidiana. Actualmente están presentes en todos lados, en los hogares, empresas, centros de estudio, en la calle, etc. Cada año se crean seis mil millones de nuevos microprocesadores, de los cuales menos del 2 % son usados en computadores de propósito general. Por lo tanto, es clara la cantidad de sistemas embebidos que van surgiendo año tras año. Además de la cantidad que va en aumento, cada vez surgen más tipos de sistemas embebidos, con nuevos objetivos, como sistemas de domótica, monitores médicos, etc.

4.3.3. Áreas de aplicación y ejemplos

Se presenta a continuación un listado de diversas áreas donde existen sistemas embebidos y sus aplicaciones. Evidentemente el listado deja afuera muchos de estos sistemas, ya que dada la cantidad que existen sería imposible incluirlos a todos.

- Medios de transporte: en todos los vehículos modernos y no tan modernos se encuentra una gran cantidad de sistemas embebidos: control del airbag, control de encendido, ABS (sistema anti-bloqueo de ruedas), ESP (control de estabilidad), aire acondicionado, GPS, entre otros. En los aviones se encuentran en el sistema de control de vuelo, sistema anti-colisión, información al piloto, etc. En el resto de medios de transporte (con componentes eléctricos) sucede lo mismo.
- Telecomunicaciones: en esta área es bien clara la cantidad de sistemas embebidos existentes. Por mencionar algunos: teléfonos, contestadora y dispositivos manos libres.
- Medicina: cada vez existen más productos de cuidado de la salud que son sistemas embebidos, no solo de monitoreo, sino también de procesamiento información y cuidados, entre otros. Ejemplos muy conocidos de este tipo de dispositivos son los marcapasos o los pulsómetros.
- Seguridad: por ejemplo control de acceso por huella, alarmas, cierres automáticos, etc.
- Electrónica de consumo: equipos de video, audio y entretenimiento.
- Equipamiento de fábricas: un área muy amplia donde se usan todo tipo de sistemas embebidos, donde generalmente es prioritario que el sistema cumpla su función correctamente frente al consumo de energía.
- Casas inteligentes (domótica): usados para incrementar el confort y eficiencia de los hogares. Algunos de ellos pueden ser aire acondicionado, iluminación automática, temporizadores, etc.
- Logística: como semáforos, RFID para control de mercancías, carga y descarga de las mismas.
- Robótica: una de las áreas más tradicionales donde este tipos de sistemas ha sido usado.
- Generación de energía: las diferentes plantas de generación de energía, sin importar de que tipo sean, utilizan sistemas embebidos para su funcionamiento.

Con este listado se quiere mostrar la cantidad y variedad existente de tipos de sistemas embebidos.

4.3.4. Características comunes

De todos los ejemplos mencionados anteriormente es claro que gran parte de estos sistemas deben cumplir ciertas características para poder funcionar.

Confiabilidad: hay que tener en cuenta que están interactuando directamente con el ambiente y tienen un impacto sobre el mismos según las acciones que el sistema realice (un controlador de una planta nuclear no puede fallar). La confiabilidad abarca los siguientes aspectos:

- Fiabilidad: la probabilidad de que un sistema no falle.
- Mantenibilidad: la facilidad con la que un sistema puede ser reparado dentro de un determinado plazo.
- Disponibilidad: la probabilidad de que el sistema esté disponible cuando se desee usarlo.
- Protección ("Safety"): la propiedad de que el sistema no causará daño al ser usado.
- Seguridad: la propiedad de que los datos confidenciales sigan siendo confidenciales y que la autenticación esté garantizada.

Eficiencia: no se puede desperdiciar nada. Para esta característica se pueden usar las siguientes métricas:

- Uso de energía: es generalmente un característica muy importante en este tipo de sistemas.
- Eficiencia en tiempo de ejecución: debe explotar todas las ventajas que el hardware pueda poseer.
- Tamaño del código: el código que ejecutará un sistema generalmente debe estar guardado en el mismo, por lo que es importante que ocupe el menor espacio posible.

Trabajo en tiempo real: muchos de estos sistemas deben trabajar (y responder) en tiempo real. El no poder completar determinada acción en cierto tiempo muestra una baja calidad del sistema y puede provocar sucesos graves en el entorno con el que interactúa, dependiendo de la función del mismo.

4.3.5. Componentes de un sistema embebido

Existe un gran variedad de hardware para este tipo de sistemas, por lo cual un software embebido diseñado para cierto sistema difícilmente funcione en otro sistema sin realizarle cambios. Esta variedad se debe a que se optimizan ampliamente las piezas de hardware utilizado para cada sistema, quitando lo innecesario buscando abaratar costos, y compartiendo todos los recursos de hardware posibles.

De todas formas, existen ciertos componentes que están presentes en todos los sistemas embebidos, como son el procesador y el software. Además, las instrucciones que componen este

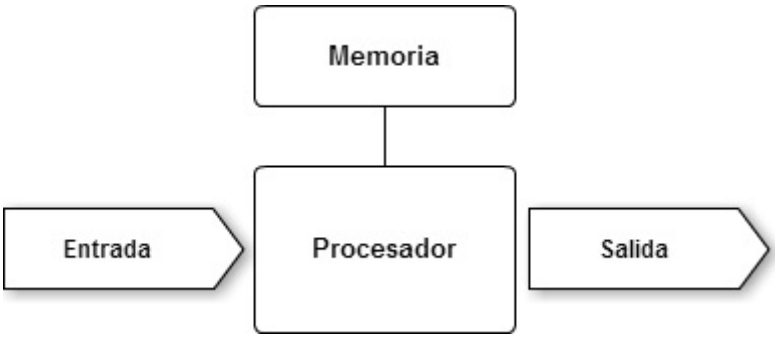

FIGURA 4.1: Sistema embebido genérico.

software deben almacenarse físicamente para poder ser leídas y procesadas cada vez que se utiliza el sistema. Existen diferentes tipos de memoria para este fin, de los cuales la mayoría de los sistemas utiliza los siguientes dos: la memoria ROM (read-only memory) y la RAM (random access memory).. Como se mencionó anteriormente, los sistemas embebidos interactúan con el entorno, para lo cual deben poseer algún tipo de entrada (botones, sensores, etc.) y/o salida (leds, sonidos, etc.). Estas salidas se activan o no en función de las entradas y el software del sistema. Físicamente un sistema embebido se podría representar con el diagrama que se muestra en la figura 4.1.

Dejando de lado los componentes que se han mencionado, el resto del hardware de los sistemas embebidos son particulares para la función que el sistema realiza, lo cual lleva a tener software embebido específico para cada sistema. Para facilitar la tarea de interactuar con cada hardware particular existen los drivers, los cuales son módulos o piezas de software embebido que se encargan de operar e interactuar con cierta pieza de hardware. A su vez, la aplicación (el software principal) que esté ejecutando en un sistema embebido utiliza este driver para interactuar con una pieza de hardware. De esta forma se evita exigir a cada aplicación que implemente la interacción con cada pieza de hardware que deba utilizar.

4.3.6. Requerimientos de diseño

Los sistemas embebidos tienen diferentes requerimientos según cuál sea la función que vayan a desarrollar, y repercuten directamente en el diseño del mismo. A continuación se presentan algunos de estos requerimientos.

De hardware:

- Poder de procesamiento: es decir la carga de procesamiento que podrá manejar. Generalmente es medido en millones de instrucciones por segundos (MIPS).
- Memoria: la cantidad de memoria RAM y ROM que tendrá, lo que repercute directamente en el software que pueda ejecutar.

Criterio	Bajo	Medio	Alto
Procesador	4 u 8 bits	16 bits	32 o 64 bits
Memoria	Menor a 64 KB	64 KB a 1 MB	Mayor a 1 MB
Costo de desarrollo	Menor a USD 100K	USD 100K a USD 1M	Mayor a 1M
Costo de producción	Menor a USD 10	USD 10 a USD 1K	Mayor a USD 1K
Cantidad de unidades	Menor a 100	100 a 10000	Mayor a 10000
Consumo de energía	Mayor a 10 mW/-MIPS	10 a 1 mW/MIPS	Menor a 1 mW/-MIPS
Tiempo de vida	Días a meses	Años	Décadas
Reliatibility	Puede fallar ocasionalmente	Debe funcionar de forma confiable	Debe ser a prueba de fallas

Tabla 4.4: Rangos de valores típicos de requerimientos de diseño.

- Consumo de energía: cuánta energía es usada por el sistema. Este requerimiento es de los más relevantes cuando el sistema es alimentado por pilas o baterías. Generalmente se mide en miliwatts por MIPS (mW/MIPS).

De negocio:

- Cantidad de unidades: es decir, la producción que se espera realizar del mismo.
- Costo de desarrollo: el costo tanto del diseño del hardware como del software.
- Vida útil: cuánto tiempo se espera que el producto se mantenga en uso antes de comenzar a fallar.
- Fiabilidad: que tan confiable debe ser el producto.

En la tabla 4.4, basada en una tabla de [57], se pueden ver los rangos de valores típicos de requerimiento de diseños, las etiquetas bajo, medio o alto son ilustrativas y no debe tomarsen como una medición real.

4.3.7. Desarrollo de software en sistemas embebidos

En la mayoría de los desarrollos de software para sistemas embebidos el desarrollador se encuentra más cerca del hardware que en un desarrollo de software convencional. En el caso de este proyecto no existe la necesidad de un gran contacto con el hardware, ya que se instaló un sistema operativo en el sistema embebido y se trabaja sobre él. De todas formas, tanto en la

instalación de este sistema operativo como en el desarrollo del software realizado se estuvo en mayor contacto con el hardware que en un desarrollo convencional.

En primer lugar, este tipo de software debe ser confiable, por lo que el diseño del mismo debe realizarse teniendo en mente la confiabilidad y los diferentes aspectos que esta contiene. Dependiendo de la función del sistema este punto será más o menos importante.

En parte porque los sistemas embebidos poseen, generalmente, hardware que cumple con lo justo las necesidades del mismo, toma gran importancia el desarrollo de código eficiente. Se debe tener especial cuidado en qué usar y cómo usarlo. Un ejemplo claro es quedarse sin espacio de memoria al querer asignar nueva memoria dinámicamente, por lo cuál el sistema dejará de funcionar. Para solucionar esto muchos sistemas reservan estáticamente todo el espacio de memoria que van a necesitar antes de comenzar su ejecución. Se debe minimizar todo lo posible el uso de los recursos del sistema para no enfrentarse a situaciones en las cuales el sistema deje de funcionar, ya que muchos de esos sistemas no están pensados para ser reiniciados en caso de tener problemas de software, por lo que el código además de eficiente debe ser robusto. También es importante la eficiencia en cuanto al consumo de energía.

Otro aspecto importante es la dificultad en el desarrollo de código reusable. Como ya se mencionó anteriormente, dada la variabilidad de estos sistemas no es sencillo desarrollar código que pueda ser usado en diferente hardware sin recibir modificaciones.

En cuanto a la comunicación con periféricos, puede ser simplificada usando los drivers de los mismos, en caso de que posean. De todas formas es necesario saber cómo comunicarse con cada dispositivo periférico para lograr un correcto funcionamiento del sistema.

En los sistemas de tiempo real es muy importante tener siempre en mente que se deben cumplir los requerimiento de tiempo para el procesamiento o respuesta del sistema.

Las herramientas a las que se puede estar acostumbrado en el desarrollo de un software en un sistema de propósito general no siempre sirven para el desarrollo de sistemas embebidos. Es posible que el debug de un software en un sistema embebido deba realizarse encendiendo leds o activando algún otro dispositivos de salida. De todas formas existen herramientas de bajo nivel que pueden ser de ayuda en este contexto.

4.3.8. Lenguajes en sistemas embebidos

Dadas las diferentes tareas que realizan los sistemas embebidos es difícil pensar en un lenguaje de propósito general que sea óptimo para todos los casos. Existe cierta variedad de lenguajes que han ido evolucionando, con sus ventajas y desventajas, y cada uno se adapta mejor a las diversas situaciones. Elegir el adecuado para el sistema que se vaya a desarrollar es un punto tan importante como complejo.

Se pueden clasificar según la organización de componentes:

- Von-Neumann: basado en la ejecución secuencial de instrucciones, las cuales son secuencias de cálculos más básicos.
- Eventos discretos: en este modelo existen eventos que se van ejecutando en orden. Se basa en la simulación de estos eventos y cómo los mismos se van procesando en el tiempo.
- Máquina de estados finito: está basado en la noción de un número finito de estados, entradas, salidas y transiciones entre estados.
- Flujo de datos: se basa en cómo la información se mueve en el sistema, cómo pasa de componente a componente y la transformación que sufre dentro de cada uno.

Además existen principalmente dos categorías de tipos de comunicación:

- Memoria compartida: los componentes acceden al mismo lugar de memoria para comunicarse. En este tipo de comunicación se debe tener especial cuidado con las modificaciones que se hagan sobre la memoria compartida cuando se quiere escribir y leer de manera concurrente.
- Pasaje de mensajes: la comunicación se realiza mediante el envío y la recepción de mensajes. Este tipo de comunicación es generalmente más lento que el anterior. Se pueden distinguir 3 sub-tipos de comunicación:
  - Pasaje de mensajes asincrónico (o no bloqueante): los mensajes son enviados a ciertos buffers y el emisor no necesita esperar a que el receptor reciba el mensaje ni a que esté disponible para recibirlo, simplemente lo envía. Tiene la problemática de que los buffers tienen un tamaño finito, por lo cual si los mensajes no se retiran a tiempo de los mismos podrían llenarse.
  - Pasaje de mensajes sincrónico (bloqueante o basado en "rendez-vous"): en este tipo de comunicación sincrónica el emisor debe esperar a que el receptor esté pronto para recibir su mensaje. No existe el problema del desbordamiento del buffer, pero la performance se ve reducida.
  - Invocación remota (o "rendez-vous" extendido): es similar al tipo anterior, con la diferencia de que el emisor debe esperar por un aviso por parte del receptor de que el mensaje ha sido recibido (o procesado) para poder continuar.

Estas dos formas de categorizar los lenguajes se combinan para generar la clasificación de los lenguajes que se puede apreciar en la tabla 4.5, basada en una tabla de [58]

	Memoria compartida	**Pasaje de mensajes asincrónicos**	**Pasaje de mensajes sincrónicos**
Basados en Von-Neumann	C, C++, Java, Assembler, LUA, Python	C, C++, Java, LUA	C, C++, Java, CSP, ADA, Python
Eventos discretos	VHDL, Verilog, SystemC		
Máquina de estados finitos	StateCharts	SDL	
Flujo de datos		KPN, SDF	

Tabla 4.5: Clasificación de los lenguajes.

Existe otro tipo de lenguaje, llamado lenguaje sincrónico, que combina máquina de estados finitos con Von-Neumann, pudiendo así expresar cómputos complejos pero siguiendo dentro del modelo de máquina de estados. Ejemplos de este tipo de lenguaje son Esterel, Lustre y SCADE.

Es importante aclarar que la categorización se realiza con la base de cada lenguaje y teniendo en cuenta que en muchos de ellos no es clara la categoría adecuada. La transición entre categorías a veces es sencilla, lo que genera dificultad al momento de clasificarlos.

A continuación se presentará una breve descripción de los diferentes lenguajes.

StateCharts es un lenguaje basado en máquinas de estados que utiliza memoria compartida y soporta jerarquía y concurrencia, lo cual hace posible definir estados dentro de otros estados y tener diferentes sub-estados activos a la vez.

Esterel es un lenguaje "reactivo". Es decir, cuando recibe un evento de entrada se activa y genera una salida. Se asume que estas activaciones con su reacción correspondiente se completan instantáneamente, lo que evita todas las discusiones sobre solapamiento de tiempos de estas activaciones.

Lustre es similar a Esterel, con la diferencia de que Esterel es más orientado a un lenguaje imperativo y Lustre es más orientado a uno de tipo flujo de datos.

SCADE es un lenguaje que combina a los dos anteriores junto a una representación gráfica.

SDL es un lenguaje también basado en máquina de estados finitos, pero utiliza pasaje de mensajes para la comunicación, lo que resulta especialmente útil en sistemas distribuidos donde la memoria compartida no es una opción viable. Se puede trabajar con este lenguaje tanto de forma textual como de forma gráfica. En versiones posteriores del lenguaje se incluyeron características de orientación a objetos.

Kahn Process Networks (KPN) es un tipo especial de lenguaje de flujos, ya que su representación no indica el orden en que se deben realizar los cálculos. La escritura en las colas de mensajes son no bloqueantes. Tiene como desventaja que estas colas de comunicación pueden llenarse y hacer que el sistema entre en deadlock.

En Synchronous Data Flow (SDF) la planificación es mucho más simple y es posible definir restricciones sobre el tamaño de los buffers, podría verse como una restricción sobre KPN. Esto se puede ver en la notación gráfica, donde se aprecia un grafo dirigido. Además, los nodos producen y consumen un número fijo de elementos por ejecución, permitiendo que sea predecible y se optimice al momento de ser compilado.

VHDL modela la concurrencia a través de procesos. Cada uno de estos procesos modela un componente (pieza de HW) de esta concurrencia. Los procesos se comunican a través de señales, las cuales se corresponden con conexiones físicas. A pesar de modelar componentes de HW, VHDL es un lenguaje textual. Describe sistemas con una estructura jerárquica. Estos sistemas están compuestos por bloques que contienen primitivas, otros bloques o procesos concurrentes. La ejecución se activa al producirse cierto evento.

Verilog es similar a VHDL con un poco menos de flexibilidad. Está más orientado a la simulación de circuitos integrados digitales, y no tanto a simulaciones de gran tamaño. Permite al usuario crear nuevas compuertas lógicas usando tablas de verdad.

SystemC proporciona un entorno de simulación de eventos discretos que mezcla fuertemente aspectos de la programación sobre hardware y software. Es una biblioteca de C++ creada para resolver los problemas de concurrencia, tiempo, lógica de múltiple valor y comportamiento determinado, que en ese momento no tenían los lenguajes de software.

CSP (communicating sequential processes) fue de los primeros lenguajes en proponer mecanismos de comunicación entre procesos, los cuales se comunican a través de canales. CSP es determinado, ya que espera por una entrada en un canal para ejecutarse, al igual que en KPN.

ADA es un lenguaje orientado a objetos, en su diseño el objetivo era crear un lenguaje orientado a objetos que se pueda usar para desarrollar software militar de nivel crítico. Una de sus más interesantes características es el poder declarar procedimientos anidados. A pesar de que pasó a ser un lenguaje público, no ha tenido mucho uso fuera de las industrias militar, aeroespacial y aeronáutica.

En Assembler las instrucciones son directamente las mismas que posee el procesador, pero escritas de una forma fácilmente legible por un humano. Cada instrucción se compone de un operador y algunos operandos. Estas instrucciones son ejecutadas secuencialmente, salvo que se indique lo contrario con ciertas instrucciones condicionales o que generan loops.

Un código en C contiene funciones construidas a partir de expresiones aritméticas, loops y condiciones. Cuando en una ejecución se invoca una función, el control de la ejecución pasa a esa función hasta que finalice y retorne al punto donde había sido invocada. C posee tipos predefinidos que están derivados de los que el procesador maneja directamente, y es posible construir nuevos tipos a partir de ellos. Un programa en C utiliza 3 tipos de memoria:

- Datos globales, asignado cuando se compila el software.
- El stack, donde se guardan, borran y modifican automáticamente las variables que se estén usando.
- El heap, donde se puede guardar datos de forma dinámica.

C++ es una extensión de C orientada o objetos, que tiene mecanismos de estructuras para programas grandes, clases, definición de espacios de nombres y manejo de excepciones, entre otros. Además incluye nuevos tipos de datos como arreglos, árboles y strings.

Java es, al igual que C++, un lenguaje orientado a objetos con clases y herencia. A diferencia de los anteriores, Java es totalmente independiente de la plataforma en la que se ejecute. Para esto necesita la máquina virtual de java, un intérprete del código que genera un programa en java. Este código que genera ocupa menos espacio que un código binario. Tiene como ventajas que fue diseñado como un lenguaje seguro, tiene manejo de excepciones y un garbage collector automático. Sin embargo no es un buen sistema para aplicaciones que necesiten ejecutar en tiempo real, ya que necesita del intérprete para poder ejecutar, lo que puede requerir bastante espacio en memoria, no tiene control directo sobre dispositivos de entrada y salida, el garbage collector introduce un costo computacional que puede afectar la ejecución y no se puede planificar cuándo será ejecutado, cuando hay más de un hilo ejecutando no se puede asegurar el orden en que serán ejecutados y típicamente los programas en Java son menos eficientes que en C.

Existe una versión de Java llamada Java Card, que es un versión muy reducida de Java con énfasis en la seguridad, pensada para tarjetas inteligentes. Además existe otra versión, Java Micro Edition (Java ME), que está pensada para sistemas con recursos limitados. Provee interfaces de usuario flexibles, seguridad y manejo nativo de protocolos de red, entre otras características.

LUA es un lenguaje de scripting ligero, que combina una sintaxis procedural simple con una poderosa construcción de descripción de datos, es tipado dinámicamente y corre sobre su propio intérprete. Existen diferentes versiones útiles para sistemas embebidos (Lua VM, LuaJIT, LLVM, entre otras).

Python es un lenguaje interpretado, que soporta programación orientada a objetos y programación imperativa. Posee un tipado dinámico y tiene una implementación del lenguaje escrita en C, CPython, que puede ser usado sobre algunos sistemas embebidos.

4.3.9. Elección del lenguaje a utilizar

Dados el hardware disponible y el firmware elegido, las opciones se reducen a un lenguaje que siga el modelo de Von-Neumann.

En estas condiciones procedimos a comparar los diferentes lenguajes para seleccionar el más indicado. El lenguaje C es de los más usados en los sistemas embebidos. No siempre ha sido así, pero cada vez más C se ha ido convirtiendo en lo más parecido a un estándar para el desarrollo de software embebido. Es interesante saber que C es elegido para desarrollo en sistemas con procesadores que van de 8-bit a 64-bit, sistemas que trabajan con diferentes unidades de memoria (B, KB, MB, etc.) y alcances de proyectos muy diferentes con equipos de una persona a cientos de ellas.

C tiene muchas ventajas, es un lenguaje pequeño y relativamente simple de aprender, existen compiladores de C para prácticamente todo tipo de procesadores y los mismos generan código bastante eficiente, y al programar en C se tiene independencia del procesador que se vaya a usar. La mayor parte de lenguajes de alto nivel también tienen estas ventajas, pero algo que diferencia a C y sin duda es una de sus grandes fortalezas es que es un lenguaje de "bajo alto nivel". Es decir, a pesar de ser un lenguaje de alto nivel, provee un alto grado de control directo del hardware sin perder los beneficios de ser de alto nivel.

En un comienzo Assembler era lo que más se usaba. Es importante recalcar que las instrucciones que se usan en Assembler dependen del procesador sobre el cual se vaya a ejecutar el software, por lo que se dispone de un gran control sobre lo que el procesador va a ejecutar, pero se renuncia a los beneficios de los lenguajes de alto nivel, ya que el código no es reusable entre diferentes tipos de procesadores y no es sencillo de usar.

En el caso de C++, se dispone de las nuevas funcionalidades que este lenguaje aporta en comparación con C, pero, a pesar de ser muy útiles, reducen la eficiencia del código generado. Por lo tanto, C++ es más usado cuando esta reducción de eficiencia es menos relevante que los beneficios que trae C++.

Ada tiene ciertos beneficios que podrían simplificar el desarrollo de software frente a C o C++, pero aún así no ha tenido gran aceptación por los desarrolladores de este tipo de software, sumado a que sería un nuevo lenguaje a aprender.

La principal desventaja de Java es la necesidad de correr sobre su propio intérprete, además de la menor eficiencia en comparación con C.

Lua posee varias ventajas, pero posee cierta complejidad e introduce cierto overhead en la compilación dinámica (JIT compilation).

CPyhton pese a poder ser usado en sistemas embebidos, no es compatible con la gran mayoría.

Por los motivos expuestos anteriormente, por el mayor conocimiento que se tiene sobre C/C++, y al correr sobre OpenWRT (que es ampliamente compatible con C/C++), se decidió utilizar C/C++ para el desarrollo de la herramienta.

4.4. Arquitecturas de monitorización

A continuación se presentan diferentes arquitecturas para monitorización de redes.

- Distribuida - Cada nodo procesa localmente los datos y se basa en la información que le envían otros nodos para tomar decisiones. No hay ningún master.
  - Ventajas
    - No es necesario transferir toda la información a un nodo central para que este ordene al resto qué debe hacer.
    - Los nodos se "conocen" entre sí y podrían tomar decisiones más rápidamente basadas en información de sus vecinos que ya posean.
    - Aunque un nodo falle, el resto sigue procesando.
  - Desventajas
    - Debe existir coordinación entre los nodos.
    - Existe la posibilidad de que un nodo tome una decisión incompatible con la decisión de otro.
    - Los nodos usan procesamiento que dejan de aportar a su función principal.
- Centralizada - Cada nodo simplemente recolecta y envía información a un master, sin realizar ningún procesamiento. El master se encarga de realizar todos los procesamientos y ordena al resto de los nodos qué hacer.
  - Ventajas
    - No van a existir decisiones incompatibles al procesarse todo desde un nodo central.
    - No se necesita coordinación entre los nodos.
    - Los routers se dedican totalmente a su función principal y no utilizan procesamiento para otras tareas.
  - Desventajas
    - Si falla el master se cae el sistema.
    - Mayor carga en el master.
    - Toda la información debe viajar al nodo master.

- Híbrido - Cada nodo hace un pequeño procesamiento y mantiene datos, pero envía gran parte de la información a un master donde se toman decisiones y se comunican al resto de los nodos.
    - Ventajas
        - No hay tanta sobrecarga en el master.
        - Aunque el master caiga se continúa realizando el procesamiento en los nodos.
        - Las decisiones que involucren vecinos directos se pueden tomar en los nodos.
        - Los nodos conocen cierta información sobre sus vecinos.
    - Desventajas
        - Los nodos usan cierto procesamiento que dejan de aportar a su función principal.
        - Debe haber coordinación entre los nodos.
        - Existe cierta carga de información hacia el master.
        - Se tiene mayor complejidad al existir decisiones locales y globales.

La carga de mensaje de red entre las 3 formas es similar.

- En el caso de un arquitectura distribuida se envían muchos mensajes para compartir la información con el resto de los nodos y comunicar las decisiones tomadas.
- En el caso de un arquitectura centralizada hay muchos mensajes para enviar datos propios y recibir órdenes del master.
- En el caso híbrido hay un poco de cada uno de los anteriores.

En este proyecto se optó por utilizar una arquitectura totalmente centralizada. Esta decisión se tomó por simplicidad y por adaptarse a la realidad del proyecto, ya que se quiso disponer de la mayor cantidad de datos posible en una misma base de datos, para lo cual el servidor central recibe todos los datos de los monitores, los procesa y los almacena; no siendo relevantes la toma de decisiones ni las consecuentes modificaciones en el comportamiento de los routers.

Siendo conscientes de las desventajas de este tipo de arquitectura, se asume el riesgo de utilizarla. Para evitar un fallo en el master, controlar la carga sobre el mismo o la carga sobre la red sería recomendable realizar un estudio de carga sobre la infraestructura.

Capítulo 5

Monitorización

5.1. Indicadores de calidad

La presente sección se basa en el libro "Redes de Computadores: Un Enfoque Descendente" [1], en los sitios [69], [70], [71] y [72], en los artículos "Montaje de un router wireless en una maquina gnu/linux" [73], "Is rssi a reliable parameter in sensor localization algorithms - an experimental study" [74], "Assessing link quality in ieee 802.11 wireless networks: Which is the right metric?" [75], "Link-level measurements from an 802.11b mesh network" [76], "Long-distance 802.11b links: Performance measurements and experience" [77], "Análisis de la calidad de señal en una red wifi con la herramienta netstumbler" [78], "Link quality and signal-to-noise ratio in 802.11 wlan with fading: A time-series analysis" [79] y "On the wifi interference analysis based on sensor network measurements" [80], y en la tesis "A measurement-based joint power and rate controller for ieee 802.11 networks" [81].

Uno de los problemas que constantemente afecta la transmisión confiable de las señales en un sistema de comunicación inalámbrico es el deterioro que éstas presentan por causa de factores como la distancia, la temperatura del ambiente, el ancho de banda utilizado, la interferencia electromagnética (producida por rayos, motores, generadores, etc.), obstáculos (como por ejemplo paredes, ventanas, pisos, árboles), entre otros. El deterioro de la señal puede presentarse como una pequeña atenuación o pérdida de potencia debido a la distancia que separa el transmisor de su receptor, o por causa de señales eléctricas aleatorias que se encuentran en el ambiente, las cuales se conocen como ruido. Es por ésto que es muy importante conocer y analizar la cantidad de ruido e interferencia que introduce el medio de transmisión a la señal, ya que si el medio presenta demasiado ruido la comunicación no será eficiente ni confiable.

Según el estándar IEEE 802.11 [82], se define un canal como una instancia de medio de comunicaciones para el propósito de pasar unidades de datos de protocolo (PDU) entre dos o más

estaciones (STA); y se define la separación entre canales como la diferencia entre las frecuencias centrales de dos canales no solapados y adyacentes del transmisor de radio.

Teniendo en cuenta estos conceptos, es importante remarcar que en el contexto de las comunicaciones inalámbricas las transmisiones realizadas en canales adyacentes se solapan y causan interferencia entre ellas, resultando en un pobre rendimiento de la red. A su vez, las señales no solo pueden provenir de redes cercanas, sino que también pueden originarse en fuentes externas tales como monitores para bebés, teléfonos inalámbricos, hornos de microondas, etc.

Para la banda de 2.4 GHz existen 14 canales separados por 5 MHz, para los cuales en cada país y zona geográfica se aplican sus propias restricciones sobre el número de canales disponibles. Por ejemplo, en Uruguay se utilizan los 11 primeros, así como en Norteamérica, mientras que en Europa se dispone de 13, y en Japón se utilizan todos. Esta distribución introduce un problema, dado que cada canal necesita 22MHz de ancho de banda para operar, y es que se produce un solapamiento de varios canales contiguos.

Al existir gran cantidad de puntos de acceso y clientes utilizando esta banda se produce un fuerte impacto en el rendimiento de la red inalámbrica producido por la interferencia que generan, que si bien es producida por todos los dispositivos, los que tienen mayor incidencia son los APs, por lo que manejar su solapamiento resulta un desafío fundamental para mejorar la performance de la red. Aquí aparece un concepto importante a tener en cuenta: el solapamiento.

Al distanciarse por 5 MHz los canales contiguos, y al utilizar 22 MHz de ancho de banda cada uno, sucede que cada canal se solapa con los 4 canales sucesivos contiguos a sí mismo hacia cada lado. Por ejemplo, el canal 1 se superpone con los canales 2, 3, 4 y 5, y el 6 con los canales 2, 3, 4, 5, 7, 8, 9 y 10. Como los dispositivos que emiten en rangos de frecuencias solapados pueden generar interferencias, es importante seleccionar el canal que menos se superponga con los puntos de acceso cercanos, de forma de no deteriorar la calidad de señal tanto en nuestro dispositivo como en los que se encuentran dentro de nuestro rango de alcance de señal.

Para poder determinar la calidad de un enlace es necesario definir indicadores que permitan conocer el estado del mismo, para luego tomar las mediciones correspondientes y así obtener el nivel de la señal recibida entre los nodos que conforman el enlace. Una vez que conocemos estos valores, podremos optar por reaccionar ante la situación que se presenta, como puede ser cambiar el canal en el que se opera, aumentar la potencia del emisor en caso de que la intensidad de la señal que llega al receptor sea muy débil, o reducirla en caso de que la señal que llega al receptor sea buena pero se esté causando interferencia en otros dispositivos de la red. A continuación se presentan los indicadores de calidad de señal más relevantes.

En comunicaciones analógicas y digitales la relación señal/ruido, comunmente escrito S/N, o SNR por su sigla en inglés, es una medida de la intensidad de la señal con respecto al ruido de fondo. La relación se mide en decibelios (dB). Si la intensidad de la señal entrante en microvoltios

es Vs, y el nivel de ruido, también en microvoltios, es Vn, entonces la relación señal/ruido en decibelios está dada por la fórmula S/N = 20 log10 (Vs / Vn). Si Vs = Vn, entonces S/N = 0. En esta situación la señal es casi ilegible, debido a que el nivel de ruido compite fuertemente con ella. En las comunicaciones digitales, ésto probablemente provocará una reducción en la velocidad transmisión de datos debido a errores frecuentes que requieren que el emisor vuelva a enviar algunos paquetes de datos. Idealmente, Vs es mayor que Vn, de modo que S/N sea positivo.

Se puede determinar la calidad de señal de la conexión observando la relación señal-ruido (SNR) de la red inalámbrica. Esta métrica compara la intensidad de la señal con los niveles de ruido existentes para ofrecer una estimación más precisa de la calidad de la conexión. A modo de ejemplo, si consideramos un valor SNR menor a 25 dB y una intensidad de señal menor al 40 % (-70 dBm) es probable que se trate de un problema de alcance o bloqueo de señal; mientras que si el valor SNR es menor a 25 dB y el nivel de ruido es mayor al 10 % (-90 dBm) es probable que se trate de un problema de interferencia ocasionada por otros dispositivos.

Lógicamente, esta métrica está muy relacionada con el Bit Error Rate (y por lo tanto al Packet Error Rate), ya que, al no distinguirse claramente la señal del ruido, se introducen posibles fuentes de error en la lectura de los paquetes, mientras que en el caso opuesto se reducen las probabilidades de que esto suceda. Sin embargo, como se indica en [75], calcularlo genera mucho overhead, por lo que, en caso de no estar disponible de forma nativa por parte de la placa de red, puede generar un impacto muy negativo en la performance del router. A su vez, este indicador varía según la tasa de transmisión, lo que sugiere utilizar un algoritmo de adaptación de tasa de transmisión en función de los valores de SNR registrados.

Un indicador muy similar al SNR es el SINR, que mide la intensidad de la señal frente a la suma de interferencia más el ruido. En este proyecto, como es demasiado costoso en la práctica (cuando no es imposible) identificar la interferencia para separarla del ruido, se considerará la interferencia como ruido. De esta forma, SNR y SINR serán considerados el mismo indicador, al cual nos vamos a referir como SNR.

A su vez, RSSI (Received Signal Strength Indicator) se define como diez veces el logaritmo del cociente de la potencia de señal recibida y una potencia de referencia. Por lo tanto, el RSSI es directamente proporcional a 10 Log P/Pref, lo que implica que mantiene una relación de proporcionalidad directa con la potencia de la señal. Ésto lo vuelve un indicador muy relevante a la hora de medir la calidad de señal, pero presenta un par de desventajas: La primera es la forma en que se toma el indicador, que se realiza en el preámbulo de la recepción de un paquete PLCP (Physical Layer Convergence Protocol, que es un protocolo especificado dentro de la capa de convergencia de transmisión que define exactamente cómo se formatean los elementos dentro de un flujo de datos para un tipo particular de instalación de transmisión) y en la lectura de sus headers, y no en el período de recepción de los datos de dicho paquete, por lo que no nos permite

conocer la potencia de la señal promedio de la recepción del paquete completo. Y la segunda desventaja es que no es capaz de detectar interferencias destructivas en el medio, lo que puede explicarse por el hecho de que las medidas sólo se toman cuando se recibe con éxito el header PLCP; de esta forma, no existen medidas de RSSI para los casos en los que falla la recepción de un paquete. Para complementar esta métrica puede utilizarse la medición del "silence value", que refleja la potencia total de la señal observada antes de que comience un frame.

Otro indicador de la calidad de la señal se puede obtener a partir de los estados de LPI (Low-Power Idle), que indican:

- Cuando la radio se encuentra transmitiendo activamente (TX)
- Cuando el mecanismo de detección de paquetes/señal provoca un intento de demodulación (RX)
- Cuando hay suficiente energía de la señal en banda en el medio, pero el mecanismo de detección de paquetes no se activa (BUSY). Ésto puede suceder debido a la interferencia, por ejemplo, puede ser provocada por una actividad fuera de banda, tal como un horno de microondas, una transmisión Bluetooth o transmisiones de paquetes IEEE 802.11 en los canales adyacentes.
- Cuando no aplica ninguno de los estados anteriores (IDLE).

Como el estado LPI sólo puede encontrarse en uno de éstos a la vez, es posible diferenciar el tiempo libre del tiempo ocupado, e incluso distinguir el tiempo que se reciben señales "limpias" que activan intentos de demodulación del tiempo que se reciben señales con interferencia. Para este trabajo, es interesante distinguir el tiempo que se insume en demodular paquetes (RX), aquel en el que se percibe señal pero no alcanza para activar el mecanismo de detección de paquetes (BUSY), y lo que se conoce como Transmission Opportunity, que es el tiempo en el que efectivamente se está transmitiendo más aquel en el que se podría estar transmitiendo (TX + IDLE). De esta forma se pueden obtener estadísticas que brindan indicios sobre la saturación del medio, lo cual posibilita tomar decisiones sobre cómo reaccionar ante la realidad que se presenta.

Siguiendo en esta línea, pero un nivel más alto dentro de la capa de enlace, es posible obtener estadísticas sobre las tramas que llegan al receptor, clasificándolas en tramas de datos, tramas de gestión y tramas de control. Esto es útil para conocer el nivel de overhead que existe en el medio, el cual disminuye la calidad de la señal recibida. Para lograrlo es necesario realizar sniffing en la red, configurando el adaptador en modo promiscuo, e inspeccionar sus headers para discernir a qué categoría pertenecen. Una vez que se identifica la trama recibida, interesa conservar su tipo (si se trata de una trama de control, o de gestión de qué tipo específico es) y un

timestamp de cuando fue recibida, además de mantener contadores por tipo de trama recibida. Estas estadísticas, junto a los tiempos registrados, sirven para conocer el nivel de uso de la red a lo largo de la jornada, lo que permite no sólo reaccionar frente a la situación actual, sino que además, permite planificar la configuración de los puntos de acceso de la red para brindar una mejor calidad de servicio según los requerimientos en horarios específicos.

Por otro lado, pueden utilizarse indicadores de calidad de la conexión para tener una noción de la calidad de señal recibida. Este tipo de métricas no brinda información directa sobre la calidad de la señal, pero, al depender tan fuertemente de ella, nos permite conocer su estado desde un punto de vista más próximo a la experiencia de los usuarios de la red. Por ejemplo, utilizando la técnica de ping flooding se puede medir la latencia entre el router y los clientes, y obtener el porcentaje de paquetes perdidos, duplicados y desordenados. Si bien para obtener estos indicadores se puede recurrir a métodos que generan más overhead en la red, pueden ser útiles para casos en los que no es posible obtener indicadores de las formas mencionadas anteriormente.

5.2. Herramientas de análisis existentes

Esta sección se basa en los artículos “Herramientas de monitorización y análisis del tráfico en redes de datos” [83] y “A perspective on trafic measurement tools in wireless networks” [84], y en el sitio [85].

Un sniffer o analizador de paquetes es un programa que captura (y opcionalmente analiza) paquetes de red, los cuales no siempre tienen por qué estar dirigidos al equipo donde el programa está ejecutando. Estos analizadores tienen diversos usos, desde simplemente almacenar el tráfico existente hasta monitorizar redes en tiempo real para detectar fallos o ataques (incluso para poder realizarlos).

Actualmente existen multitud de programas de este tipo. La mayoría de ellos tienen en común ciertas funcionalidades:

- Capturar tráfico de redes LAN cableadas e inalámbricas.
- Capturar tráfico a través de las interfaces de red que el equipo posea (en modo promiscuo permite capturar todo el tráfico de la red sin importar el destinatario).
- Posibilidad de examinar, guardar y cargar capturas de paquetes en diferentes formatos.
- Interpretar el funcionamiento de diferentes protocolos utilizados en la red (DHCP, TCP, HTTP, etc.).

- Aplicar filtros sobre los paquetes a capturar o a mostrar.
- Calcular estadísticas y representar gráficos sobre diferentes indicadores (velocidades, paquetes enviados, recibidos, etc.).
- Generar reportes en tiempo real.
- Configurar alarmas que notifiquen ante cierto evento.
- Obtener información de los nodos presentes en la red (MAC, IP, sistema operativo, fabricante).
- Reconstruir sesiones TCP.

A continuación se presenta un breve análisis de las diferentes herramientas existentes.

Tcpdump

Es un software libre y de código abierto que se ejecuta sobre la consola. Permite interceptar y visualizar paquetes TCP/IP (entre otros) que estén viajando en la red a la cual el equipo esté conectada, ya sea inalámbrica o cableada. Posee versiones para Linux, Windows y OS X entre otros.

Wireshark

Es uno de los analizadores más utilizado, además de ser de los más completos. Al igual que tcpdump, permite capturar los paquetes de red. Muestra con gran nivel de detalle de cada paquete. Posee interfaz gráfica y gran posibilidad de aplicar filtros. Puede reconstruir y presentar flujos TCP e interacciones completas. Está desarrollado bajo licencia GNU y posee versiones para la mayoría de los sistemas operativos actuales.

CommView

A diferencia de las anteriores, esta herramienta es comercial y está diseñada para obtener una completa imagen del flujo de tráfico de redes LAN cableadas y para reconstruir las sesiones fácilmente. Posee un analizador para tráfico VoIP. Corre en Windows y su interfaz gráfica es fácilmente manipulable. Existe una versión para monitoreo del tráfico en un equipo de forma remota.

Kismet

Es usado como sniffer y como IDS (sistema de detección de intrusos) para redes 802.11 a/b/g/n. Trabaja con tarjetas que soportan modo monitor. Es posible agregarle plugins para incrementar sus funcionalidades, permite detectar redes escondidas y soporta la mayoría de los sistemas operativos.

NetworkMiner

Esta herramienta, únicamente para Windows, está enfocada en el análisis forense, e intenta obtener toda la información posible sobre los hosts presentes en la red (cableada o inalámbrica). Su interfaz gráfica presenta una sencilla y a la vez completa forma de visualizar la información obtenida. Puede obtener una gran cantidad de información sobre cada host, y posee una versión gratuita con las funcionalidades mencionadas y una versión comercial que extiende las mismas, permitiendo exportación de resultados a CSV y Excel, coloreo de hosts y geolocalización IP, entre otras.

OmniPeek

Esta herramienta, comercial, es un analizador de red (cableada e inalámbrica) con una interfaz gráfica sencilla e intuitiva. Posee visibilidad en tiempo real, y mediante la interfaz se puede analizar rápidamente las condiciones de la red para detectar problemas. Puede analizar tráfico de voz y video IP.

Existen otras herramientas que no son analizadores de paquetes, pero ayudan a conocer el estado de la red o redes existentes.

InSSIDer

Es una herramienta para escanear las redes 802.11 presentes. Es de software libre y tiene versiones para Windows, Linux y dispositivos móviles. Recoge información de las redes que detecte como direcciones MAC, SSID, canal, fabricante y tipo de seguridad, entre otros. Puede generar gráficos para mostrar la intensidad y la localización de las diferentes redes y permite filtrar las redes a mostrar.

NetStumbler

Es una herramienta libre, para Windows, que permite la detección de redes LAN inalámbricas que utilicen los estándares 802.11a/b/g. Permite escanear todo el espectro de forma rápida, logrando ver las redes cercanas, los niveles de señal, el SNR, velocidad de transferencia y marcas de los equipos. Posee gráficos con cierta información de las redes, como SNR en tiempo real, y puede detectar la ubicación de antenas inalámbricas direccionales.

Acrylic

Esta herramienta puede realizar un análisis de cobertura y seguridad de redes LAN inalámbricas. Posee una interfaz sencilla e intuitiva donde visualizar todos los dispositivos de comunicaciones existentes, cómo se comportan y cuáles son las opciones de seguridad implantadas. Tiene la funcionalidad de generar informes automáticamente.

Channel ▲	Unique APs seen	Radios assigned	Interfering rogues	
1	64	66	4	Details
6	95	102	6	Details
11	68	62	5	Details
36	38	42	0	Details
44	51	71	1	Details
108	1	1	0	Details
132	1	1	0	Details
149	58	89	0	Details
157	34	35	0	Details
161	2	1	0	Details

FIGURA 5.1: Merkai - Información por canales.

5.3. Merkai - CISCO

Esta sección está basada en los sitios de CISCO Merkai [86], [87], [88], [89] y [90], y en los datasheets [91] y [92].

Merkai es una empresa fundada en 2006 que actualmente forma parte de CISCO, y ofrece productos que se administran completamente desde la nube, incluyendo LAN inalámbrica, switches Ethernet y dispositivos de seguridad. Es una empresa a nivel global con más de veinte mil implementaciones de sus productos en redes de todo el mundo. El objetivo de Merkai es construir redes inteligentes, administrables a través de la nube que simplifiquen las redes empresariales. Se presenta en este proyecto como una alternativa paga similar a lo que se busca comenzar a resolver en este trabajo.

Está enfocado en la administración centralizada de la red, pero también presenta índices y mediciones. Automatiza, optimiza e incrementa la performance en las redes inalámbricas de alta densidad aún sobre condiciones de mucha interferencia. Esto lo realiza utilizando un tercer radio dedicado a esta función que poseen los puntos de acceso de Merkai, con el cual mide la utilización del canal, la fuerza de la señal, el throughput, las señales de los APs que no sean de Merkai y las interferencias que no sean por WiFi. Posee herramientas para analizar en tiempo real el espectro de señales y la utilización de los canales en cualquier punto de la red, de forma de detectar interferencias y actuar en consecuencia. Utilizando todos los datos que obtiene en tiempo real y el histórico adapta automáticamente el canal a usar, el poder de la señal de los APs y la configuración de las conexiones de los clientes, para optimizar y mejorar la performance de la red.

En las figuras 5.1 y 5.2 se pueden apreciar algunas capturas de la información que brinda.

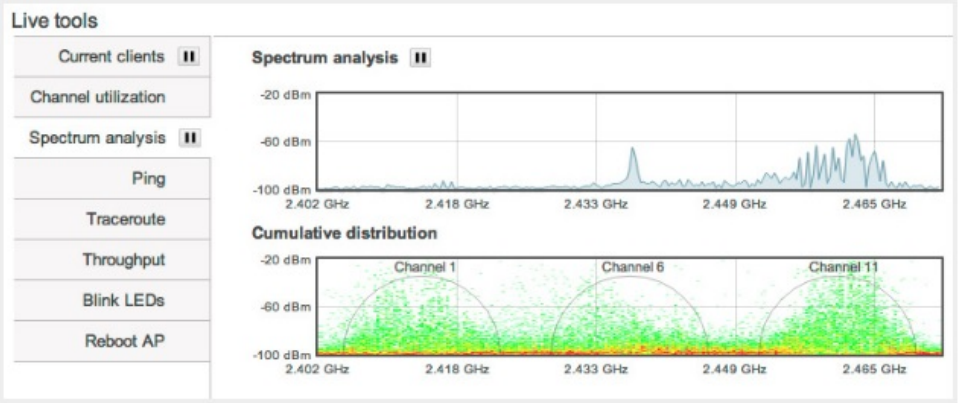

FIGURA 5.2: Merkai - Análisis de espectro en vivo.

Funciona de forma tal que los datos de gestión de la red (configuración, estadísticas, etc.) pasan de los dispositivos Merkai a la nube de Merkai y los datos de los usuarios no van a la nube, sino que van directamente a su destino a través de la red (LAN o WAN según el caso).

En la nube de Merkai se analizan todos los datos y se toman decisiones sobre la red, de forma de evitar tener que configurarla manualmente para adaptarla a las condiciones en las que se encuentra y sacarle el mayor provecho.

También posee herramientas para monitorizar y resolver los problemas que puedan tener la red, los dispositivos o los clientes en cualquier punto de la misma.

Finalmente posee utilidades para analizar el uso que los clientes le dan a la red. Entre ellos se encuentran el tiempo de uso, análisis de permanencia en zonas, ubicar visitantes nuevos y localización actual, con posibilidad de filtros por sistema operativo y hardware entre otros.

Por ser un sistema totalmente cerrado y propiativo no se explica cómo se logra tomar todas estas decisiones ni exactamente qué medidas toman.

Capítulo 6

Desarollo

6.1. Generalidades

Luego del planteamiento del problema y el estudio realizado, se continuó con la intención de desarrollar una herramienta que ayude a atacar el problema planteado.

Esta sección está basada en los sitios de OpenWRT [93], [94], [95], [96], [97], [98], [99], [100], [101], [102], [103], [104], [105], [106], [107], [108], [109], [110], [111] y [112], sobre como instalar OpenWRT [113], [114], [115] y [116], sobre dhcp en ubuntu [117], sobre tftp en ubuntu [118], de kernel.org [119] y [120], de Linux Wireless [121], de iwconfig manpage [122], sobre Kismet [123], [124], [125] y [126], de iw github [127], sobre "wireless tool for linux" [128] y sobre "banwidth monitoring tool for linux" [129], y en los documentos "Kismet Readme" [130], "Step By Step Guide for Starting "Hello, World!" on OpenWRT" [131] y "Linux & 802.11 Wireless" [132].

6.1.1. Acercamiento a OpenWRT y las herramientas disponibles

Se investigó el sistema OpenWRT, las guías básicas de comienzo, la forma de instalar paquetes nuevos y en particular el funcionamiento de las redes inalámbricas sobre este sistema.

La configuración definida para la red inalámbrica se encuentran en el archivo /etc/config/wireless. Cuando se inicia el router por primera vez, el sistema detecta qué tarjeta posee y arma una configuración por defecto para esa tarjeta. Esta configuración posee dos secciones importantes, una referida a la red física, "wifi-device", y otra que refiere a una red virtual que se monta sobre la física, "wifi-iface". A continuación se presentan los campos que contiene esta configuración en la placa Routerboard 133 con la versión Kamikaze del firmware, con un detalle de cada uno (no todos los campos son obligatorios):

```
config wifi-device  Nombre interno del adaptador
  option type       Tipo (broadcom, atheros, mac80211)
  option country    Siglas del país
  option channel    Canal
  option distance   Distancia al cliente mas lejano (1-n, si es soportado)
  option hwmode     Modo de HW (11b, 11g, 11a, 11bg, si es soportado)
  option rxantenna  Indica la antena receptora (0,1,2, si es soportado)
  option txantenna  Indica la antena transmisora (0,1,2, si es soportado)
  option txpower    Poder de transmisión (en dBm)

config wifi-iface
  option network    La interfaz de red a la cual se va a montar la virtual
  option device     Identificador para el hw de radio
  option mode       Modo de la red (ap, sta, adhoc, monitor o wds)
  option ssid       Nombre de la red
  option bssid      Dirección bssid
  option encryption Cifrado (none, wep, psk, psk2, wpa o wpa2)
  option key        Clave de cifrado
  option keyX       Otra clave
  option hidden     Red oculta o visible (0, 1)
  option isolate    Aisla a los clientes entre si (0,1)
```

Se continuó investigando en profundidad el funcionamiento de las redes sobre este firmware.

OpenWRT posee una gran cantidad de paquetes. Nos enfocamos en aquellos que puedan resultar provechosos para el objetivo que se persigue. Entre las herramientas evaluadas se encuentran: “iwconfig” (similar a ifconfig aplicado a redes inalámbricas, mediante el cual se pueden setear diferentes parámetros y sacar datos y estadísticas), “iw” (la cual reemplaza al anterior), “tcpdump” (un capturador de tráfico), entre otras. Una herramienta en particular, “kismet”, resultó muy interesante y más adelante se profundiza sobre la misma.

También se buscó información de sobre cómo realizan sus funciones estas herramientas, revisando sus código fuente, para intentar encontrar que “systemcalls” utilizan o de qué forma obtienen sus datos y poder replicarlo en nuestra herramienta.

Por otro lado se investigó cómo compilar paquetes propios para que ejecuten sobre nuestra placa. Se presentaron diversos problemas a la hora de realizar esta tarea, y no se logró generar un paquete funcional para la versión utilizada de OpenWRT.

6.1.2. Trabajando sobre la Routerboard 133

Se investigó si era posible trabajar en esta tarjeta en modo monitor, para lo que se buscó información sobre el chipset que utiliza, resultando que era posible. Por lo tanto, se comenzó a trabajar sobre la misma, instalando OpenWRT. A continuación se presentan los pasos realizados:

Flasheo con RouterBOOT 2.7

- En primer lugar se instaló la versión 2.7 del firmware RouterBOOT (el cual es un gestor de arranque). La versión instalada se descargó desde http://www.routerboard.com/files/rb-2.7/adm5120-2.7.fwf
- Se estableció una conexión al router utilizando el puerto serial y se creó una configuración para conectarse al mismo. Para esto se ejecutó el comando "minicom -s", dentro de la sección "Serial port setup"se ubicó el device en el que se encuentra el router (con dmesg — grep tty) y se lo seleccionó en "serial device". Luego se configuró el "Bps/Par/Bits" en 1152000 y 8N1 y se guardó la configuración.
- A continuación se inició con la configuración creada: "minicom nombre_archivo_config"
- Se enciende el router y se presiona cualquier tecla rápidamente para entrar al setup.
- Dentro del setup se eligió la opción "g", "Upgrade firmware", luego la "s", "Over serial port". Desde este punto el router se queda a la espera del archivo. Presionando Ctrl + a + z se elige la opción "s", transferir archivo, seleccionando "xmodem" se busca el archivo y se lo envía. Luego de finalizada la transferencia hay que reiniciar el router.
- Finalmente se formatea la nand, con la opción "e" del setup quedando el dispositivo flasheado con RouterBOOT 2.7.

Instalación OpenWRT

- Luego se procedió a instalar OpenWRT realizando un arranque por red. Para esto fue necesario levantar un servidor DHCP y un TFTP.
- Por otro lado se descargó la imagen de OpenWRT para arranque por red openwrt-adm5120-2.6-vmlinux.elf.
- Como servidor DHCP se utilizó dhcp3-server, el cual se descargó y se modificó su configuración (archivo /etc/dhcp/dhcpd.conf):

```
subnet 192.168.100.0 netmask 255.255.255.0 {
    range 192.168.100.2 192.168.100.250;
```

```
        option routers 192.168.56.1;
        option broadcast-address 192.168.100.255;  default-lease-time 600;
        max-lease-time 7200;
    }
```

- Se reinició la interfaz /etc/init.d/isc-dhcp-server restart
- Se cambió la configuración de red del sistema donde estaba ejecutando (Ubuntu) y se deshabilitaron otras interfaces de red existentes.
- A continuación se levantó un servidor TFTP. Para esto se utilizó xinetd (sudo apt-get install xinetd tftpd tftp -y), se creó el fichero /etc/xinetd.d/tftp, donde se guardó la configuración del mismo (fue necesario mantener el nombre "tftp"):

```
service tftp{
    protocol = udp
    port = 69
    socket\_type = dgram
    wait = yes
    user = nobody
    server = /usr/sbin/in.tftpd
    server*args = var/lib/tftpboot -s
    disable = no
}
```

- Se creó el directorio /var/lib/tftpboot y se le asignaron los permisos necesarios (sudo chown -R nobody:nobody /var/lib/tftpboot y sudo chmod -R 777 /var/lib/tftpboot)
- Finalmente se reinició el servicio para que se apliquen los cambios (sudo service xinetd stop y sudo service xinetd start). Es importante destacar que una vez el archivo a transferir esté en el directorio hay que darle permisos al mismo.
- Teniendo ambos servidores levantados se modificó la configuración del DHCP para que el router pueda tomar el archivo de imagen del firmware. Para esto se agregó:

```
host router{
    hardware ethernet [MAC\_ROUTER];
    filename "/openwrt-adm5120-2.6-vmlinux.elf";  // Imagen de OpenWRT
    fixed-address 192.168.100.3;
}
```

- Se modificó la configuración que ya existía previamente agregando la dirección IP del servidor TFTP

```
next-server 192.168.100.5;
```

- Se conectó el router por red cableada y se configuró para que arranque por red (opción "e"), luego se reinició (previamente se verificó que los servidores dhcp y tftp estuvieran levantados y funcionando correctamente) obteniendo así el router ejecutando OpenWRT pero con un inicio por red. Aún no está instalado.
- Se configuró la red desde el router para poder acceder a la pc donde se tiene el OpenWRT completo (openwrt-adm5120-rb1xx-kernel y openwrt-adm5120-router_le-rootfs.tar.gz). Para esto se ejecutó ifconfig eth0 192.168.100.5.
- Utilizando scp se obtuvieron ambos archivo y renombrándose la imagen del firmware a "kernel".

  ```
  scp pgmonitoreo@192.168.100.5:/home/pgmonitoreo/nombre\_archivo .
  ```

- Finalmente, con los archivos en el router, se realizaron los pasos adecuados para la instalación de OpenWRT:

  ```
  mount /dev/mtdblock2 /mnt
  cd /mnt
  copiar kernel a /mnt
  cd /
  umount /mnt
  mount /dev/mtdblock3 /mnt
  cd /mnt
  copiar openwrt-adm5120-router\_le-rootfs.tar.gz a mnt
  tar zxvf openwrt-adm5120-router\_le-rootfs.tar.gz
  rm openwrt-adm5120-router\_le-rootfs.tar.gz
  cd /
  umount /mnt
  sync
  reboot
  ```

- Entrando nuevamente al router, se cambió la configuración para que bootee desde nand, se reinició y quedó instalado y funcionando OpenWRT.

Luego de instalar el firmware, se comenzó a trabajar sobre él, investigando diferentes herramientas y capacidades del mismo.

A grandes rasgos no se consiguió un avance significativo, ya que surgieron varias problemáticas que imposibilitaron el avance. Entre ellas, al no poder instalar una versión de OpenWRT

más reciente, ya que la placa no lo soporta, muchas herramientas o features no se encontraban disponibles. Tampoco se logró compilar un paquete para instalar nosotros mismos, y lo más importante, se hizo muy complicado configurar la red de forma correcta. Al cambiar la configuración de las interfaces no era posible quitarle el bridge sin perder la conectividad con el dispositivo por más que según la documentación se estaba configurando de forma correcta. Se revisó también que no se tratara de un problema causado por el de firewall, pero tampoco se tuvo éxito. Al conectarlo a otra pc se intentó cambiar el "default gateway" en la tabla de ruteo, pero tampoco se logró que la red quedara funcional, ni se consiguieron resultados intentando conectar el router como cliente a otro servidor dhcp (ya sea de un router o de un servidor).

Luego de aproximadamente dos meses de intentos fallidos, se decidió hacer un cambio en el hardware utilizado.

6.1.3. Trabajando sobre TP-Link TL-WDR4300

Sobre este hardware se pudo instalar una versión de OpenWRT más reciente, la versión 12.09 "Attiude Adjustment", y se lograron manejar de forma correcta las configuraciones de las redes y demás. Como contrapartida, el mismo no cuenta con las diferentes interfaces inalámbricas que poseía el R133, por lo cual no se va a poder utilizar como AP y monitor a la vez.

Luego de sortear los diferentes problemas que surgen al trabajar con un nuevo hardware, nos enfocamos en la herramienta "kismet". Dicha herramienta es un sniffer, detector de redes y funciona también como IDS (sistema de detección de intrusos), mencionado en el capítulo anterior. Nuestro interés sobre ella se encuentra en los datos que la misma obtiene de la red. Posee 3 partes bien diferenciadas: un cliente, un servidor y uno o más drones:

- Cliente: muestra al usuario la información que la herramienta posee de forma amigable. Corre sobre linux.
- Servidor: es quien recepciona y procesa los datos que envían los drones. Corre sobre linux.
- Dron: se encarga de tomar los datos de las redes y enviarlos al servidor. Corre sobre OpenWRT.

Para hacer funcionar esta herramienta surgieron diferentes obstáculos, tanto en el router como en la PC que iba a oficiar de cliente y servidor. Por el lado del router, no se lograba ejecutar el "kismet dron" de forma correcta. Se instalaron diferentes versiones del mismo hasta que se logró obtener una funcional. En cuanto a la PC, como se estaba trabajando sobre un Ubuntu 11.10 no se logró instalar el servidor sobre esta versión del sistema operativo por problemas de compatibilidad. Se intentaron resolver los errores que presentaba la instalación, pero finalmente

se optó por actualizar la versión de Ubuntu a la 12.04. En esta versión de Ubuntu no se manifestaron los problemas que se presentaban anteriormente, pero aparecieron otros. Finalmente se agregó el repositorio de kismet directamente al listado de Ubuntu y se logró instalar de forma correcta el servidor y el cliente.

Luego de instalar la herramienta se investigó la configuración necesaria para hacerla funcionar. Nos encontramos con el problema de que el dron no enviaba los datos al servidor, a pesar de estar todo configurado como indica la documentación. Luego de probar diferentes configuraciones a partir de información extraída de foros de usuariosse logró hacer que se envíen.

Se analizó la herramienta y se intentó de alguna forma utilizar los datos que posee para nuestro interés, probando con obtener directamente a partir de los datos que envía el dron, leyendo los logs que genera el server, realizando plugins sobre la herramienta, utilizando otra herramienta ("kismon") que lee información del servidor (que resultó que no era lo que precisábamos), pero no se logró obtener la información deseada.

También se investigó su código fuente para entender su funcionamiento y basarse en el mismo para desarrollar nuestra propia herramienta, pero el resultado no fue satisfactorio.

Finalmente se optó por utilizar tcpdump y realizar un parseo sobre su archivo de salida. Más adelante se profundiza en este tema.

6.2. Implementación

A continuación se presenta el desarrollo realizado. El mismo ha sido construido en C/C++. Para el desarrollo se utilizó el libro "802.11ac: A Survival Guide" [133], el documento "Programing with Libpcap - Sniffing the network" [134], el artículo "Plab: a packet capture and analysis architecture" [135] y los sitios sobre tcpdump y libpcap [136], [137], [138], [139], [140], [141], [142], [143], [144], [145], [146] y [147], sobre instalar y programar para OpenWRT [148], [149], [150], [151], [152] y [153], de scp [154], y [155], sobre manejo de señales [156] y [157], de postgre [158], [159] y [160] y sobre 802.11 [161], [162] y [163], [164].

6.2.1. Idea general y arquitectura

La idea es disponer de uno o más routers en modo monitor (los cuales llamaremos "monitores") capturando todos los paquetes e información del uso del aire que puedan obtener del medio. Estos routers cada cierto tiempo envían los datos al servidor, donde se procesan, se almacenan en una base de datos y se presentan en parte al usuario. En la figura 6.1 se puede apreciar la arquitectura del sistema.

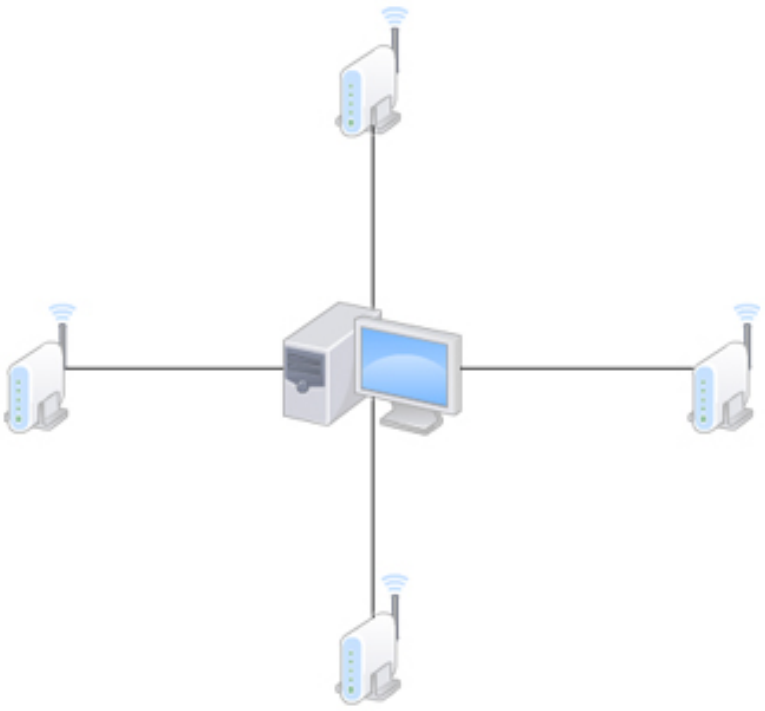

FIGURA 6.1: Arquitectura del sistema desplegado.

Todos los datos obtenidos por los routers se procesan y se guardan en una base de datos, de forma de poder realizar cualquier consulta sobre la misma, cubriendo los diferentes intereses en cuanto a datos o estadísticas que puedan surgir por parte de los usuarios.

Se utilizó el lenguaje inglés en diferentes partes del código y en la base de datos para evitar inconvenientes al utilizar tildes o eñes.

6.2.2. Servidor

El servidor ejecuta sobre linux, se encarga de procesar los datos que los monitores envían, guardarlos en la base de datos y mostrarlos al usuario. También interactúa con el usuario para que el mismo pueda configurar diferentes parámetros, navegar por la información que se presente y realizar ciertas consultas. En la figura 6.2 se puede ver el menú de opciones que ofrece la pantalla principal.

Internamente el servidor posee dos hilos, un hilo principal que se encarga de la presentación de la información e interacción con el usuario, y un hilo secundario que procesa la información recibida por los monitores y la guarda en la base de datos.

La herramienta toma su configuración desde archivo, y sus parámetros de configuración son:

- rutaDirCapturas: ruta donde estarán los directorios con las capturas.

FIGURA 6.2: Pantalla principal, selección de acciones.

- prefijoInfoDron: prefijo para saber si un archivo contiene la información de los paquetes que capturó un monitor.
- prefijoInfoScript: prefijo para saber si un archivo contiene la información que un monitor censó del uso del aire.
- tiempoEntreLecturas: tiempo en segundos entre lecturas de archivos.
- cantidadMaximaPaquetes: cantidad máxima de paquetes que puede existir en un mismo archivo de captura.

Por el lado del hilo que procesa la información, el mismo es básicamente un parser que procesa los archivos enviados por los monitores y actualiza la base de datos. Para el parser se utiliza la biblioteca libpcap (sobre la cual se entrará en mayor detalle en la sección siguiente). Los archivos de datos se organizan dentro del servidor de forma de diferenciar los enviados por cada router, los ya procesados y los que tuvieron algún problema al ser procesados. Esto último con la finalidad de disponder de datos con los que debuguear la herramienta, ya sea con el objetivo de arreglarla o de extenderla para que soporte el nuevo caso.

Para obtener mayor información en cada paquete se utiliza el header “radiotap”. El mismo es un header que se acopla a la trama de enlace y brinda información específica del hardware, obtenida interactuando directamente con el controlador. Por esa razón la información a la que se puede acceder es muy dependiente del fabricante del hardware y del driver.

El servidor utiliza una base de datos relacional para guardar toda la información capturada por los monitores. En la sección siguiente se presentan los detalles de la base.

En el apéndice C se presenta la parte más relevante del código del servidor.

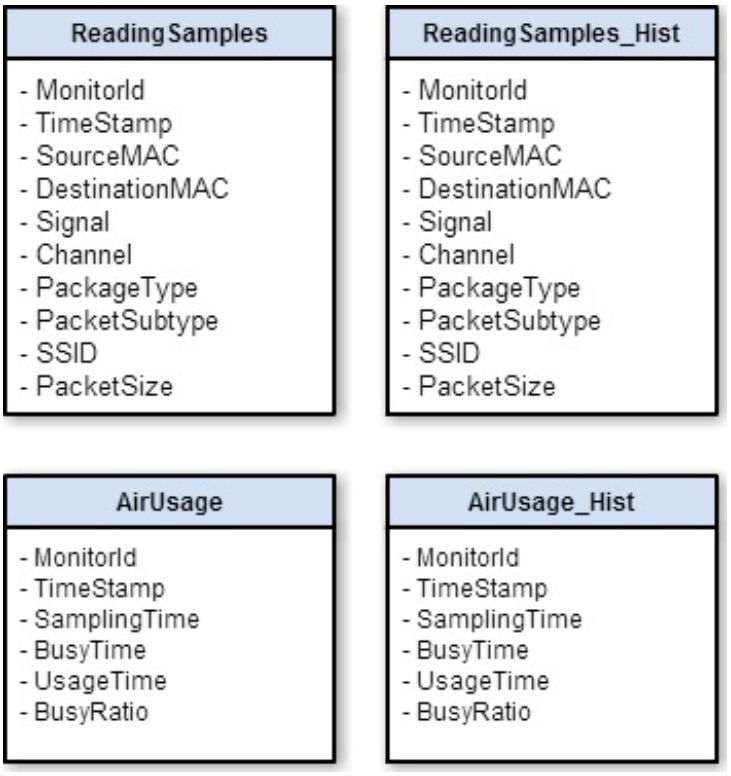

FIGURA 6.3: Diseño de la base de datos.

6.2.3. Base de datos

Respecto a la base de datos, se utilizó el manejador de base de datos "PostgreSQL v9.1" sobre un sistema operativo Ubuntu 12.04. Se creó una base llamada wifi_qos cuyo dueño es un usuario sin privilegios, que es el que se utiliza para conectarse desde el servidor. Los pasos para realizar la instalación y configuración de la base se pueden encontrar en el apéndice B.

Se crearon las tablas "ReadingSamples" y "AirUsage" para almacenar la información correspondiente a la captura de paquetes por parte de la interfaz de red en modo promiscuo y de las estadísticas de tiempos de uso del aire respectivamente. También se cuenta con dos tablas extra ("ReadingSamples_Hist" y "AirUsage_Hist") donde se guarda la información histórica correspondiente a las tablas mencionadas anteriormente. Esto se realizó así para volcar hacia ellas la información que no sea actual, manteniendo todos los datos recabados en diferentes sesiones sin afectar el rendimiento del motor de la base sobre las tablas con datos actuales, que es donde se realizan las consultas que se muestran en la interfaz gráfica. En la figura 6.3 se puede ver el diseño de la base de datos.

La estructura de la tabla "ReadingSamples" es:

- MonitorId: char(17) - MAC del nodo monitor.

- TimeStamp: bigint - Timestamp con precisión en microsegundos de cuando la interfaz de red recibió el paquete.
- SourceMAC: char(17) - MAC del emisor del mensaje.
- DestinationMAC: char(17) - MAC del destinatario del mensaje.
- Signal: smallint - Potencia de señal medida desde el nodo monitor.
- Channel: smallint - Canal de la banda de 2.4 Ghz utilizado en el envío del paquete.
- PackageType: smallint - Tipo de paquete (puede ser administración, control o datos).
- PacketSubtype: smallint - Sub-tipo de paquete (los valores que puede tomar dependen del tipo).
- SSID: char(32) - El SSID del AP con el cual se está interactuando.
- PacketSize: int - Tamaño del paquete en bytes.

La clave primaria de la tabla son los atributos MonitorId y TimeStamp.

La estructura de la tabla "AirUsage" es:

- MonitorId: char(17) - MAC del nodo monitor.
- TimeStamp: bigint - Tiempo con precisión en microsegundos de toma de los datos.
- SamplingTime: int - Duración del muestreo en milisegundos.
- BusyTime: int - Tiempo en que el medio estuvo ocupado en milisegundos (tiempo en que no pudo transmitir si hubiera querido).
- UsageTime: int - Tiempo en que efectivamente estuvo recibiendo paquetes en milisegundos.
- BusyRatio: float - Cociente entre BusyTime y SampleTime.

La clave primaria de la tabla son los atributos MonitorId y TimeStamp.

La biblioteca utilizada en el servidor para conectarse a la base es "libpqxx v4.0".

6.2.4. Interfaz con el usuario

Mediante la interfaz, el usuario puede acceder a ciertos datos en pantalla, los cuales se calculan en tiempo real y corresponden a los últimos sesenta minutos. Los mismos se presentan a continuación.

Información básica de los APs discriminada por el par monitor y AP, la cual contiene:

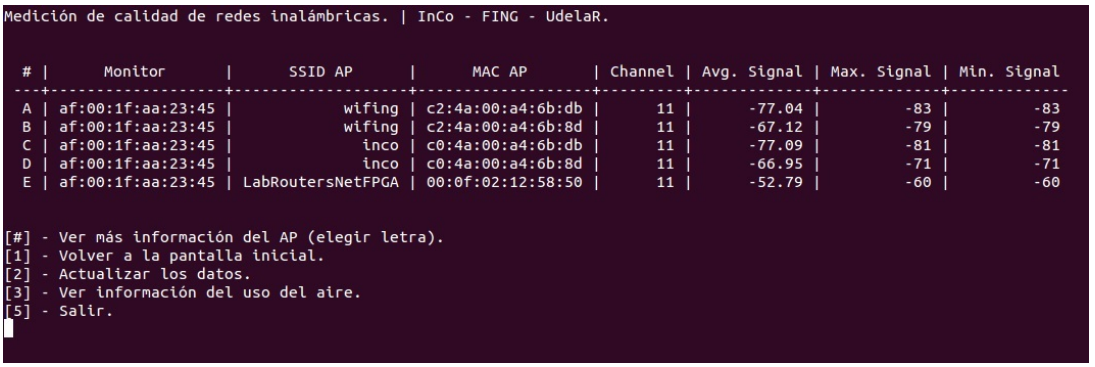

FIGURA 6.4: Presentación de datos de APs.

- Monitor que captó la señal.
- SSID AP.
- MAC AP.
- Canal.
- Señal promedio del AP.
- Señal máxima del AP.
- Señal mínima del AP.

Desde allí, es posible entrar en mayor detalle sobre cada AP, donde se puede apreciar, separado por monitor que captó los paquetes, la siguiente información:

- MAC Cliente.
- Promedio, máximo y mínimo de señal enviado por cliente.
- Cantidad de paquetes y cantidad de bytes enviados por cliente.
- Promedio, máximo y mínimo de señal recibido por cliente.
- Cantidad de paquetes y cantidad de bytes recibidos por cliente.
- Cantidad de clientes.

En las figuras 6.4 y 6.5 pueden verse estos datos.

También es posible acceder a la información del uso del aire, donde se muestra la siguiente información:

- Monitor que realizó la medición.

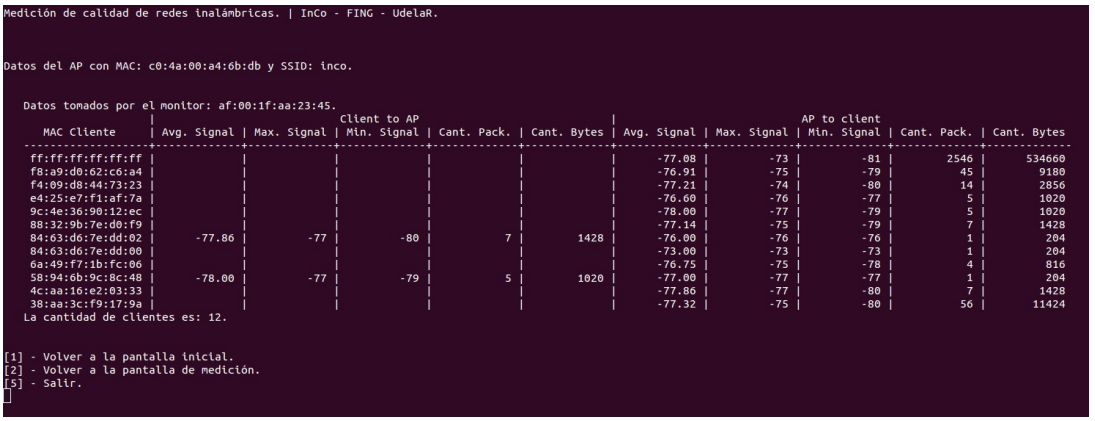

FIGURA 6.5: Información en profundidad de un AP.

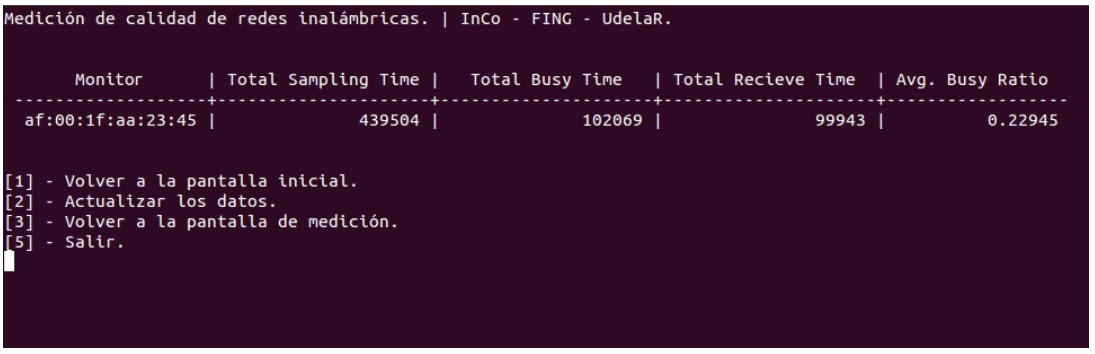

FIGURA 6.6: Datos de uso del aire.

- Duración del muestreo.
- Tiempo ocupado.
- Tiempo de uso.
- Relación de ocupación.

Dicha información puede verse en la figura 6.6.

El usuario puede ver y cambiar los datos de configuración del servidor en tiempo de ejecución. También se puede cambiar la ruta del archivo de configuración de forma de tener diferentes configuraciones ya establecidas y poder cambiar entre ellas fácilmente.

Además de la información mencionada que el usuario puede visualizar a través de la interfaz, la herramienta cuenta con un conjunto de consultas pre definidas que posiblemente sean atractivas para el usuario. Como se mencionó anteriormente, cualquier usuario puede realizar sus propias consultas directamente sobre la base. Las que aquí presentamos fueron seleccionadas por su relevancia y utilidad. Las misas son:

- Información general de APs.

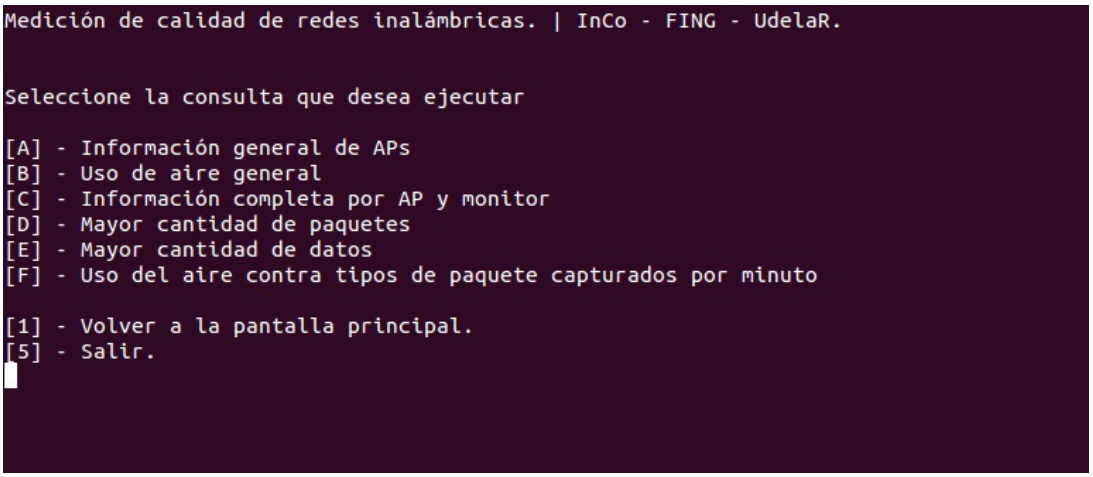

FIGURA 6.7: Consultas establecidas a la BD.

- Uso de aire general.
- Información completa por AP y monitor.
- Mayor cantidad de paquetes.
- Mayor cantidad de datos.
- Uso del aire contra tipos de paquete capturados por minuto.

El detalle de las mismas se encuentra en el apéndice A y una captura de ellas puede verse en la figura 6.7.

6.2.5. Monitor

A la herramienta que ejecuta sobre el router la llamamos monitor (internamente también se le llamó dron). El monitor captura toda la información del medio en modo promiscuo, toma ciertos valores internos del router y envía esos datos al servidor.

Internamente el monitor inicia, carga la configuración, invoca el script "macState.sh", creado por el tutor Matías Richart, y comienza a capturar paquetes en modo promiscuo utilizando libpcap. Cada cierto tiempo envía los archivos que genera al servidor. El script se puede ver en el apéndice C.

El script mencionado lee ciertos valores utilizando la herramienta "iw", para finalmente escribir cada cierto tiempo en un archivo los valores de tiempo activo, tiempo ocupado, tiempo recibiendo y la relación de ocupado sobre el total de la muestra en el propio nodo monitor (estos valores se corresponden con los mencionados anteriormente de la tabla AirUsage). Dado que el nodo monitor está en modo promiscuo, estos valores representan el estado del medio.

Libpcap es una biblioteca de C para captura y análisis de tráfico. En este proyecto se la utiliza para capturar los paquetes y guardarlos a archivo en el monitor. Luego, también se la utiliza para levantar los archivos transferidos al servidor. Comenzar a utilizar libpcap puede resultar complejo, pero a la larga resultó beneficioso.

Los archivos son enviados al servidor utilizando scp. Se configuraron el servidor y los monitores con "ssh-keys" para que no sea necesario introducir credenciales para enviar los datos. Para esto se realizaron los siguientes pasos (ayudados por la herramienta dropbear):

- Se genera una clave en el monitor.

  ```
  dropbearkey -t rsa -f ~/.ssh/id_rsa
  ```

- Se cambia el formato de la clave.

  ```
  dropbearkey -y -f ~/.ssh/id_rsa | grep "^ssh-rsa" >> authorized_keys
  ```

- Se agrega el "authorized_keys" al " /.ssh" del servidor.
- Asegurarse que los permisos del archivo sean 600.

Al igual que el servidor, el monitor posee un archivo de configuración, en el cual se pueden configurar los siguientes parámetros:

- rutaCapturas: ruta donde se dejarán los archivos a enviar.
- prefijoCapturasPaquetes: prefijo de las capturas de paquetes.
- prefijoCapturasMatias: prefijo de las capturas del script MacState.
- rutaScript: ruta de ejecución del script MacState.
- ipServidor: ip a donde mandar los archivos.
- puertoServidor: puerto por donde mandar los archivos.
- rutaServidor: ruta donde dejar los archivos en el servidor.
- usuarioServidor: usuario en el servidor.
- dispositivoID: identificador del dispositivo. Utilizamos la MAC.
- refresco: tiempo máximo en segundo de timeout al inicializar la captura.
- maxPaquetes: cantidad de paquetes máxima capturados por archivo.

```
Herramienta para la captura y envío de datos.

Configuración tomada del archivo: "cliente.conf".

El cliente ejecutará con la siguiente configuración:
  Las capturas se obtienen desde:                "."
  El prefijo de las capturas es:                 "dron"
  El prefijo de las capturas de MacState es:     "script"
  El script MacState se tomará de:               "macState.sh"
  Las capturas se enviarán hacia:                "192.168.10.111:22"
  Las capturas se dejarán en:                    "/home/pgmonitoreo"
  Se utilizará el usuario:                       "pgmonitoreo"
  El tiempo máximo entre capturas es:            "1"
  La cantidad máx de paquetes por captura es:    "1000"
  La cantidad de MacState por archivo es:        "60"
  El ID del dispositivo es:                      "af:00:1f:aa:23:45"
  La interfaz de captura es:                     "wlan0"

  Se invocará el script con el comando: "sh macState.sh script . 60 &"
Presione Ctrl+c para finalizar

Se iniciará el script macState en segundo plano.
Se comenzará con la captura de paquetes y el envío al servidor.
```

FIGURA 6.8: Ejecución del monitor, inicio y configuración.

```
Se envió un archivo al servidor - 05/04/2015 23:44:17
dron_1430783057.pcap                          100%  166KB 165.8KB/s   00:01
Se envió un archivo al servidor - 05/04/2015 23:44:26
dron_1430783066.pcap                          100%  181KB 180.9KB/s   00:00
scriptscript2.csv                             100% 1920     1.9KB/s   00:00
Se enviaron 2 archivos al servidor - 05/04/2015 23:44:43
dron_1430783083.pcap                          100%  157KB 157.2KB/s   00:00
Se envió un archivo al servidor - 05/04/2015 23:44:52
dron_1430783092.pcap                          100%  195KB 194.7KB/s   00:00
Se envió un archivo al servidor - 05/04/2015 23:45:05
dron_1430783105.pcap                          100%  209KB 209.4KB/s   00:00
Se envió un archivo al servidor - 05/04/2015 23:45:16
dron_1430783116.pcap                          100%  175KB 174.7KB/s   00:00
Se envió un archivo al servidor - 05/04/2015 23:45:27
dron_1430783127.pcap                          100%  175KB 175.4KB/s   00:00
Se envió un archivo al servidor - 05/04/2015 23:45:36
dron_1430783136.pcap                          100%  188KB 188.3KB/s   00:00
Se envió un archivo al servidor - 05/04/2015 23:45:48
scriptscript3.csv                             100% 1920     1.9KB/s   00:00
dron_1430783148.pcap                          100%  160KB 159.5KB/s   00:00
Se enviaron 2 archivos al servidor - 05/04/2015 23:46:04
dron_1430783164.pcap                          100%  181KB 181.2KB/s   00:00
Se envió un archivo al servidor - 05/04/2015 23:46:15
dron_1430783175.pcap                          100%  193KB 192.7KB/s   00:00
```

FIGURA 6.9: Ejecución del monitor, envío de archivos.

- maxMacState: cantidad de lecutras macState por archivo.
- interfaz: interfaz donde se capturarán paquetes.

En las figuras 6.8 y 6.9 se pueden ver dos capturas de la ejecución del monitor.

Existieron dificultades considerables para poder compilar el paquete del monitor para que funcione correctamente en el router. Se utilizó el SDK de OpenWRT para realizar la compilación

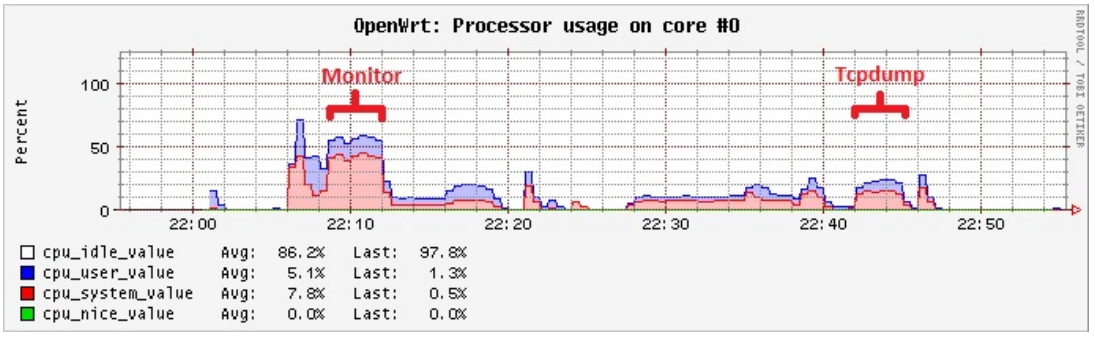

FIGURA 6.10: Gráfica de uso de CPU al ejecutar el monitor (libpcap) y tcpdump.

cruzada en Ubuntu, dado que no se puede compilar directo en el router. Para realizar esta compilación se debe configurar el SDK de forma correcta, ubicar los archivos a compilar en el lugar adecuado y realizar dos makefiles. Uno de ellos es el makefile común que se utiliza en cualquier programa de C/C++, y el otro es específico para OpenWRT en el que se configuran diferentes parámetros del paquete que se está compilando. Además, se deben configurar las diferentes cabeceras que se incluyan y utilicen en el código. Es importante tener en cuenta que las bibliotecas típicas usadas en cualquier programa en C/C++ son sustituidas por otras versiones reducidas de las mismas para que funcionen en OpenWRT. En el apéndice D se puede ver el paso a paso realizado en mayor detalle junto a los correspondientes makefiles.

En el apéndice C se presenta parte del código del monitor.

6.2.6. Comparación libpcap y tcpdump

Dado que en un comienzo se utilizó tcpdump para obtener los datos y luego se pasó a utilizar la biblioteca libpcap, se decidió realizar una comparación de performance entre ambas soluciones. Para esto se utilizó “luci-app-statistics” y “collectd”.

Se realizó el mismo tipo de captura en ambas, midiendo uso de CPU y memoria RAM. Entre las 22:09 y 22:12 se ejecutó el monitor (el cual realiza la captura con libpcap, ejecuta el script de uso del aire y envía las capturas usando scp), y entre las 22:42 y 22:45 se ejecutó un tcpdump junto al script de uso de aire y el envío de datos hacia el servidor. Los resultados obtenidos se presentan en las gráficas 6.10 y 6.11.

El uso de CPU al utilizar tcpdump es inferior a la mitad que al ejecutar el monitor. En cuanto al uso de memoria RAM, la misma es similar en ambos casos. Hay que resaltar que el monitor obtiene más datos que los que se obtienen con tcpdump, lo que ya de por sí justifica su preferencia, pero si además consideramos que el nodo monitor se utiliza de forma dedicada para monitorizar (valga la redundancia), resulta innecesario el ahorro de uso de CPU que aporta utilizar tcpdump.

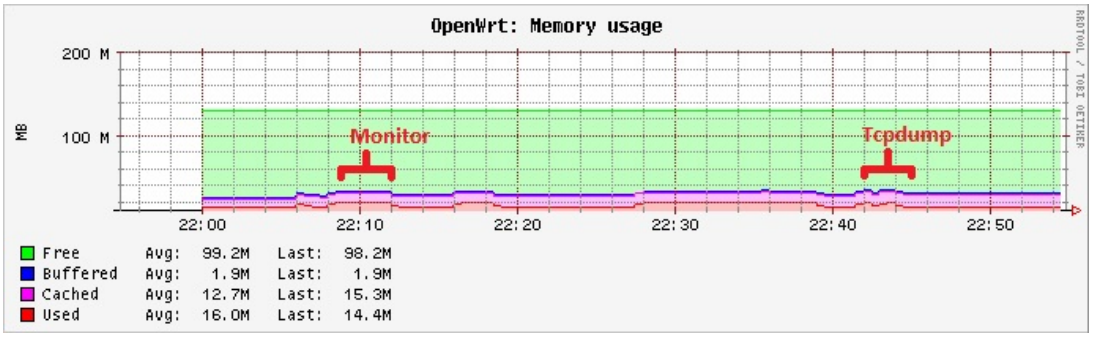

FIGURA 6.11: Gráfica de uso de memoria al ejecutar el monitor (libpcap) y tcpdump.

Capítulo 7

Evaluación

7.1. Caso de estudio

Se han realizado estudios de diferentes estados del arte y se ha desarrollado una herramienta capaz de monitorizar exitosamente las redes inalámbricas densas, de forma de brindar datos para comprender su funcionamiento y poder tomar decisiones en su planificación. A continuación se detallan las particularidades del caso de estudio.

7.1.1. Escenario de pruebas

En el Instituto de Computación de la Facultad de Ingeniería, se realizó el despliegue de los monitores. Se ubicaron 4 routers distribuidos en dos pisos. En la figura 7.1 se pueden ver los 4 routers. Los mismos monitorizaron todas las redes inalámbricas existentes en el aire durante una semana.

El primero que se aprecia (de color rojo), en la sala 103, corresponde al monitor "sala de carrera". A su derecha (de color amarillo), pegado a la sala 140, se encuentra el monitor "impresora planta baja". En el siguiente piso, comenzando por la izquierda (de color azul), en el espacio libre que se encuentra al centro se ubica el monitor "impresora subsuelo", y finalmente, a su derecha (de color verde), en la sala 036, se encuentra el monitor "sala grados 1". De esta forma nos referiremos a los diferentes monitores de aquí en adelante.

La disposición de los mismos estuvo condicionada al acceso que se pudo tener a las diferentes oficinas, así como el acceso a la red eléctrica y de datos.

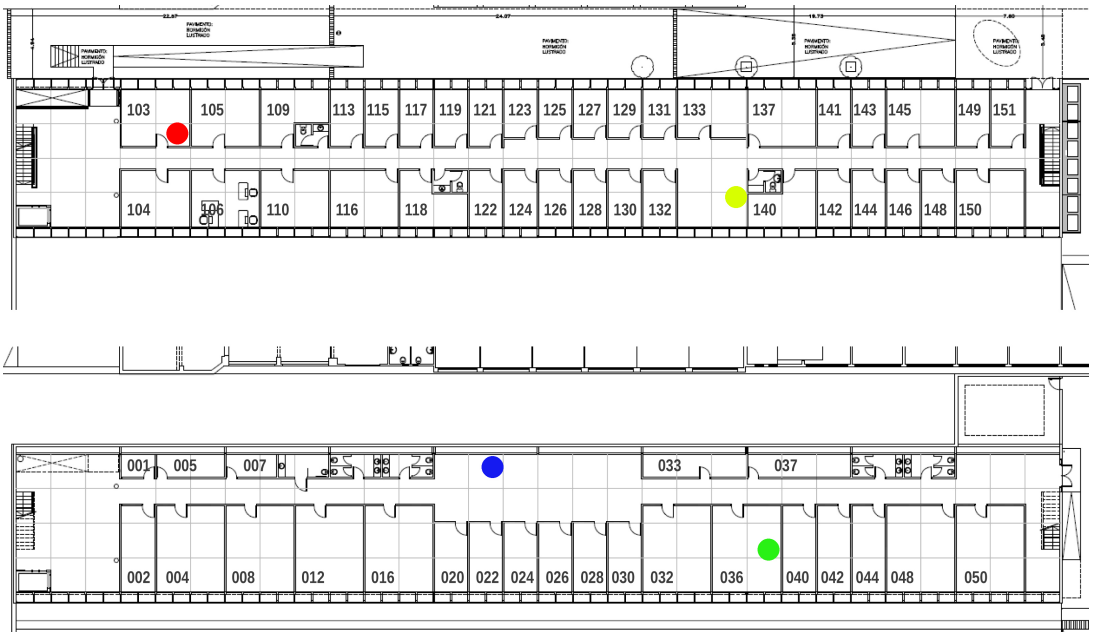

FIGURA 7.1: Plano del InCo con los monitores.

7.1.2. Métricas de las capturas de datos

Se presentan diferentes métricas sobre la captura de datos.

- Cada monitor envió al servidor, en promedio, 4.700 capturas por día.
- El tamaño promedio de cada captura fue de 285KB.
- La cantidad promedio de paquetes por captura fue de 1.000.
- La base de datos creció, en promedio, 4.14 GB por día.
- La base de datos aumentó, en promedio, 19 millones de tuplas por día.

7.2. Procesamiento de los datos capturados

Tomando como conjunto de datos la información capturada por los 4 monitores en el plazo de una semana, se realizaron los siguientes análisis.

7.2.1. Análisis medio ocupado

Se presenta el análisis realizado sobre el uso del medio. Se tomaron mediciones de “Busy Time” (tiempo en el cual la interfaz está ocupada recibiendo cualquier tipo de datos) y “Recieve Time”

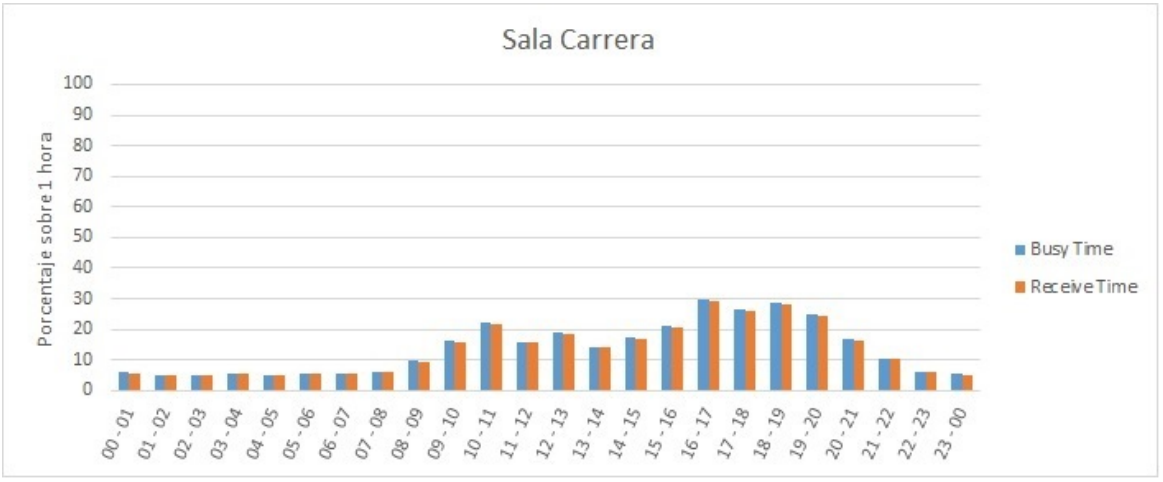

FIGURA 7.2: Porcentaje de medio ocupado de sala de carrera - Lunes 15.

(tiempo en el cual la interfaz está recibiendo datos que logra interpretar). Por lo comentado anteriormente, es claro que el "Busy Time" incluye al "Recieve Time".

Al encontrarse los monitores en modo promiscuo, los valores aquí presentados corresponden a la utilización del medio (aire), por lo cual estos datos proporcionan información sobre la utilización del medio en sí. Se tomó un día con nivel de uso representativo para mostrar la evolución del uso del aire en el correr del día.

Esta métrica resulta de especial interés, dado que refleja la oportunidad de transmitir que poseen los dispositivos. Si el medio se encuentra ocupado la mayor parte del tiempo los dispositivos tendrán menos oportunidad de transmitir.

En las figuras 7.2, 7.3, 7.4 y 7.5 se pueden ver los porcentajes de "Busy Time" y "Recieve Time" durante todo un día (hábil) agrupado cada una hora, de los 4 monitores. En estas gráficas se puede apreciar la diferencia de utilización del medio en los diferentes monitores. Además es posible observar como varía el uso de las redes a lo largo del día. Otra información relevante que brinda este análisis es la diferencia entre el "Busy Time" y el "Recieve Time" que se aprecia en las gráficas, donde la misma corresponde a las interferencias y/o colisiones ocurridas en ese lapso de tiempo.

En la figura 7.6 se presenta, en una misma gráfica, los diferentes porcentajes de utilización del medio, capturado por cada uno de los monitores.

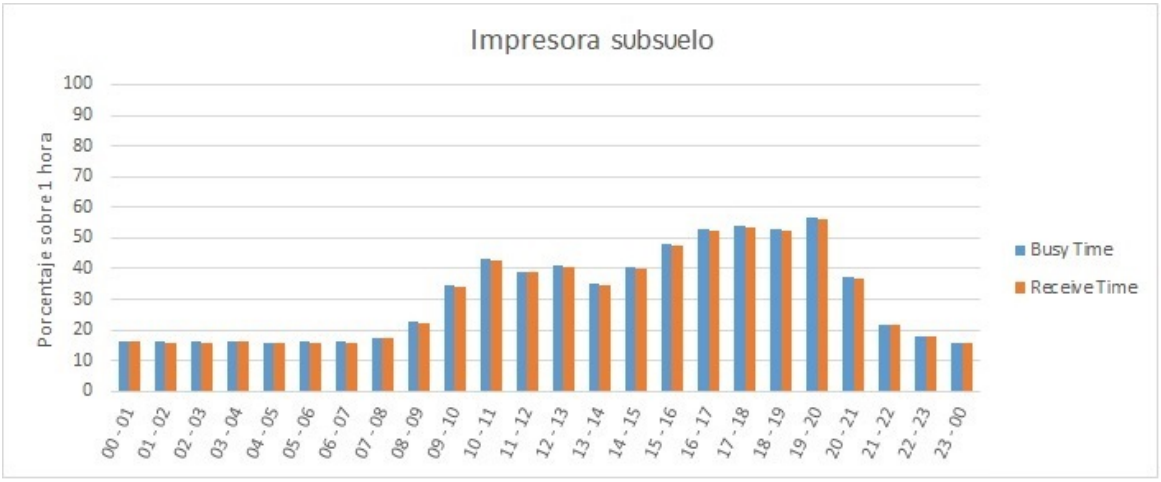

FIGURA 7.3: Porcentaje de medio ocupado de impresora subsuelo - Lunes 15.

FIGURA 7.4: Porcentaje de medio ocupado de sala grados 1 - Lunes 15.

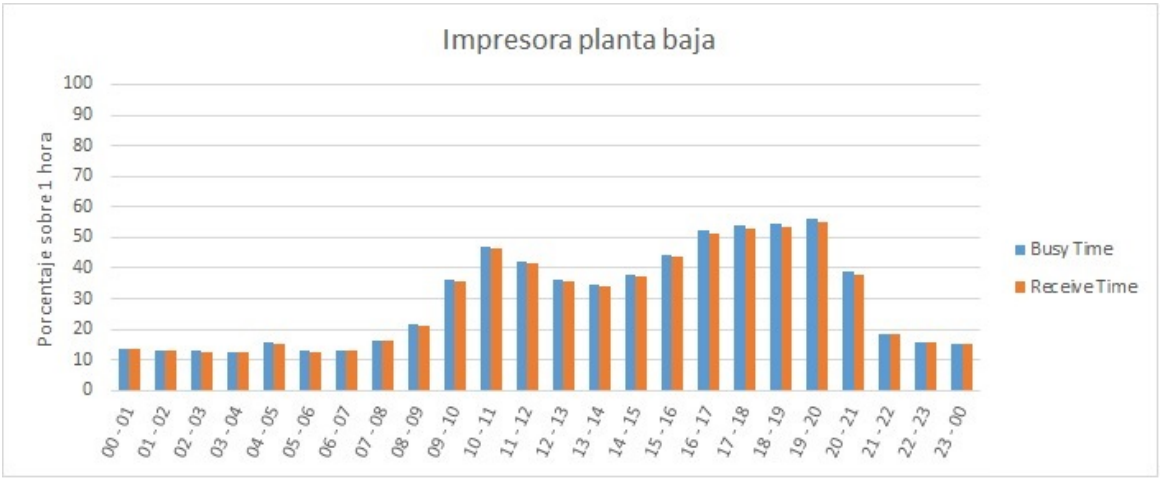

FIGURA 7.5: Porcentaje de medio ocupado de impresora planta baja - Lunes 15.

FIGURA 7.6: Porcentaje de utilización del medio para los 4 monitores - Lunes 15.

7.2.2. Tipos de paquetes

En esta sección se presentan diferentes tablas donde se muestra la cantidad de paquetes de cada tipo y subtipo observados por cada uno de los monitores durante un día tipo agrupados por AP (es decir, por cada punto de acceso que está prestando servicio). Con esto es posible apreciar los diferentes tipos de paquetes que son intercambiados en cada conjunto de servicio básico.

Estos datos se pueden ver en las tablas 7.2, 7.3, 7.4 y 7.5. La tabla 7.1 muestra la relación entre la MAC del AP y el identificador utilizado en el resto de las tablas.

A continuación se presenta una breve descripción de cada sub-tipo de paquete que se encontró durante la monitorización:

- Association Request: enviado hacia un AP para asociarse al mismo (en caso de existir seguridad en la red, el cliente debe estar previamente autenticado).
- Association Response: enviado por un AP en respuesta a un "Association Request".
- Reassociation Request: es similar a un "Association Request", con la diferencia de que incluye la información sobre la actual asociación que posea el cliente.
- Reassociation Response: enviado por un AP en respuesta a un "Reassociation Request".
- Probe Request: se utiliza para buscar APs disponibles.
- Probe Response: enviado por un AP en respuesta a un "Probe Request".
- Beacon: enviado periódicamente por un AP para anunciarse.
- Disassociation: indica la finalización de una asociación.
- Authentication: se utiliza para la autenticación de un cliente frente a un AP.

MAC AP	ID	MAC AP	ID
00:0f:02:12:58:50	AP 1	c0:4a:00:a4:6b:8d	AP 9
90:f6:52:f5:d2:d0	AP 2	c0:4a:00:a4:6b:db	AP 10
90:f6:52:f5:d7:1c	AP 3	c2:4a:00:a4:57:e1	AP 11
92:f6:52:f5:d2:d0	AP 4	c2:4a:00:a4:6a:b5	AP 12
c0:4a:00:a4:57:e1	AP 5	c2:4a:00:a4:6b:5f	AP 13
c0:4a:00:a4:6a:b5	AP 6	c2:4a:00:a4:6b:81	AP 14
c0:4a:00:a4:6b:5f	AP 7	c2:4a:00:a4:6b:8d	AP 15
c0:4a:00:a4:6b:81	AP 8	c2:4a:00:a4:6b:db	AP 16

Tabla 7.1: Relación MAC - Identificador de AP.

- Deauthentication: indica que quien la recibe no está mas autenticado.
- PS-Poll: se utiliza en el modo de ahorro de energía.
- RTS: utilizado para la coordinación del acceso al medio.
- CTS: se utiliza en respuesta a un "RTS".
- ACK: utilizado para confirmar una recepción.
- CF End: indica la finalización de un período "CFP". Se utiliza cuando el AP se encuentra en modo "PCF".
- Data: contiene datos.

De estos datos llama la atención, dentro de los paquetes de tipo gestión la gran cantidad de "Beacons" que se envían. En cuanto a los paquetes de tipo control, se destacan tanto los "RTS" como los "ACK". En ambos tipos (gestión y control) se detectaron paquetes que se reconocen dentro de estas categorías, pero no se logra detectar su subtipo. Específicamente, el valor del campo del paquete que representa el subtipo no es ninguno de los esperados en el estándar 802.11.

La comparación entre los diferentes tipos de paquetes (managment, control y data) se realiza más adelante.

Tipo de paquete	AP 3	AP 5	AP 7	AP 10	AP 11	AP 13	AP 16
Association Request	6	28	4	16	138	2	50
Association Response	0	168	0	57	221	0	21
Reassociation Request	14	0	0	1	53	7	12
Reassociation Response	0	0	0	2	77	0	4
Probe Request	27	0	1	3197	440	69	47
Probe Response	2	18623	0	0	25378	0	2159
Beacon	29	763488	0	88244	751055	5	84534
Disassociation	0	22	0	5	337	8	11
Authentication	64	503	25	119	744	16	94
Deauthentication	11	178	1	82	941	5	57
Managment Desconocido	30	180	9	74	7218	14	181
Desconocido Control	209	150	7	77	55278	20	423
PS-Poll	133	0	1	0	1462	57	656
RTS	146	4136	0	244	63537	120	3149
CTS	737	119	26	38	5996	443	669
ACK	367	3538	217	791	39770	547	1083
CF End	0	0	0	0	6229	0	0
Data	2036	1035	26	299	2208140	849	183398

Tabla 7.2: Cantidad de tipos de paquete de sala carrera.

Tipo de paquete	AP 1	AP 7	AP 8	AP 9	AP 10	AP 13	AP 14	AP 15	AP 16
Association Request	0	109	103	88	98	1	6	18	7
Association Response	0	73	151	159	133	14	38	50	99
Reassociation Request	0	0	0	7	2	0	4	11	6
Reassociation Response	0	0	0	7	3	7	10	16	12
Probe Request	4761	0	0	0	0	0	3	38	11
Probe Response	527437	4240	8857	16841	6493	6708	12536	27142	6401
Beacon	0	324445	516248	544314	467671	334583	497816	538824	460166
Disassociation	0	1	21	2	8	19	27	43	19
Authentication	0	322	536	334	310	35	73	255	130
Deauthentication	0	102	136	194	149	55	222	257	120
Managment Desconocido	19	156	341	284	253	3844	5105	3965	982
Desconocido Control	12	152	223	244	212	60	13621	2565	255
PS-Poll	0	0	0	0	0	0	58	1	3802
RTS	0	354	1173	3176	423	4555	19488	4735	19206
CTS	0	233	296	141	186	23	772	250	4076
ACK	1790	1347	3096	5248	3088	1895	12516	31643	1630
CF End	6	0	0	0	0	0	0	30	11
Data	2121	2375	2510	4054	1990	423052	803120	1236543	521654

Tabla 7.3: Cantidad de tipos de paquete de impresora subsuelo.

Tipo de paquete	AP 6	AP 7	AP 8	AP 9	AP 12	AP 13	AP 14	AP 15
Association Request	10	54	19	22	3	0	8	4
Association Response	0	0	250	209	0	0	75	61
Reassociation Request	0	0	0	0	0	0	5	3
Reassociation Response	0	0	1	7	0	0	18	28
Probe Request	0	0	0	0	19	2	6	10
Probe Response	11	5	16538	18469	10	3	23771	30727
Beacon	2312	724	663900	584351	2155	649	654269	575506
Disassociation	0	0	17	0	3	1	56	50
Authentication	6	73	582	375	7	0	135	334
Deauthentication	0	0	281	279	0	0	313	336
Managment Desconocido	4	63	376	266	338	3	8287	3088
Desconocido Control	4	54	196	193	7161	2	0	173
PS-Poll	0	0	0	0	0	0	667	0
RTS	0	0	1948	689	2788	9	27674	4564
CTS	49	114	15	14	5137	48	577	128
ACK	598	766	1698	1373	10624	172	18385	916
CF End	0	0	0	0	0	0	0	0
Data	340	498	1201	2692	6278	80	1483315	1485105

Tabla 7.4: Cantidad de tipos de paquete de sala grados 1.

Tipo de paquete	AP 2	AP 4	AP 5	AP 6	AP 7	AP 8	AP 11	AP 12	AP 13	AP 14
Association Request	0	0	0	0	4	2	4	7	11	2
Association Response	2	12	0	10	129	198	93	62	22	39
Reassociation Request	0	0	84	0	0	0	7	11	9	3
Reassociation Response	0	9	0	1	0	1	17	33	12	13
Probe Request	0	1	0	0	1	0	10	23	21	1
Probe Response	2061	5548	3892	4837	10610	8065	4912	18795	13189	12515
Beacon	92484	87408	203876	545526	548186	522082	194154	536074	550038	495735
Disassociation		5	19		1	17	71	141	19	33
Authentication	14	19	180	15	390	446	126	248	50	71
Deauthentication	4	42	93	23	191	194	227	431	117	168
Managment Desconocido	4	1023	59	11	106	238	1197	3349	4577	5067
Desconocido Control	31	62	40	10	77	115	282	50513	10544	12550
PS-Poll	0	0	0	0	1	0	86	3283	339	198
RTS	127	12594	1715	400	634	1403	8717	25285	7932	21812
CTS	4	45	1	3	8	10	472	4318	1078	585
ACK	12	82	46	1049	842	533	379	35402	4013	6392
CF End	0	0	0	0	0	0	0	0	0	0
Data	229	143239	110	117	743	172	376271	901458	784917	783961

Tabla 7.5: Cantidad de tipos de paquete de impresora planta baja.

7.2.3. Clientes que más paquetes y volumen de datos enviaron

En la tabla 7.6 se pueden ver las MACs de los 5 clientes que enviaron mayor cantidad de paquetes durante un día, separado por monitor. Luego se presenta, en la tabla 7.7, información similar, pero esta vez sobre los 5 clientes que enviaron mayor cantidad de tráfico ese mismo día.

Sala carrera		
Cliente	Cantidad de paquetes	Bytes
Desconocido	91004	3640160
58:94:6b:9c:8c:48	35783	9318451
70:1a:04:a1:82:25	26132	3144990
08:ed:b9:9a:1d:9b	25064	4186225
d3:54:95:07:4a:00	24290	971600
Impresora subsuelo		
Cliente	Cantidad de paquetes	Bytes
5f:00:62:30:6b:8d	29912	1196480
Desconocido	23766	950640
18:1e:b0:ab:db:a0	17805	1489025
00:18:e7:02:cb:11	14484	1448801
2b:03:97:3c:cb:11	10365	414600
Sala grados 1		
Cliente	Cantidad de paquetes	Bytes
Desconocido	23126	925142
74:4c:d4:39:4a:00	14455	578200
b8:ee:65:43:7f:db	12763	707870
19:da:cc:0e:6b:81	11565	462600
19:da:cc:0e:6b:8d	11278	451120
Impresora planta baja		
Cliente	Cantidad de paquetes	Bytes
Desconocido	43907	1756382
3c:cb:7c:32:67:5b	34633	4423040
80:89:7b:01:6a:b5	17608	704320
b8:ee:65:43:7f:db	10019	841568
19:da:cc:0e:6b:81	7372	294880

Tabla 7.6: Top 5 de clientes que más paquetes enviaron en un día.

Sala carrera		
Cliente	Cantidad de paquetes	Bytes
58:94:6b:9c:8c:48	35783	9318451
08:ed:b9:9a:1d:9b	25064	4186225
Desconocido	91004	3640160
70:1a:04:a1:82:25	26132	3144990
a4:9a:58:12:25:34	22736	1772603
Impresora subsuelo		
Cliente	Cantidad de paquetes	Bytes
58:94:6b:9c:8c:48	35783	9318451
08:ed:b9:9a:1d:9b	25064	4186225
Desconocido	91004	3640160
70:1a:04:a1:82:25	26132	3144990
a4:9a:58:12:25:34	22736	1772603
Sala grados 1		
Cliente	Cantidad de paquetes	Bytes
58:94:6b:9c:8c:48	9568	1393258
c0:18:85:7e:94:ad	6849	1128613
Desconocido	23126	925142
24:fd:52:f5:e0:04	5574	894128
b8:ee:65:43:7f:db	12763	707870
Impresora planta baja		
Cliente	Cantidad de paquetes	Bytes
3c:cb:7c:32:67:5b	34633	4423040
Desconocido	43907	1756382
b8:ee:65:43:7f:db	10019	841568
80:89:7b:01:6a:b5	17608	704320
6c:b7:f4:9e:c2:ca	4716	351137

Tabla 7.7: Top 5 de clientes que más datos transfirieron en un día.

Como se puede apreciar, en todas las tablas existe una fila para la cual no se conoce la MAC de origen del paquete, ya que la misma estaba vacía. Este resultado generó interés por saber de que tipo de tráfico se trataba, por lo cual se analizó el mismo en mayor profundidad. De la especificación, se sabe que los paquetes "Clear To Send" son los únicos que no poseen MAC origen. Analizando estos datos en profundidad, efectivamente se comprobó que los mismos corresponden a paquetes CTS, de los cuales alrededor de 23.000 tienen como destino a algún AP, y aproximadamente 154.000 poseen como destino a algún cliente (se detectaron en total 75 destinos diferentes). Esto indica claramente el overhead que este protocolo está generando sobre la red.

7.2.4. Cantidad de paquetes y tráfico durante una semana

En esta sección se presentan gráficas comparativas de los 3 diferentes tipos de paquetes existentes durante toda una semana (desde el Martes 09 hasta el Martes 16), medidos por cada monitor. Se tienen dos gráficas por cada uno, comparando así cantidad de paquetes y tráfico generado.

En las gráficas 7.7 y 7.8 se ve la información relativa a la sala de carrera. Se aprecia claramente la utilización de la red a lo largo del día (con gran uso entre las 08 y las 20) y la baja de utilización el fin de semana. En particular, el día 11, alrededor de las 19 horas, existió un pico de paquetes de control. Se investigaron los subtipos, tamaños, orígenes y destinos de estos paquetes en la ventana de tiempo que va desde las 18 hasta las 21. En la tabla 7.8 se presentan los datos relevantes, donde se puede apreciar el alto número de paquetes RTS y CTS, lo cual destaca frente a la cantidad de estos paquetes durante el resto de la semana. También se presenta una gran cantidad de paquetes de control con subtipo desconocido. En cuanto a los orígenes y destinos de estos paquetes, no se encontraron datos relevantes.

Subtipo	Cantidad de paquetes	Cantidad total en KB
Desconocido	218107	12353
RTS	253909	11406
CTS	236663	9244
ACK	252463	9861

Tabla 7.8: Análisis particular de subtipos de paquete en sala carrera de las 18 a las 21.

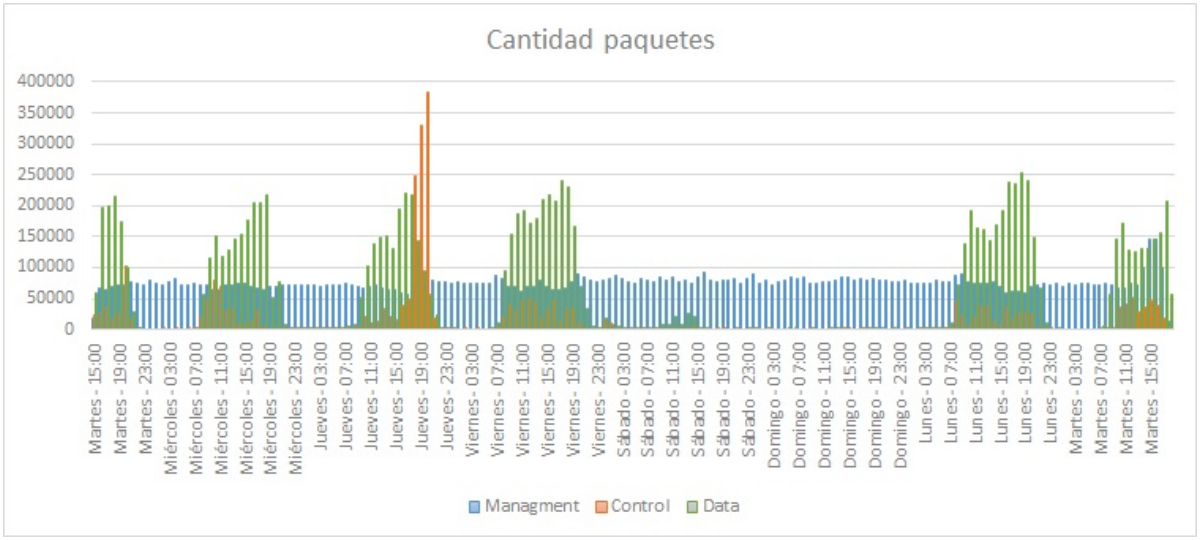

FIGURA 7.7: Comparación de cantidad de paquetes de cada tipo durante una semana en sala de carrera.

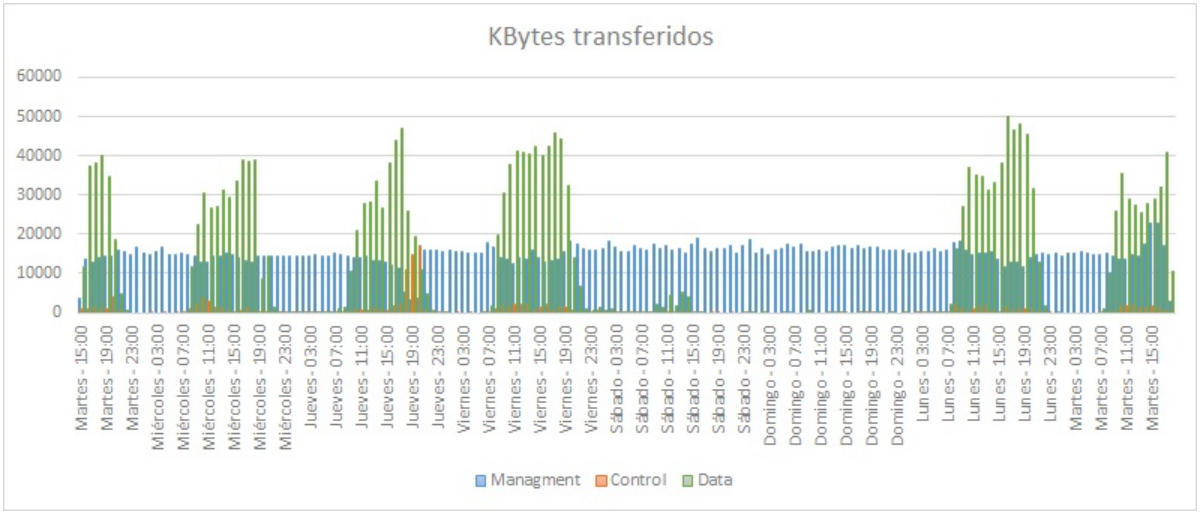

FIGURA 7.8: Comparación de tráfico de cada tipo de paquete durante una semana en sala de carrera.

En las gráficas 7.9 y 7.10 se ve la información relativa a la impresora subsuelo. Se aprecian 3 intervalos donde no se tienen datos, que corresponden a un error ocurrido en este monitor. Por razones que se desconocen, la interfaz de red dejaba de funcionar correctamente y no capturaba paquetes. Se investigó y se buscó información relacionada al problema pero no se logró detectar las causas del mismo, ni se obtuvo información de un problema similar al que se pudiera vincular. Se optó por no repetir la toma de datos, ya que se disponía de la semana completa en otros monitores, y para las comparaciones utilizando los cuatro monitores se utilizaron muestras diarias completas.

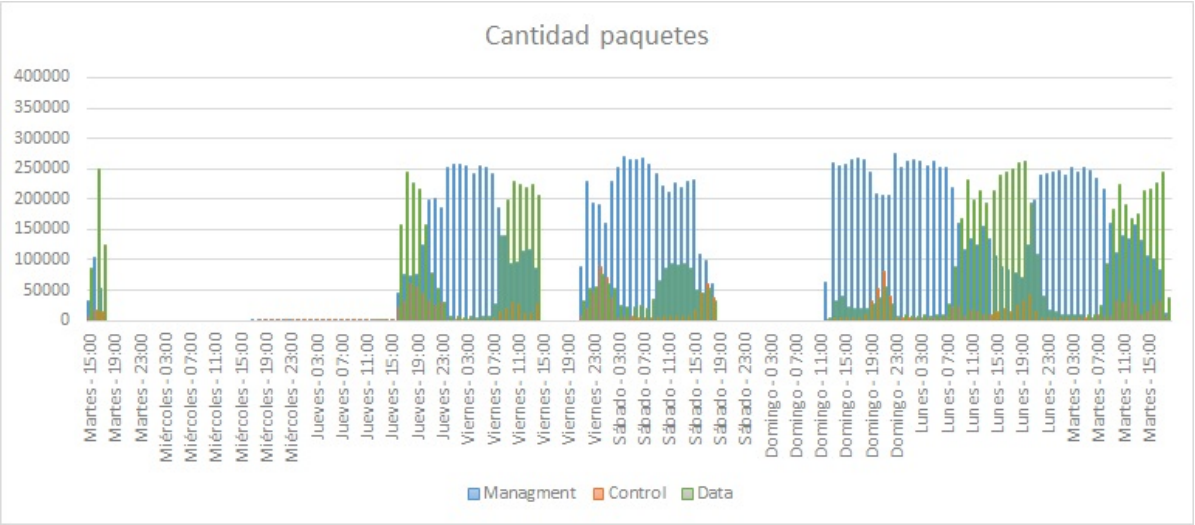

FIGURA 7.9: Comparación de cantidad de paquetes de cada tipo durante una semana en impresora subuselo.

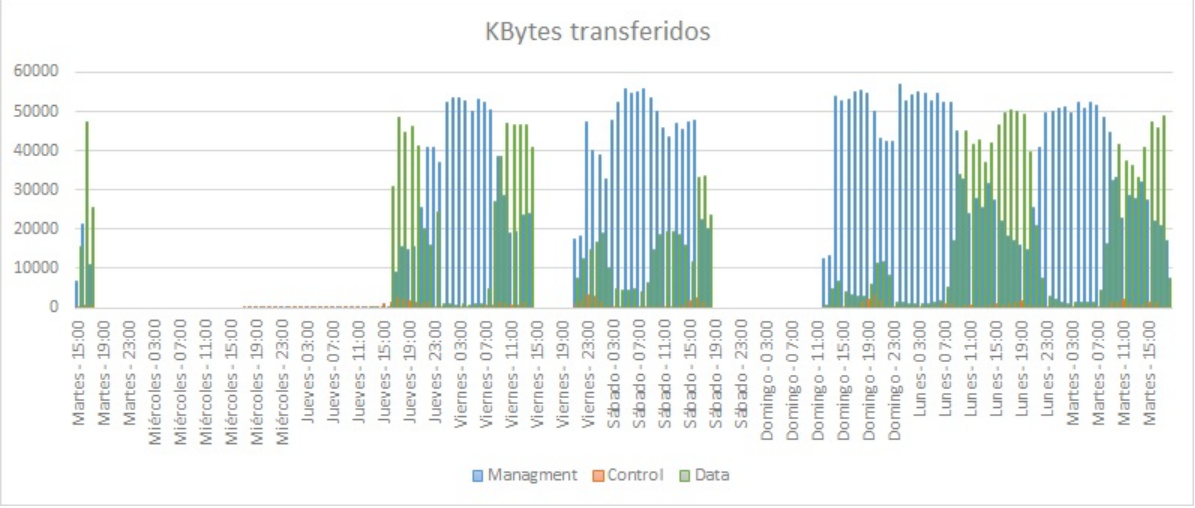

FIGURA 7.10: Comparación de tráfico de cada tipo de paquete durante una semana en impresora subuselo.

En las gráficas 7.11 y 7.12 se vé la información relativa a la sala grados 1. La gráfica es similar a la de sala de carrera, con una mayor cantidad de paquetes y volumen de tráfico.

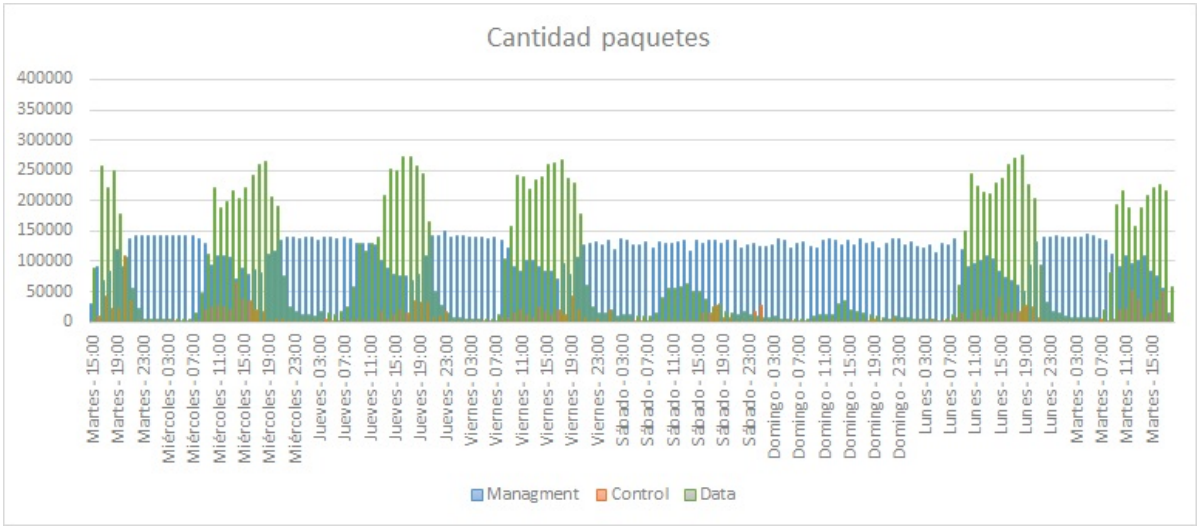

FIGURA 7.11: Comparación de cantidad de paquetes de cada tipo durante una semana en sala grados 1.

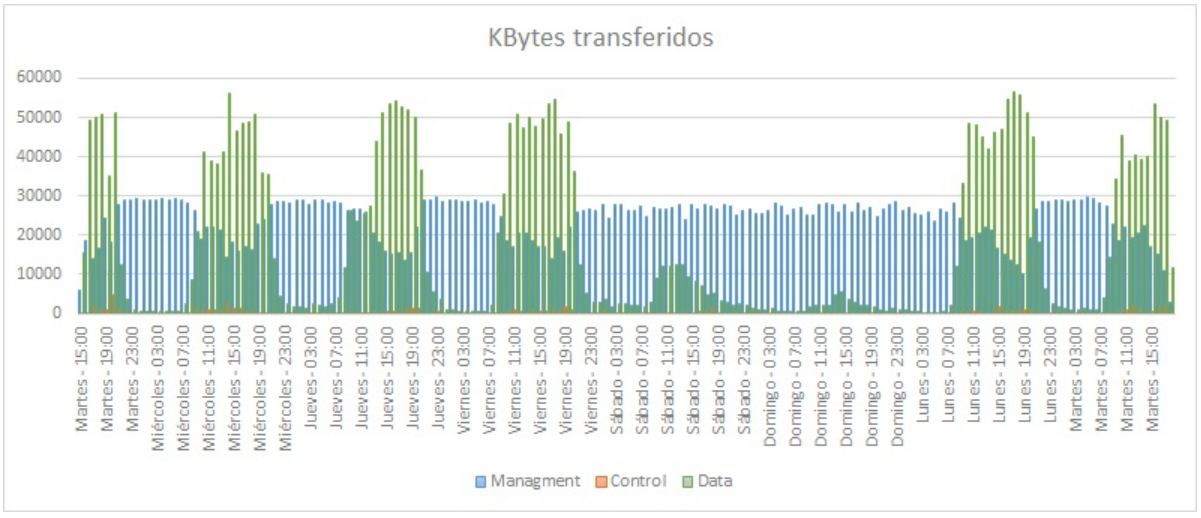

FIGURA 7.12: Comparación de tráfico de cada tipo de paquete durante una semana en sala grados 1.

En las gráficas 7.13 y 7.14 se ve la información relativa a la impresora planta baja. De la misma forma que en la impresora subuselo, en este monitor se tienen dos cortes en la toma de datos por la misma razón. Se destacan en estas gráficas la alta cantidad de paquetes managment en comparación con los paquetes data.

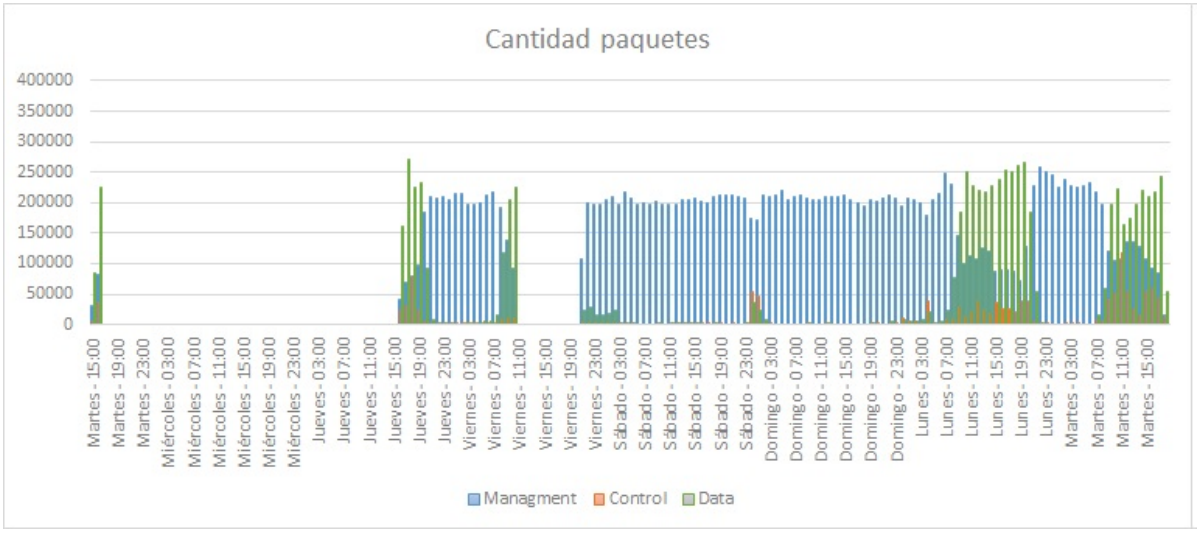

FIGURA 7.13: Comparación de cantidad de paquetes de cada tipo durante una semana en impresora planta baja.

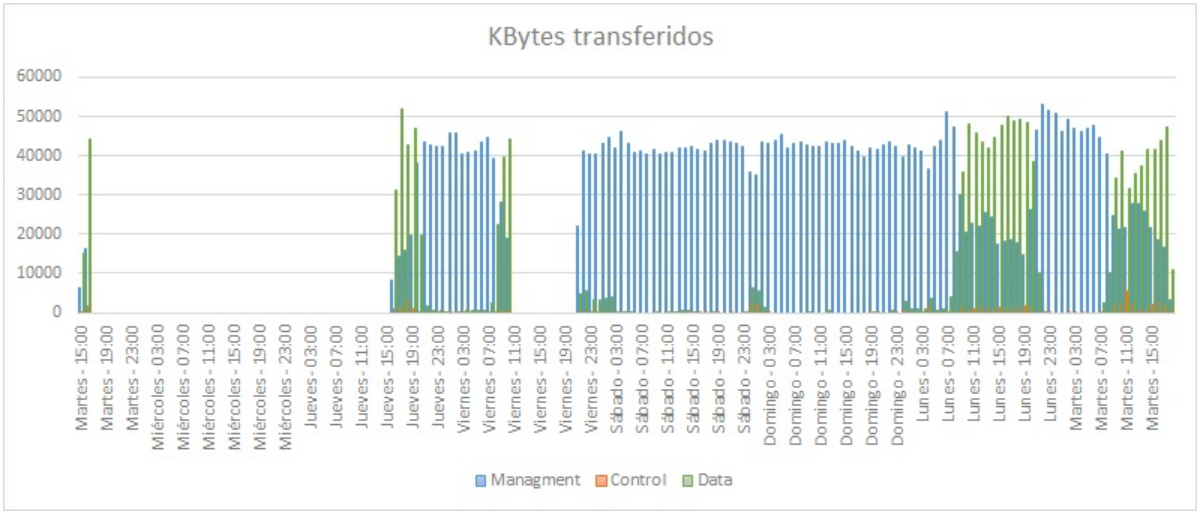

FIGURA 7.14: Comparación de tráfico de cada tipo de paquete durante una semana en impresora planta baja.

7.2.5. Comparación de cantidad de paquetes y transferencia semanal

En la gráfica 7.15 se presenta la comparación semanal de la cantidad de paquetes detectada por cada monitor. En la gráfica 7.16 se realiza esta misma comparación por tráfico.

Se aprecia que en los valles los monitores de las impresoras poseen una actividad mayor que los otros dos, mientras que en las crestas esta diferencia se hace menor, alcanzando la igualdad en algunos momentos.

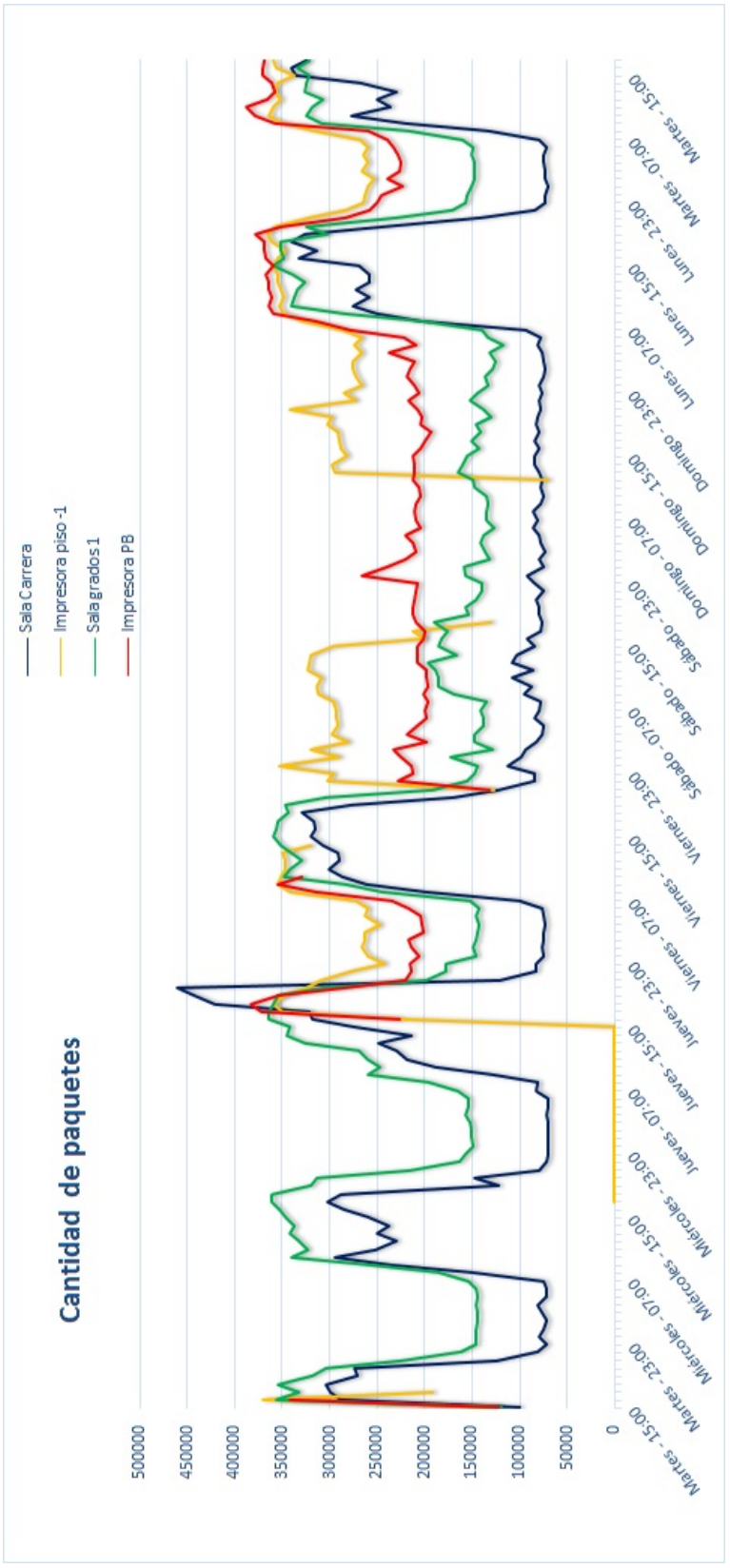

FIGURA 7.15: Comparación de cantidad de paquetes capturada por cada monitor.

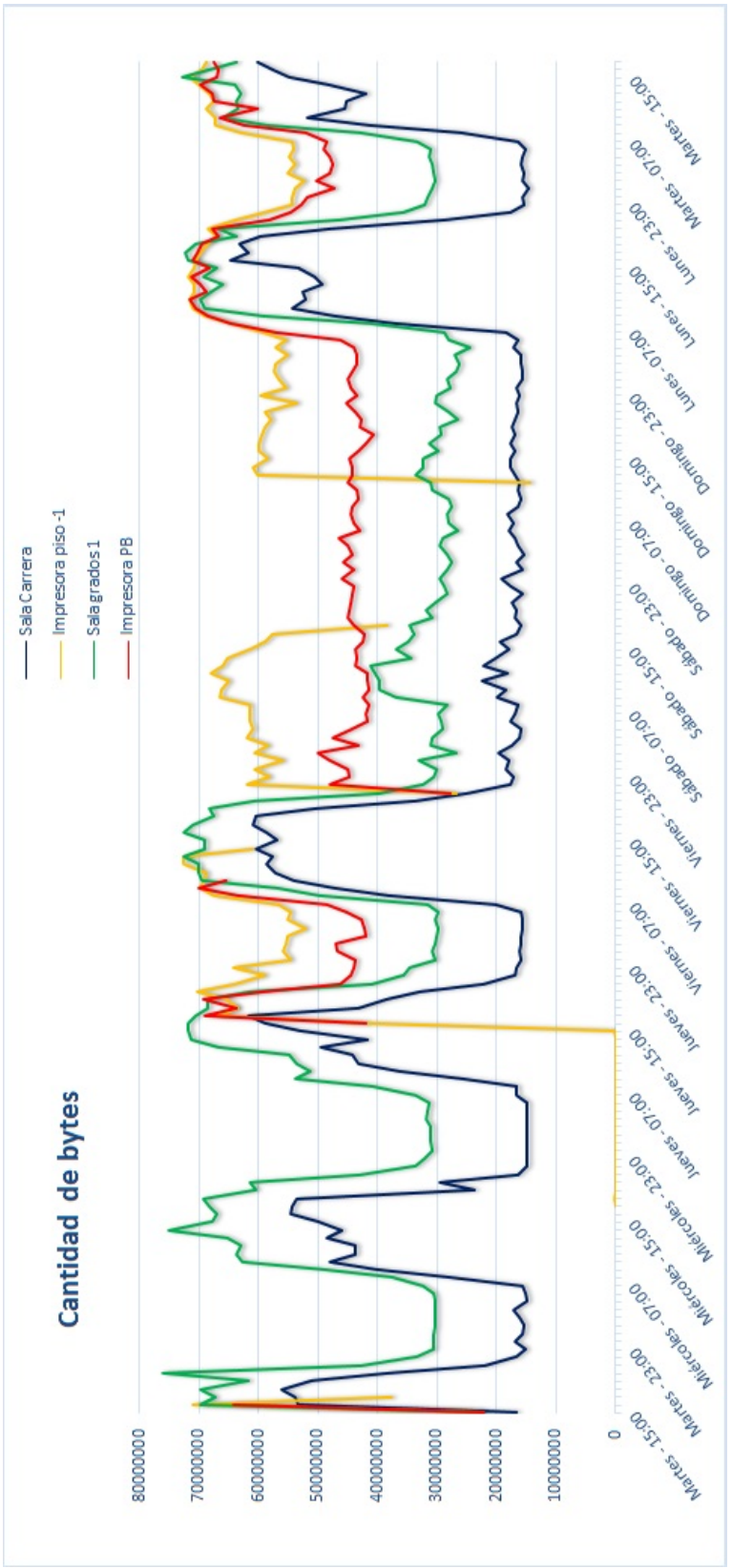

FIGURA 7.16: Comparación de cantidad de tráfico capturado por cada monitor.

7.2.6. Medición de señal

Se analizó el comportamiento de la señal durante un día, tomando las mediciones capturadas por un monitor que se encontraba al lado de un AP, y teniendo en cuenta solo aquellos paquetes dirigidos hacia el AP. En las figuras 7.17 y 7.18 se pueden ver los resultados.

El pico que se aprecia en ambas gráficas puede deberse a que al menos un cliente se aproximó al AP monitorizado, aumentando la intensidad de la señal registrada por el monitor. Se entiende que esto es así ya que es común utilizar WiFi en los celulares, la hora del pico coincide con la hora que arranca la jornada y el AP se encuentra al lado de unas sillas de espera.

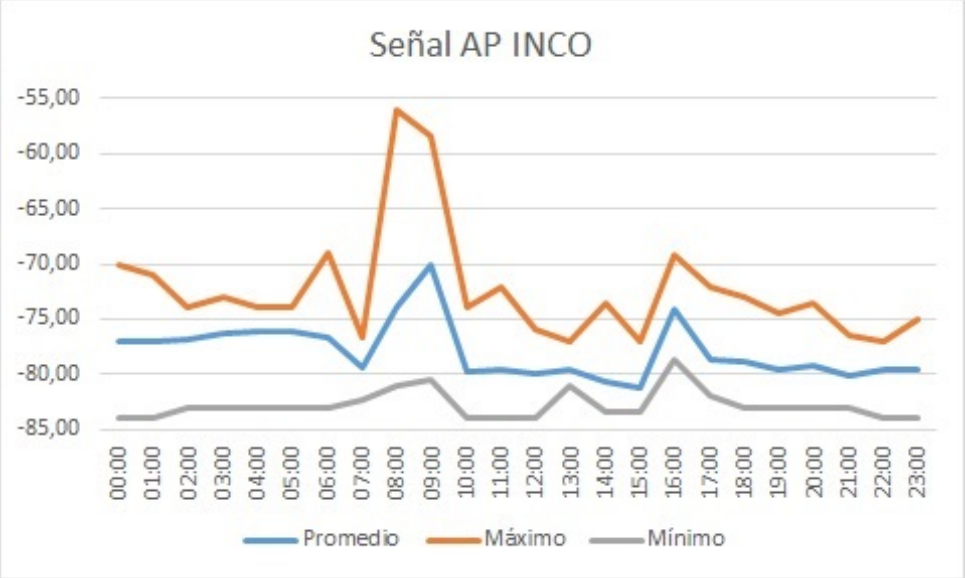

FIGURA 7.17: Comportamiento de la señal a lo largo del día en un AP de la red INCO.

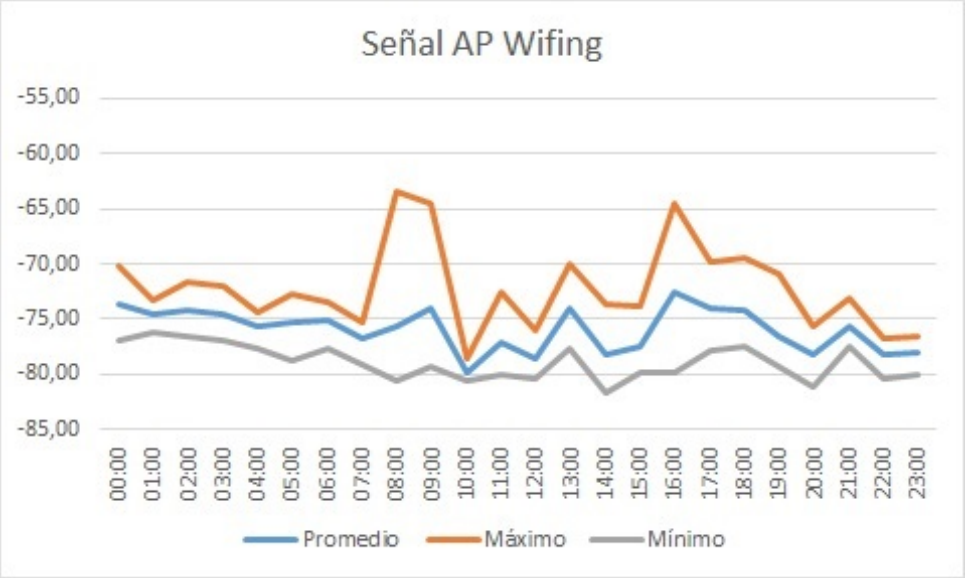

FIGURA 7.18: Comportamiento de la señal a lo largo del día en un AP de la red Wifing.

7.2.7. Comparación de tipos en cada monitor

En la figura 7.19 se presentan 4 gráficas de tipo torta donde se muestra la relación entre la cantidad de paquetes de los 3 tipos existentes para cada monitor durante la semana. Resalta el bajo porcentaje de paquetes de datos en comparación con los otros dos. Más aún, el alto porcentaje de paquetes managament muestra la burocracia a la que se enfrentan las redes inalámbricas 802.11 para poder funcionar. Sin embargo, este resultado no es del todo representativo, ya que la muestra tomada incluye las noches y el fin de semana, cuando la cantidad de datos desciende considerablemente. Por lo tanto, se realizaron gráficas similares tomando como muestra una hora (correspondiente al lunes 15 entre las 15:00 y las 16:00). Estos resultados se presentan en la figura 7.20, donde se aprecia el predominio de los paquetes de datos frente al resto.

En la figura 7.21 se presentan 4 gráficas similares a las anteriores, comparando por volumen de tráfico en lugar de cantidad de paquetes. Destaca el alto tráfico generado por los paquetes de

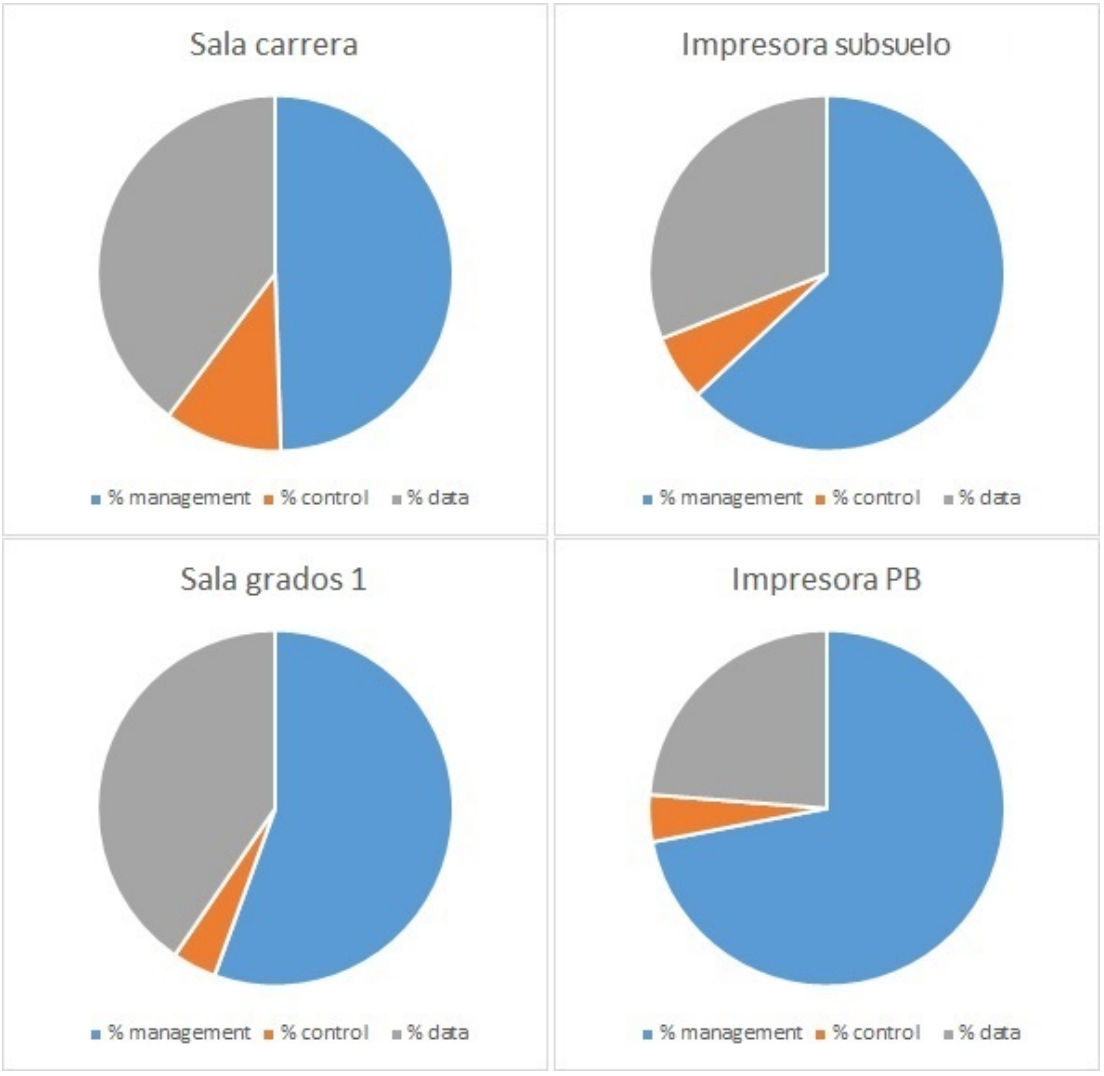

FIGURA 7.19: Relación semanal entre cantidad de paquetes de cada tipo por monitor.

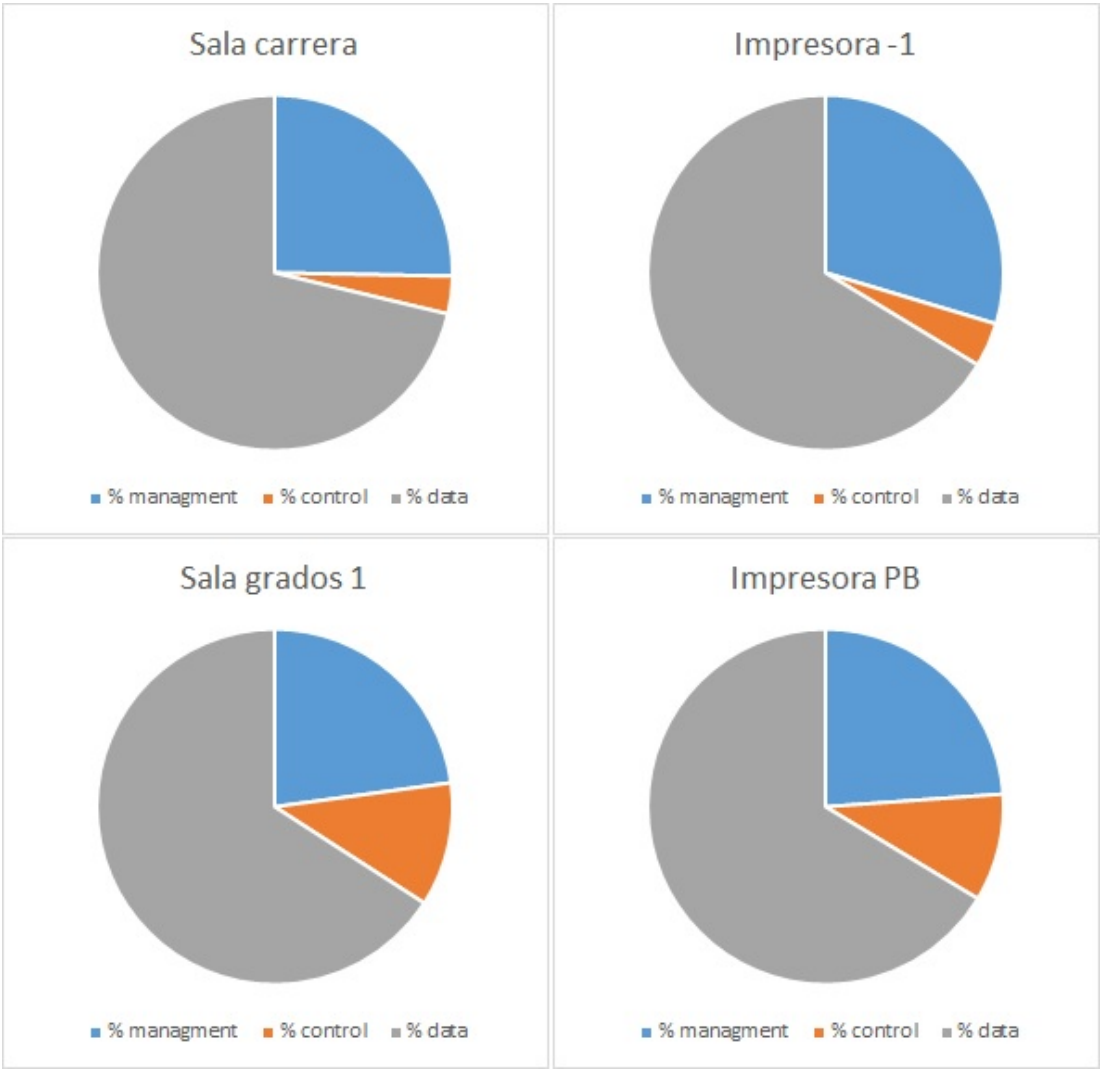

FIGURA 7.20: Relación en una hora entre cantidad de paquetes de cada tipo por monitor.

tipo managment frente a los de tipo data, donde se esperaba que se "emparejaran" un poco las proporciones con respecto a la cantidad de paquetes, dado que los paquetes de tipo management tienen tamaño fijo y los de tipo data pueden ser más grandes. De forma similar a las gráficas del punto anterior, se analizó el volumen de tráfico en el mismo rango horario. Los resultados se presentan en la figura 7.22.

Este resultado

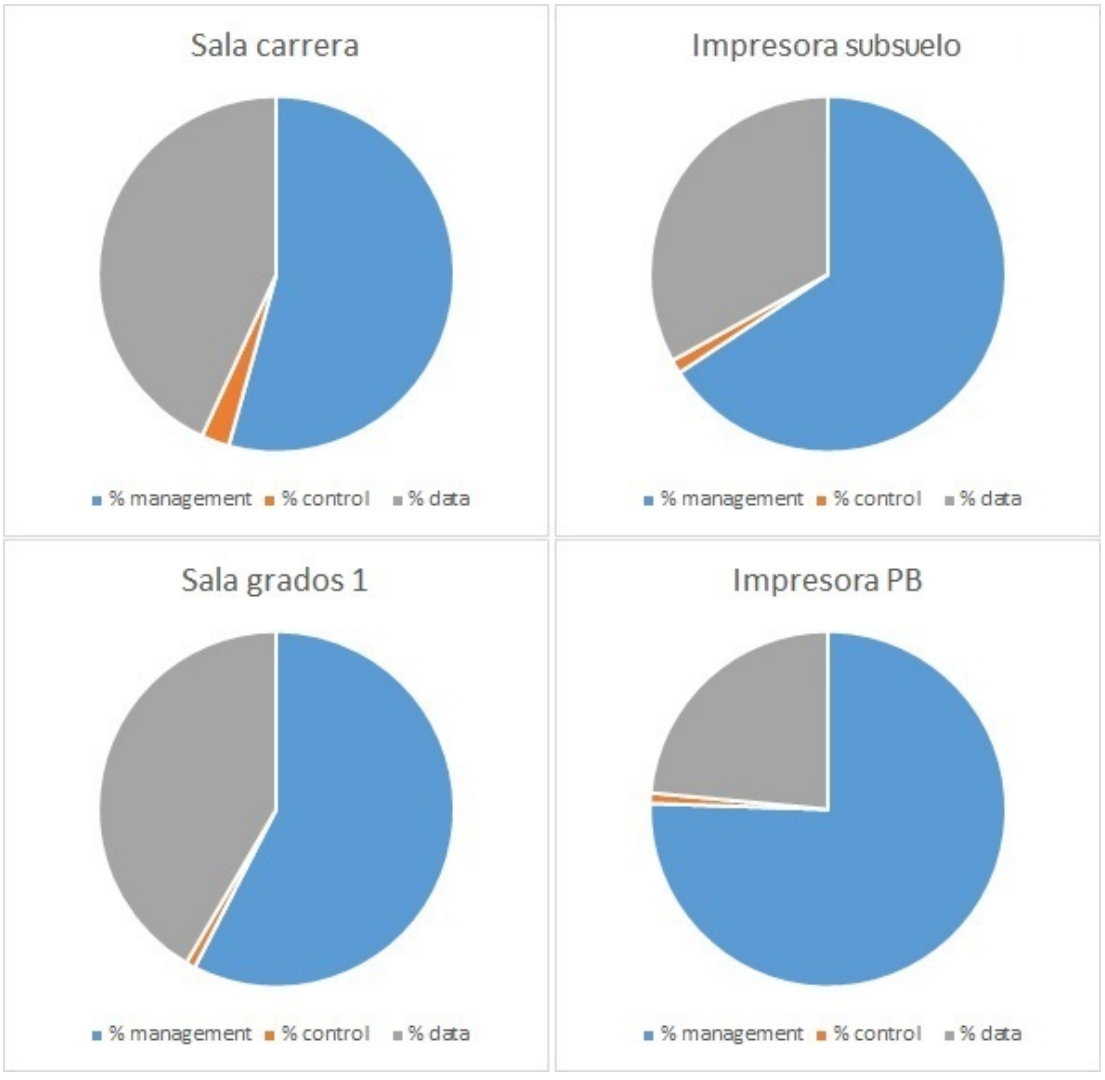

FIGURA 7.21: Relación semanal entre tráfico de cada tipo de paquetes por monitor.

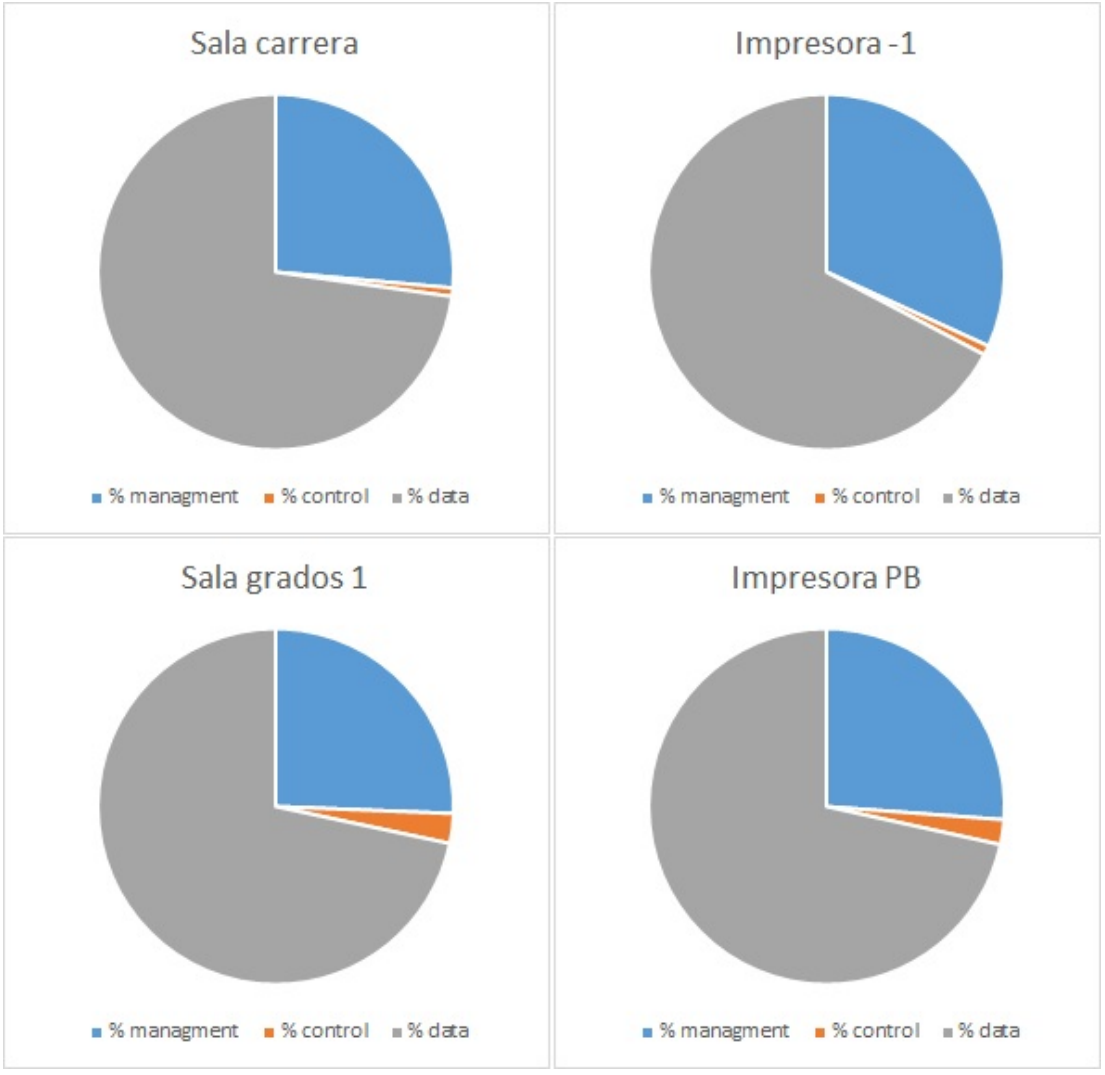

FIGURA 7.22: Relación en una hora entre cantidad de paquetes de cada tipo por monitor.

7.2.8. Cantidad de dispositivos registrados

Se presenta en esta sección la cantidad de dispositivos registrados por los monitores durante una semana y durante un día. Se registra cualquier dispositivo que envíe y/o reciba paquetes hacia o desde un AP. Por ejemplo, si un dispositivo envía un "Probe Request" hacia un AP, entonces es registrado.

En la figura 7.23 se presenta la cantidad semanal, donde se tomaron intervalos de 6 horas para cada valor. Estos datos corresponden a lo registrado por los monitores en todo el InCo y sus alrededores.

En la figura 7.24 se presenta la cantidad diaria, donde se tomaron intervalos de 1 hora para cada valor.

FIGURA 7.23: Cantidad de dispositivos totales en intervalos de 6 horas durante una semana.

FIGURA 7.24: Cantidad de dispositivos totales en intervalos de 1 hora durante un día.

Como era de esperar, en las noches y durante todo el fin de semana, la cantidad de dispositivos se reduce considerablemente.

A raíz de los altos valores observados se realizó un análisis en profundidad para el mayor pico registrado (jueves 11 de 15:00 a 21:00). Se obtuvieron por separado las MACs de los dispositivos que enviaron paquetes al AP de aquellas a las que el AP les envió paquetes, resultando en un total de 999 dispositivos que enviaron y 2060 que recibieron. Cruzando estos resultados se desprende que los que mantuvieron una interacción bilateral son 85. Por otra parte, del total de dispositivos que enviaron paquetes, 843 transmitieron más de 50, mientras que para el caso de los dispositivos que recibieron paquetes este valor fue de 1357. Esto indica que una tercera parte de los dispositivos detectados en el pico no alcanzaron a enviar ni recibir 50 paquetes en un lapso de 6 horas.

7.2.9. Tráfico sin AP origen ni destino

En esta sección se analiza el tráfico que no posee como origen ni destino a un AP. Para identificar los APs existentes se buscaron los orígenes de los paquetes de tipo beacons. Por lo tanto, si un AP no envía beacons, no será considerado como AP. Sería posible realizar un análisis en mayor profundidad, buscando el origen de otros paquetes que sean enviados específicamente por los APs para refinar esta lista, pero no se hizo de esa manera porque en la infraestructura de la red del InCo todos los APs envían beacons.

En un lapso de una semana, se encontraron 3.716.462 paquetes que entran en esta categoría. Dentro de estos, se distinguen 12.813 parejas distintas de origen y destino. En la tabla 7.9 se puede ver cómo fueron variando estos paquetes por día. Destaca la gran cantidad de paquetes de control.

7.2.10. Comparación entre tráfico enviado y recibido por los APs

Se realizó una comparación entre todo el tráfico enviado y recibido por los APs. En la figura 7.25 se realiza esta comparación por cantidad de paquetes, mientras que en la figura 7.26 se realiza por tráfico.

Como era de esperarse, el tráfico que se origina en los APs es ampliamente superior al generado desde los clientes. En el caso de los paquetes de control, es prácticamente la mitad para los APs y la mitad para los clientes.

	09/06	10/06	11/06	12/06	13/06	14/06	15/06
Association Request	13	45	3	3	50	28	0
Reassociation Request	2	1	10	3	0	0	0
Probe Request	19737	70188	80516	5678	28528	167371	0
Probe Response	18463	55	852	286	168	289	161
Disassociation	4	4	25	0	0	0	0
Authentication	85	13	41	18	4	332	74
Deauthentication	30	0	0	0	0	0	0
Desconocido	10	81	72	45	1259	70	0
PS-Poll	72	155	2	0	0	0	0
RTS	351080	55	0	0	0	0	0
CTS	61526	306851	122186	8652	2363	154030	121869
ACK	217473	3305	288094	356886	109150	548148	478334
CF End	142	299	8	83	34	0	0
Data	452	549	1178	421	252	36826	1246

TABLA 7.9: Evolución semanal de paquetes sin intervención de APs.

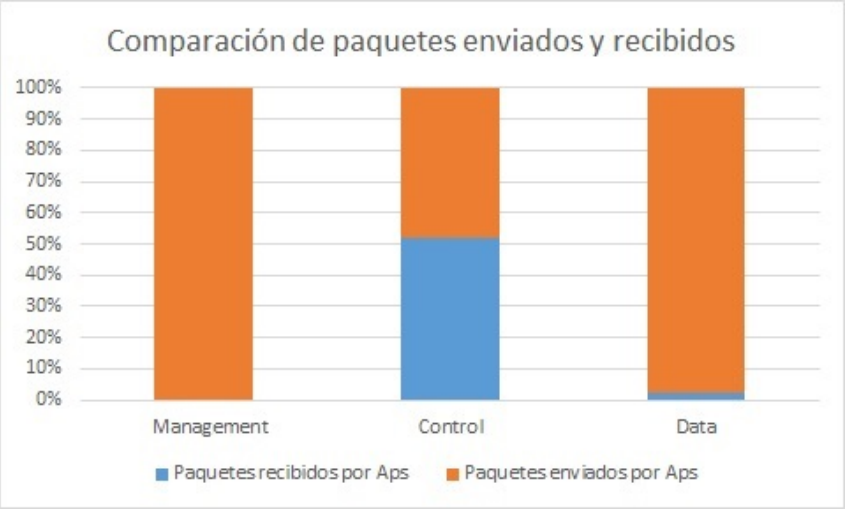

FIGURA 7.25: Comparación de cantidad de paquetes semanales enviados y recibidos por todos los APs.

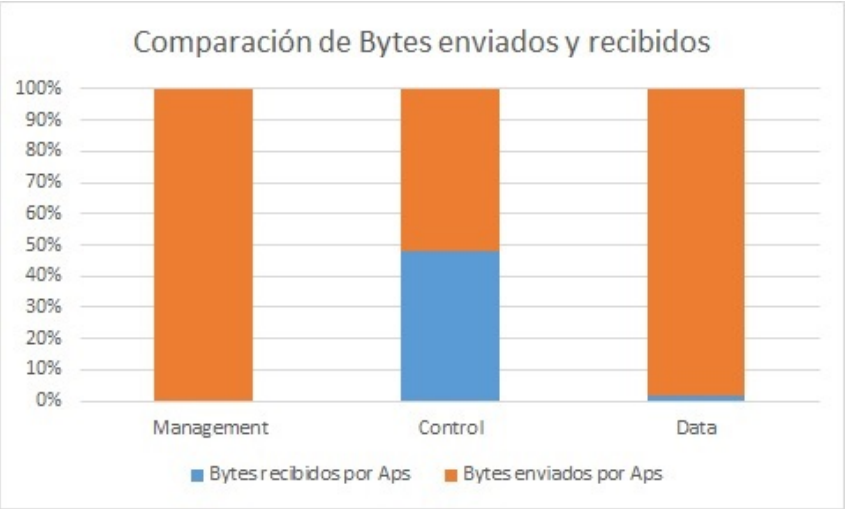

FIGURA 7.26: Comparación de tráfico semanal enviado y recibido por todos los APs.

7.2.11. Conclusiones de la evaluación

El análisis realizado, si bien abarca diferentes aspectos de la realidad de las redes en el InCo, no es exhaustivo. Podría investigarse en mayor profundidad, de forma de ampliar la información obtenida, pero dicho análisis excede el alcance de este proyecto.

Capítulo 8

Conclusiones y trabajo a futuro

En este capítulo se detallan las conclusiones obtenidas como resultado del proyecto, en las que se incluyen los resultados respecto a los objetivos planteados al comienzo y los puntos más importantes que surgieron a lo largo del mismo, y el trabajo futuro, en el que se detallan los puntos que por alguna razón no se pudo completar y aquellas oportunidades de mejora que se detectaron una vez finalizado el desarrollo.

8.1. Conclusiones

8.1.1. Investigación

Una gran parte del proyecto consistió en la investigación del estado del arte de las áreas de interés realacionadas a la temática del proyecto, entre las cuales se incluyen los hardware de routers inalámbricos, los firmware de código abierto para los mismos, los lenguajes de programación para sistemas embebidos y las arquitecturas de monitorización.

A su vez, se estudiaron como marco teórico el funcionamiento de las redes inalámbricas en general, la familia de redes 802.11 y su funcionamiento, los problemas característicos del uso de las mismas, los posibles indicadores de calidad a utilizar, los detalles del hardware (tanto para aquellos disponibles como los últimos modelos de los fabricantes más relevantes) y las herramientas existentes en materia de monitorización de redes inalámbricas.

El análisis de la información recabada permitió tomar las decisiones pertinentes para comenzar el desarrollo de la herramienta de monitorización. Si bien en algunos casos las decisiones fueron condicionadas por limitantes existentes, como en los casos del hardware y la disponibilidad del mismo en el grupo de investigación MINA o de los indicadores de calidad a utilizar y la capacidad del hardware y/o el software de obtener esos indicadores, en la amplia mayoría de los

casos las decisiones se desprenden del resultado de la investigación realizada, como en la elección del firmware, del lenguaje de programación a utilizar o de la arquitectura de monitorización.

Y aunque de cierta parte de la investigación no se desprenda ninguna decisión que defina el desarrollo de la herramienta, el estudio realizado resultó fundamental para el desarrollo de la misma. En particular, el conocimiento adquirido de las herramientas existentes permitió seleccionar las características deseables a implementar en nuestra herramienta, mientras que el dominio del funcionamiento de la familia de redes 802.11 y los detalles de su trama facilitaron enormemente la implementación a bajo nivel del parser de las capturas.

A modo de recapitulación, las decisiones que se desprenden de la investigación realizada son:

- La utilización de OpenWRT como firmware, dado que es el único que ofrece un file system, tiene buena documentación y gran apoyo de su comunidad, se utiliza frecuentemente dentro del grupo de investigación, y es abierto, gratuito y compatible con el hardware disponible.
- La utilización de C/C++ como lenguaje de programación para desarrollar la herramienta de monitorización, siendo que es un lenguaje con el que se posee experiencia, es compatible con el firmware, permite un alto control del hardware y ofrece independencia frente al procesador sobre el que vaya a ejecutar el programa.
- La utilización de una arquitectura centralizada para implementar la herramienta de monitorización, ya que se deseaba disponer de toda la información en una misma base de datos, no era relevante la toma de decisiones sobre acciones a ejecutar por parte de los nodos monitores y era la opción más sencilla que cumplía los requisitos.

8.1.2. Desarrollo

El desarrollo de la herramienta de monitorización representa la parte central de los objetivos de este proyecto, y el éxito de la misma al ser desplegada en el ambiente de producción del InCo, logrando obtener información relevante de las redes inalámbricas monitorizadas, indica que se cumplió el objetivo. Con la herramienta desarrollada es posible realizar análisis de redes inalámbricas densas, obtener información relevante para la planificación de las mismas, e incluso analizar el comportamiento del protocolo 802.11.

Adicionalmente, el hecho de haber completado el análisis de los datos obtenidos durante el despliegue, detectando problemas y particularidades a mejorar de las redes, centrando el foco en la planificación y en la mejora de la calidad del servicio, es una prueba de la utilidad de la herramienta. El ejemplo de uso de la herramienta para conocer y entender el comportamiento de una red inalámbrica densa y detectar problemas respalda la afirmación de que se cumplió con el objetivo.

En otro orden, es importante destacar que entre los objetivos iniciales se encontraba el de desarrollar metodologías que permitan evaluar las soluciones propuestas, pero el mismo se cambió en el transcurso del proyecto por el caso de estudio consistente en el despliegue de la herramienta desarrollada en la red productiva del InCo, y el posterior análisis de los datos obtenidos. Resulta claro por lo descrito en los párrafos anteriores que este objetivo también fue cumplido.

8.2. Trabajo a futuro

Dado que en la base de datos se guarda todo lo capturado, ante una ejecución prolongada, o sucesivas ejecuciones, se puede generar una gran cantidad de tuplas. Por lo tanto, sería interesante poder reducir la misma sin perder información que pueda resultar útil ni restar velocidad a las consultas que se ejecuten. El hecho de mantener un volumen de datos acotado se ataca, en parte, con la migración a las tablas históricas, lo cual permite tener valores "en vivo" sobre una tabla acotada haciendo que no se pierda velocidad en el acceso a esos datos. Sin embargo, este enfoque no soluciona el problema de fondo, ya que la tabla histórica pasa a ser la que padece el problema, y las consultas que no se ejecuten sobre los "en vivo" mantienen la penalización de performance de ejecutar en una tabla excesivamente poblada.

Se podría atacar esta problemática de otra forma: en lugar de mover los datos a otras tablas, podrían ser compactados de forma que ocupen menos espacio. Este compactamiento se podría llevar adelante identificando paquetes iguales o muy similares existentes en la base, donde, en lugar de tenerlos duplicados, se los podría juntar de cierta forma reduciendo la cantidad de paquetes. En la misma línea, podría realizarse manualmente un hashing de las direcciones MAC registradas, provocando que las tuplas que contienen los datos de las capturas pasen de almacenar tres strings de 17 caracteres a almacenar tres identificadores numéricos, reduciendo significativamente el espacio requerido para almacenar cada tupla. Éste enfoque no sólo presenta una mejora en cuanto al espacio de almacenamiento, sino que además mejora la performance de las consutlas ejecutadas sobre esta tabla, dado que el manejador debe realizar una cantidad significativamente menor de lecturas de bloques de disco para recorrer la tabla, ahorrándose accesos al componente más lento del servidor (ya que se trata del único componente mecánico).

La interfaz desarrollada es sencilla y ágil, no está pensada para ser una interfaz de usuario completa. Por lo tanto, el tamaño que ocupa la misma no está pensada para resoluciones de pantalla pequeñas. Esto ocurre por la necesidad de mostrar los diferentes resultados de las consultas en tablas que sean agradables a la vista y permitan una buena lectura de la información presentada. Podría mejorarse la herramienta con una interfaz que logre adaptarse al tamaño de la consola donde se esté ejecutando. También cabe la posibilidad de mantener la interfaz en consola como se encuentra actualmente y realizar una aplicación cliente con su propia interfaz que consuma los datos. No se entró en detalle sobre este aspecto, dado que uno de los objetivos

fundamentales del proyecto era la toma de datos y no así la presentación de los mismos. A su vez, la interfaz se desarrolló como un prototipo para visualizar los resultados, por lo que no está preparada para desplegar en pantalla ni mantener la capacidad de interacción con la gran cantidad de datos a los que se enfrentaría en una red de gran tamaño o al uso de más nodos monitores. Para ello es necesario cambiar la forma en la que se agrupa la información a la hora de ser mostrada, y la forma en la que se interactúa con la aplicación.

Otro posible aspecto a atacar es la performance de las consultas que se realizan sobre la base de datos desde la herramienta. Podrían estudiarse las mismas en profundidad para intentar optimizarlas, reduciendo el tiempo de ejecución y en consecuencia el tiempo de espera del usuario.

Al procesar y manejar una cantidad de información importante, sería posible realizar técnicas de Big Data sobre los datos que se tienen para realizar análisis más potentes.

Sería interesante ejecutar el sistema en un ambiente donde se cuente con más nodos monitores (que a su vez requeriría un servidor central con más recursos computacionales), recabando así mayor información de la red o redes que se estén monitoreando. También podría realizarse la medición durante un tiempo mayor al realizado, disponiendo de estadísticas más fieles, se dispondría de un muestreo más grande y en consecuencia más representativo de la realidad.

Podría ser de utilidad integrar el sistema con algún mecanismo de toma de decisiones sobre la red, utilizando la información recabada, procesándola y decidiendo realizar ciertos cambios sobre la red. De este modo podría lograrse que la red se adapte a los cambios en cuanto al uso a los que se somete a lo largo del tiempo.

Listado de figuras

Listado de tablas

Apéndice A

Uso de la herramienta

A.1. Consideraciones para que la herramienta funcione correctamente

Para el monitor se debe tener en cuenta que:

- Se probó en la versión 12.09 Attitud Adjustment de OpenWRT.
- Opkg debe estar disponible.
- El script macState.sh debe encontrarse en la ruta configurada.
- El archivo dron.conf debe encontrarse en donde se vaya a ejecutar la herramienta y debe estar configurado correctamente.
- El router debe permitir el modo promiscuo y el mismo debe estar habilitado en la interfaz que se vaya a utilizar.
- Libpcap debe estar instalado.
- Libstdcpp debe estar instalado.
- El router debe tener conectividad con el servidor.
- La key para ssh debe estar configurada en el servidor.
- Antes de comenzar a ejecutar el monitor, dar de baja y volver a levantar la interfaz donde se vaya a capturar (aunque ya estuviese configurada en modo monitor).

Para el servidor se debe tener en cuenta que:

- La base de datos debe estar instalada y la estructura esperada debe estar armada.
- El archivo de configuración debe estar en el mismo lugar que el servidor y debe estar configurado correctamente.
- Debe haber un servidor ssh corriendo.
- Se debe tener creada las carpetas donde cada monitor mueva sus archivos (incluyendo la carpeta procesados y errores).

A.2. Funcionamiento de la herramienta

El uso de la herramienta no presenta mucha dificultad, una vez se esté seguro de que se cumplen los puntos de la sección anterior, se debe instalar el dron.opkg en los routers, levantar el servidor, y ejecutar los monitores. La recepción y procesamiento de datos comenzará.

La ejecución del monitor es sencilla, ya que simplemente se ejecuta sin tener que hacer nada más. En el servidor, en cambio, se tiene un menú con diferentes opciones, las cuales son bastante intuitivas, utilizando los números y letras del teclado se va accediendo a las diferentes opciones.

Las consultas pre-hechas que se encuentran en la herramienta son:

- Información general de APs: retorna, agrupado por AP y monitor, información de cada AP (SSID, MAC, Canal y señales).
- Uso de aire general: retorna, agrupado por monitor, las estadísticas completas de uso del aire.
- Información completa por AP y monitor: retorna la cantidad de paquete de cada combinación de tipo y subtipo que pasan por cada AP (tanto entrantes como salientes) según lo que capturó cada monitor y el total de bytes que circularon de ese tipo por ese AP.
- Mayor cantidad de paquetes: retorna los 10 clientes que más paquetes hayan enviado en la red, indicando también el tamaño del total enviado por cada uno de ellos.
- Mayor cantidad de datos: retorna los 10 clientes que hayan enviado mayor cantidad de datos en la red, indicando también la cantidad de paquetes enviados por cada uno de ellos.
- Uso del aire contra tipos de paquete capturados por minuto: retorna las estadísticas de uso del aire de cada monitor agrupadas por minuto junto a los tipos de paquetes que se transfirieron en ese mismo intervalo.

Apéndice B

Base de datos

B.1. Pasos para instalar y configurar la base de datos

- Se instaló el motor.

  ```
  sudo apt-get install postgresql-9.1
  ```

- Se cambió el password del usuario “postgres” y se creó un nuevo usuario “pgmonitoreo”.
- Para conectarse a la base mediante psql se utilizó.

  ```
  psql -h localhost -d wifi_qos -U pgmonitoreo
  ```

B.2. Script para generar las tablas

El siguiente script genera las tablas necesarias para el funcionamiento de la herramienta:

```
CREATE TABLE ReadingSamples(
   MonitorId CHAR(17) NOT NULL,
   TimeStamp BIGINT NOT NULL,
   SourceMAC CHAR(17) NOT NULL,
   DestinationMAC CHAR(17) NOT NULL,
   Signal SMALLINT NOT NULL,
   Channel SMALLINT NOT NULL,
   PacketType SMALLINT NOT NULL,
   PacketSubtype SMALLINT NOT NULL,
   SSID CHAR(32),
```

```
   PacketSize INT NOT NULL,
   PRIMARY KEY( MonitorId, TimeStamp )
);

CREATE TABLE AirUsage(
   MonitorId CHAR(17) NOT NULL,
   TimeStamp BIGINT NOT NULL,
   SamplingTime INT NOT NULL,
   BusyTime INT NOT NULL,
   RecieveTime INT NOT NULL,
   BusyRatio FLOAT NOT NULL,
   PRIMARY KEY( MonitorId, TimeStamp )
);

CREATE TABLE ReadingSamples_Hist(
   MonitorId CHAR(17) NOT NULL,
   TimeStamp BIGINT NOT NULL,
   SourceMAC CHAR(17) NOT NULL,
   DestinationMAC CHAR(17) NOT NULL,
   Signal SMALLINT NOT NULL,
   Channel SMALLINT NOT NULL,
   PacketType SMALLINT NOT NULL,
   PacketSubtype SMALLINT NOT NULL,
   SSID CHAR(32),
   PacketSize INT NOT NULL,
   PRIMARY KEY( MonitorId, TimeStamp )
);

CREATE TABLE AirUsage_Hist(
   MonitorId CHAR(17) NOT NULL,
   TimeStamp BIGINT NOT NULL,
   SamplingTime INT NOT NULL,
   BusyTime INT NOT NULL,
   RecieveTime INT NOT NULL,
   BusyRatio FLOAT NOT NULL,
   PRIMARY KEY( MonitorId, TimeStamp )
);
```

Apéndice C

Código

C.1. Script macState.sh

```
#!/bin/ash
#Recibe como parámetros
#nombre de archivo - ruta donde guardarlo - cantidad de lecturas por paquete
iw wlan0 survey dump > /tmp/tmpDump
activeOld=‘cat /tmp/tmpDump | awk ’{if ($4 == "[in")
 {getline;getline; print $4}} ’‘
busyOld=‘cat /tmp/tmpDump | awk ’{if ($4 == "[in")
 {getline;getline;getline; print $4}} ’‘
rxOld=‘cat /tmp/tmpDump | awk ’{if ($4 == "[in"
 {getline;getline;getline;getline; print $4}} ’‘
txOld=‘cat /tmp/tmpDump | awk ’{if ($4 == "[in")
 {getline;getline;getline;getline;getline; print $4}} ’‘
initTime=$activeOld
#estas variables son para el tema de escribir un nuevo archivo cada cierto
# tiempo
nuevo=0
tope=$3
contador=0
archivo="$2/script$1$contador.csv"
contador=‘expr $contador + 1‘
temporal="procesando"
while true
do
```

```
   sleep 1
   iw wlan0 survey dump > /tmp/tmpDump
   activeNew=‘cat /tmp/tmpDump | awk ’{if ($4 == "[in")
    {getline;getline; print $4}}’‘
   busyNew=‘cat /tmp/tmpDump | awk ’{if ($4 == "[in"
   ) {getline;getline;getline; print $4}}’‘
   rxNew=‘cat /tmp/tmpDump | awk ’{if ($4 == "[in")
    {getline;getline;getline;getline; print $4}}’‘
   txNew=‘cat /tmp/tmpDump | awk ’{if ($4 == "[in")
    {getline;getline;getline;getline;getline; print $4}}’‘
   active=‘expr $activeNew - $activeOld‘
   time=‘date +’%s’‘
   tx=‘expr $txNew - $txOld‘
   rx=‘expr $rxNew - $rxOld‘
   busy=‘expr $busyNew - $busyOld‘
   idle=‘expr $active - $busy - $tx‘
   busyRatio=‘echo "scale=4; ($busy-$tx)/($active-$tx)" | bc -l‘
   # echo $time $active $busy $rx $tx $busyRatio
   # time = Tiempo con precisión en milisegundos de toma de los datos.
   #  active = duración de la muestra
   # busy = duración ocupado en la muestra
   # (tiempo en que no pudo transmitir si hubiera querido, medio ocupado)
   # rx = duración que estuvo recibiendo paquetes que fue procesado
   # tx = 0
   # busyRatio = porcentaje de ocupado
   echo "$time,$active,$busy,$rx,$tx,$busyRatio" >> $temporal
   activeOld=$activeNew
   busyOld=$busyNew
   rxOld=$rxNew
   txOld=$txNew
   nuevo=‘expr $nuevo + 1‘
   if [ $nuevo == $tope ]
   then
      mv "$temporal" "$archivo"
      nuevo=0
      archivo="$2/script$1$contador.csv"
      contador=‘expr $contador + 1‘
   fi
done
```

C.2. Servidor

Se presentan las principales partes del código que ejecuta en el servidor.

Guardado de datos en la base.

```
// Función invocada por hilo secundario, se encarga de actualizar los datos
// en la BD
void* FuncionHilo(void*)
{
  try
  {
    while(continuar)
    {
      sleep(tiempoEntreLecturas);
      // Se accede a la carpeta rutaDirCapturas
      vector<string> ficheros = vector<string>();
      if (GetDir(rutaDirCapturas,ficheros) != -1)
      {
        for (int i = 0;i < ficheros.size();i++)
        {
          // Se entra en cada directorio en busca de los archivos
          vector<string> lecturas = vector<string>();
          // Se abre cada una de las carpetas
          if ((ficheros[i] != ".") && (ficheros[i] != "..") &&
           GetDir(rutaDirCapturas + "/" + ficheros[i],lecturas) != -1)
          {
            for (int j = 0;j < lecturas.size();j++)
            {
              // se revisa cada uno de los ficheros para ver si son capturas
              // y si son se procesa
              if(lecturas[j].find(prefijoInfoDron) == 0)
              {
                // Como es dron se abre el archivo y se procesa
                //ifstream infoDron;
                string archivoCaptura = rutaDirCapturas + "/" + ficheros[i]
                 + "/" + lecturas[j];
                unsigned int size = (97 + sizeof(int)*5 +
                 sizeof(long long int)) * 2 * cantidadMaximaPaquetes;
```

```
char* res = (char*) malloc(size * sizeof(char));
memset(res, '\0', size);

if(!parse((char*) archivoCaptura.c_str(), res, 0))
{
  string captura = res;
  list<vector<string> > caps;
  while(!captura.empty())
  {
    string lineaCompleta;
    // Se obtiene la linea de una captura
    int inicio = captura.find("#");
    captura = captura.substr(inicio+1);
    int fin = captura.find("#");
    // SI fin es -1 es xq no hay mas capturas q la actual
    if (fin == -1)
    {
      fin = captura.length();
      lineaCompleta = captura.substr(0, fin);
      captura = "";
    }
    else
    {
      lineaCompleta = captura.substr(0, fin);
      captura = captura.substr(fin);
    }
    // Los valores q se obtiene al separar por coma son los
    // q se inserta en la BD
    stringstream linea(lineaCompleta);
    string aux;
    string monitor = ficheros[i];
    // TS
    getline(linea, aux, ',');
    string timestamp = aux;
    // Origen
    getline(linea, aux, ',');
    string origen = aux;
    // Destino
    getline(linea, aux, ',');
```

```
  string destino = aux;
  // Señal
  getline(linea, aux, ',');
  string signal = aux;
  // Canal
  getline(linea, aux, ',');
  string channel = aux;
  // Tipo
  getline(linea, aux, ',');
  string type = aux;
  // Sub Tipo
  getline(linea, aux, ',');
  string subtype = aux;
  // SSID
  getline(linea, aux, ',');
  string ssid = aux;
  // PacketSize
  getline(linea, aux, ',');
  string packetSize = aux;
  vector<string> capturaIndividual;
  capturaIndividual.push_back(monitor);
  capturaIndividual.push_back(timestamp);
  capturaIndividual.push_back(origen);
  capturaIndividual.push_back(destino);
  capturaIndividual.push_back(signal);
  capturaIndividual.push_back(channel);
  capturaIndividual.push_back(type);
  capturaIndividual.push_back(subtype);
  capturaIndividual.push_back(ssid);
  capturaIndividual.push_back(packetSize) ;
  caps.push_back(capturaIndividual);
}
// Se inserta en la BD la captura entera
string res = Insert80211Packets(caps);
if(res != "OK")
{
  cout << "No ha sido posible insertar un paquete en la BD: "
   << res << endl;
  // Se mueve el archivo procesado a "/errores"
```

```
        ifstream src(archivoCaptura.c_str(), std::ios::binary);
        string destino = rutaDirCapturas + "/" + ficheros[i]
         + "/errores/" + lecturas[j];
        ofstream dst(destino.c_str(), std::ios::binary);
        dst << src.rdbuf();
      }
      else
      {
        // Se mueve el archivo procesado a "/procesados"
        ifstream src(archivoCaptura.c_str(), std::ios::binary);
        string destino = rutaDirCapturas + "/" + ficheros[i]
         + "/procesados/" + lecturas[j];
        ofstream dst(destino.c_str(), std::ios::binary);
        dst << src.rdbuf();
      }
      remove(archivoCaptura.c_str());
    }
    else
    {
      // Se mueve el archivo procesado a "/errores"
      ifstream src(archivoCaptura.c_str(), std::ios::binary);
      string destino = rutaDirCapturas + "/" + ficheros[i]
       + "/errores/" + lecturas[j];
      ofstream dst(destino.c_str(), std::ios::binary);
      dst << src.rdbuf();
      remove(archivoCaptura.c_str());
      cout << "Error al querer leer una captura de un monitor,
       el archivo " << lecturas[j] << " ha sido movido a /errores"
        << endl;
    }
    // Se libera lo reservado para el res del parser
    free(res);
  }
  else if (lecturas[j].find(prefijoInfoScript) == 0)
  {
    // Como es script se abre el archivo y se procesa
    ifstream infoScript;
    string archivoCaptura = rutaDirCapturas + "/" + ficheros[i]
     + "/" + lecturas[j];
```

```
infoScript.open(archivoCaptura.c_str());
// Si se abre correctamente
if (infoScript.is_open())
{
  string lineaCompleta;
  list<vector<string> > airLines;
  // Se procesa el archivo por linea
  while(getline(infoScript, lineaCompleta))
  {
    stringstream linea(lineaCompleta);
    if (lineaCompleta.length() != 0)
    {
      // Split por comas
      string aux;
      // Monitor
      string monitor = ficheros[i];
      // TS
      //MonitorId,TimeStamp,SamplingTime,BusyTime,
      //RecieveTime,BusyRatio
      getline(linea, aux, ',');
      // Se le agrega 3 ceros para matchear la precision de
      // los paquetes de las capturas
      string timestamp = aux + "000000";
      // SamplingTime
      getline(linea, aux, ',');
      string samplingTime = aux;
      // BusyTime
      getline(linea, aux, ',');
      string busyTime = aux;
      // RecieveTime
      getline(linea, aux, ',');
      string recieveTime = aux;
      // TX NO importa!, es 0 xq esta en modo promiscuo
      getline(linea, aux, ',');
      // BusyRatio
      getline(linea, aux, ',');
      string busyRatio = aux;

      // Se agrega a las líneas de airusage
```

```
            vector<string> airIndividual;
            airIndividual.push_back(monitor);
            airIndividual.push_back(timestamp);
            airIndividual.push_back(samplingTime);
            airIndividual.push_back(busyTime);
            airIndividual.push_back(recieveTime);
            airIndividual.push_back(busyRatio);
            airLines.push_back(airIndividual);
          }
        }
        // Se inserta en la BD la captura entera
        string res = InsertAirSamples(airLines);
        if(res != "OK")
        {
          cout << "No ha sido posible insertar un paquete en la BD: "
           << res << endl;
          // Se mueve el archivo procesado a "/errores"
          ifstream src(archivoCaptura.c_str(), std::ios::binary);
          string destino = rutaDirCapturas + "/" + ficheros[i]
           + "/errores/" + lecturas[j];
          ofstream dst(destino.c_str(), std::ios::binary);
          dst << src.rdbuf();
        }
        else
        {
          // Se mueve el archivo procesado a "/procesados"
          ifstream src(archivoCaptura.c_str(), std::ios::binary);
          string destino = rutaDirCapturas + "/" + ficheros[i]
           + "/procesados/" + lecturas[j];
          ofstream dst(destino.c_str(), std::ios::binary);
          dst << src.rdbuf();
        }
        remove(archivoCaptura.c_str());
      }
      else
      {
        // Se mueve el archivo procesado a "/errores"
        ifstream src(archivoCaptura.c_str(), std::ios::binary);
        string destino = rutaDirCapturas + "/" + ficheros[i]
```

```
                    + "/errores/" + lecturas[j];
                   ofstream dst(destino.c_str(), std::ios::binary);
                   dst << src.rdbuf();
                   remove(archivoCaptura.c_str());
                   cout << "Error al querer leer una captura de un script,
                    el archivo " << lecturas[j]
                    << " ha sido movido a /errores" << endl;
                 }
               }
             }
           }
         }
       }
       else
         cout << "Error al querer leer la información de los drones" << endl;
     }
   }
   catch (int e)
   {
     cout << textoError << " Código Hilo" << e <<  endl;
   }
}
```

Parseo de captura ".pcap".

```
// Procesamiento de una captura resultado de la extracción de datos
// Bandera de debug, si está en 1 imprime lo q va tomando
int parse(char* infile, char* res, int debug) {
   pcap_t *pcap_handle_in;
   char pcap_errbuf[PCAP_ERRBUF_SIZE];
   const u_char *pcap_packet;
   struct pcap_pkthdr *pcap_header;
   struct bpf_program pcap_filter;
   // open infile
   pcap_handle_in = pcap_open_offline(infile, pcap_errbuf);
   if (pcap_handle_in == NULL) {
      fprintf(stderr, "No ha sido posible abrir el archivo: %s\n",
         pcap_errbuf);
      return 2;
```

```
}
int fallido = 0;
while (pcap_next_ex(pcap_handle_in, &pcap_header, &pcap_packet) > 0) {
   // se obtiene el timestamp del paquete capturado
   long long int seconds = pcap_header->ts.tv_sec;
   long long int microseconds = pcap_header->ts.tv_usec;
   int packetSize = pcap_header->len;
   if(debug)
      printf("\nPaquete capturado a las %lld,%lld.\n", seconds,
         microseconds);
   // se referencia el header de radiotap segun el comienzo del cuerpo
   // del paquete pcap
   struct ieee80211_radiotap_header *buf =
      (struct ieee80211_radiotap_header*) pcap_packet;
   int buflen = buf->it_len;
   if(debug)
      printf("Largo del header 802.11_radiotap = %d\n", buflen);
   // movida de parseo del radiotap header pro
   int channel = 0;
   char signal = '0';
   struct ieee80211_radiotap_iterator iterator;
   // inicializo el iterador
   int ret = ieee80211_radiotap_iterator_init(&iterator, buf, buflen, NULL);
   while (ret == 0) {
      // avanzo al siguiente parametro del header de radiotap
      ret = ieee80211_radiotap_iterator_next(&iterator);
      if (ret != 0)
         continue;
      //see if this argument is something we can use
      switch (iterator.this_arg_index) {
         case IEEE80211_RADIOTAP_DBM_ANTSIGNAL :{
            signal = (*iterator.this_arg);
            if(debug)
               printf("señal = %d\n", signal);
            break;}
         case IEEE80211_RADIOTAP_RX_FLAGS:
            if (*iterator.this_arg & 0x0001) {
            fallido = 1;
            if(debug)
```

```
            printf("Paquete fallido\n");
         }
         break;
      case IEEE80211_RADIOTAP_CHANNEL:{
         char* frequency = iterator.this_arg;
         // contine dos valores de 16 bits. en el 1ero esta la
         // frecuencia, y en el 2do las flags.
         int fH = frequency[1]*256;
         // leemos la frecuencia en little endian y armamos el valor
         int fL = frequency[0]&0xff;
         channel = GetChannel(fH+fL);
         if(debug)
            printf("Canal = %d\n", channel);
            // convertimos la frecuencia en el canal correspondiente
         }
         break;
      default:
         break;
      }
} // while mas rt headers
if (ret != -ENOENT){
   if(debug)
      printf("-1\n");
}
// fin movida pro con radiotap
// se controla que el paquete procesado no haya fallado
if (fallido){
   fallido = 0;
   continue;
}
// arrancamos a procesar el paquete 802.11 ninja style
char* paquete80211 = (char *) buf;
paquete80211 += buflen;
// obtenemos el tipo y el subtipo
int type = (paquete80211[0] & 0x0f) >> 2;
int subtype = ((paquete80211[0] & 0xf0) >> 4);
if(debug)
   printf("Tipo = %d\nSubtipo = %d\n", type, subtype);
// variables para obtener source y destination MACs
```

```
char flags = paquete80211[1] & 0x03;
unsigned char dest[6];
unsigned char destination[18];
unsigned char src[6];
unsigned char source[18] = "\0";
// se extraen los 6 bytes que contienen la MAC de destino, y se le da
// el formato apropiado
memcpy(dest, &paquete80211[4], 6);
snprintf(destination, 18,"%02x:%02x:%02x:%02x:%02x:%02x", dest[0],
   dest[1], dest[2], dest[3], dest[4], dest[5]);
if(debug)
   printf("Destination Address: %s\n", destination);
// se controla que el paquete no sea CTS, porque en ese caso no
// trae MAC de origen
if (type != 0x01 || subtype != 0x0c){
   // se extraen los 6 bytes que contienen la MAC de origen, y se le da
   // el formato apropiado
   memcpy(src, &paquete80211[10], 6);
   snprintf(source, 18,"%02x:%02x:%02x:%02x:%02x:%02x", src[0], src[1],
      src[2], src[3], src[4], src[5]);
   if(debug)
      printf("Source Address: %s\n", source);
}
// se captura el SSID del lan management frame
unsigned char ssid[33] = "\0";
if (type == 0x00 && subtype == 0x08){
   // salto los 24 bytes del header MAC y los 12 de los params fijos del
   // header
   char * mngmntHdr = &paquete80211[36];
   // copio el SSID a mi variable segun el largo declarado en el header
   memcpy(ssid, &mngmntHdr[2], mngmntHdr[1]);
   if(debug)
      printf("SSID: %s\n", ssid);
}
// controla que no se cuelen paquetes fallidos
if (channel > 0 && signal <= 0){
   strcat(res, "#");
   char strAux[100];
   sprintf(strAux, "%lld", seconds*1000000 + microseconds);
```

```
            strcat(res, strAux);
            strcat(res, ",");
            strcat(res, source);
            strcat(res, ",");
            strcat(res, destination);
            strcat(res, ",");
            sprintf(strAux, "%d", signal);
            strcat(res, strAux);
            strcat(res, ",");
            sprintf(strAux, "%d", channel);
            strcat(res, strAux);
            strcat(res, ",");
            sprintf(strAux, "%d", type);
            strcat(res, strAux);
            strcat(res, ",");
            sprintf(strAux, "%d", subtype);
            strcat(res, strAux);
            strcat(res, ",");
            strcat(res, ssid);
            strcat(res, ",");
            sprintf(strAux, "%d", packetSize);
            strcat(res, strAux);
         }
      }
      pcap_close(pcap_handle_in);
      return 0;
}
```

C.3. Monitor

Se presentan las principales partes del código que ejecuta en los nodos monitores.

Captura de paquetes:

```
bool inicializarCaptura()
{
   try
   {
```

```
      // Si se quiere tomar una por defecto sería esto
      // dev = pcap_lookupdev(lpcapErrbuf);
      // Se inicia la captura, en la interfaz configurada, modo promiscuo y
      // con un timeout dado
      if (!(lpcapCaptura = pcap_open_live(lpcapDevice, BUFSIZ, 1,
            refresco*1000, lpcapErrbuf)))
      {
         return false;
      }
      else
      {
         pcap_set_datalink(lpcapCaptura, DLT_IEEE802_11_RADIO);
         // La parte de archivo hay que hacerla cada vez
         return true;
      }
   }
   catch (int e)
   {
      cout << textoError << e <<  endl;
      return false;
   }
}

bool capturar()
{
   try
   {
      // Se arma el nombre del archivo del archivo
      time_t marcaTiempo = time(0);
      stringstream temp;
      temp << marcaTiempo;
      string nombreTemp = rutaCapturas + "/" + prefijoCapturasPaquetes + "_"
         + temp.str() + ".pcap";
      strcpy(lpcapArchivo, nombreTemp.c_str());
      // Se setea el DUMP al archivo indicado
      if ((lpcapDump = pcap_dump_open(lpcapCaptura, lpcapArchivo)) == NULL)
      {
         return false;
      }
```

```
      // Lee HASTA maxPaquetes, pero si hay menos luego del timeout, retorna
      // con menos
      pcap_loop(lpcapCaptura, maxPaquetes, pcap_dump,
         (unsigned char *)lpcapDump);
      pcap_dump_close(lpcapDump);
      return true;
   }
   catch (int e)
   {
      cout << textoError << e <<  endl;
      return false;
   }
}
```

Envío de datos al servidor:

```
// Envía los archivos que haya en la carpeta especificada con las capturas
// a enviar
bool EnviarDatos()
{
   try
   {
      // Cantidad de capturas que se enviaron en este intento de enviar
      // (NO es global, el global es cantCapturasEnviadas)
      int cantEnviar = 0;
      // Se abre la carpeta donde se tienen las capturas
      vector<string> ficheros = vector<string>();
      if (GetDir(rutaCapturas,ficheros) != -1)
      {
         // Se recorre lo que haya
         for (int i = 0;i < ficheros.size();i++)
         {
            // Se buscan archivos que cumplan el prefijo
            if((ficheros[i].find(prefijoCapturasPaquetes) == 0) ||
               (ficheros[i].find(prefijoCapturasMatias) == 0))
            {
               cantEnviar++;
               // Se obtiene la info del archivo a enviar
               string rutaArchivo = rutaCapturas + "/" + ficheros[i];
```

```
                // Se envía el archivo usando scp y las calls a system
                string scp = "scp " + rutaArchivo + " " + usuarioServidor + "@"
                   + ipServidor + ":" + rutaServidor + "/" + dispositivoID
                   + " -i ~/.ssh/id_rsa";
                system(scp.c_str());
                // TIENEN QUE ESTAR LAS CARPETAS REMOTAS ANTES!!!
                // Se borra el fichero
                remove(rutaArchivo.c_str());
                cantCapturasEnviadas++;
             }
          }
          // Para poner la hora de envío
          time_t tiempo = time(0);
          tm * ptm = localtime(&tiempo);
          char buffer[32];
          strftime(buffer, 32, "%m/%d/%Y %H:%M:%S", ptm);
          if (cantEnviar == 1)
             cout << "Se envió un archivo al servidor - " << buffer << endl;
          else if (cantEnviar > 1)
             cout << "Se enviaron " << cantEnviar << " archivos al servidor - "
                 << buffer << endl;
          else
             cout << "No se enviaron archivos al servidor - " << buffer<< endl;
       }
       return true;
    }
    catch (int e)
    {
       cout << textoError << e <<  endl;
       return false;
    }
}
```

Apéndice D

Compilación para OpenWRT

D.1. Paso a paso para la compilación cruzada para OpenWRT

Teniendo el código fuente del paquete a compilar se deben realizar los siguientes pasos:

- Crear un directorio base con el nombre del paquete.
- Dentro del directorio se debe crear un subdirectorio "src" donde se colocará el código fuente.
- Crear un Makefile común para compilar ese código (como el de la sección siguiente).
- En el Makefile en lugar de utilizar "gcc" o "g++" se debe utilizar una variable que tenga el compilador, "$(CC)" o "$(CXX)", esto es importante para la compilación con OpenWRT, ya que no se cuenta con los compiladores estándares.
- Poner el Makefile creado en la carpeta src y compilar para asegurarse de que todo esté bien.
- Descargar (si no se tiene ya) el SDK de OpenWRT, el SDK varía según la arquitectura del equipo que se esté usando para hacer la compilación cruzada, la arquitectura del router donde se vaya a instalar y ejecutar el paquete y la versión de OpenWRT que se esté usando.
- Luego de descargar, extraer lo descargado y mover el directorio que se creó en el primer paso hacia la carpeta "package" de OpenWRT.
- Se debe crear un nuevo Makefile dentro del directorio movido en el punto anterior. Este Makefile es especial para realizar la compilación cruzada. En la última sección de este apéndice se presenta el Makefile de este tipo realizado, el cual se puede tomar como ejemplo para relizar otros Makefiles (se basa en el presentado en el artículo [148]).
- Es importante utilizar tabulador y no espacios para formatear este Makefile.

- Finalmente se debe ir a la carpeta raíz del SDK descargado y ejecutar "make".
- Si todo salió correctamente, dentro del directorio "bin/packages" se encuentra el paquete compilado.

D.2. Makefile común

```
dron: dron.o
    $(CXX) $(LDFLAGS) dron.o -o dron -lpcap

dron.o: dron.cpp
    $(CXX) $(CXXFLAGS) -c dron.cpp -lpcap

#borrar
clear:
    rm -rf *.o dron
```

D.3. Makefile OpenWRT

Se recalca nuevamente que para dar el formato al Makefile se debe utilizar tabulador y no espacios. Este Makefile se basa en el Makefile presentado en [148].

```
#############################################
# OpenWrt Makefile para el monitor
#
#
# Most of the variables used here are defined in
# the include directives below. We just need to
# specify a basic description of the package,
# where to build our program, where to find
# the source files, and where to install the
# compiled program on the router.
#
# Be very careful of spacing in this file.
# Indents should be tabs, not spaces, and
# there should be no trailing whitespace in
# lines that are not commented.
```

```
#
##############################################

include $(TOPDIR)/rules.mk

# Name and release number of this package
PKG_NAME:=dron
PKG_RELEASE:=1

# This specifies the directory where we're going to build the program.
# The root build directory, $(BUILD_DIR), is by default the build_mipsel
# directory in your OpenWrt SDK directory
PKG_BUILD_DIR := $(BUILD_DIR)/$(PKG_NAME)

include $(INCLUDE_DIR)/package.mk

# Specify package information for this program.
# The variables defined here should be self explanatory.
define Package/dron
    SECTION:=utils
    CATEGORY:=Utilities
    TITLE:=dron -- Proyecto de Grado - FING - UdelaR
    DEPENDS:=+libstdcpp +libpcap
#   DESCRIPTION:=\
#   If you can't figure out what this program does, \\\
#   you're probably brain-dead and need immediate \\\
#   medical attention.
endef

define Package/dron/description

endef

# Specify what needs to be done to prepare for building the package.
# In our case, we need to copy the source files to the build directory.
# This is NOT the default.  The default uses the PKG_SOURCE_URL and the
# PKG_SOURCE which is not defined here to download the source from the web.
# In order to just build a simple program that we have just written, it is
# much easier to do it this way.
```

```
define Build/Prepare
    mkdir -p $(PKG_BUILD_DIR)
    $(CP) ./src/* $(PKG_BUILD_DIR)/
endef

# We do not need to define Build/Configure or Build/Compile directives
# The defaults are appropriate for compiling a simple program such as this one

# Specify where and how to install the program. Since we only have one file,
# the helloworld executable, install it by copying it to the /bin directory on
# the router. The $(1) variable represents the root directory on the router running
# OpenWrt. The $(INSTALL_DIR) variable contains a command to prepare the install
# directory if it does not already exist.  Likewise $(INSTALL_BIN) contains the
# command to copy the binary file from its current location (in our case the build
# directory) to the install directory.
define Package/dron/install
    $(INSTALL_DIR) $(1)/bin
    $(INSTALL_BIN) $(PKG_BUILD_DIR)/dron $(1)/bin/
endef

# This line executes the necessary commands to compile our program.
# The above define directives specify all the information needed, but this
# line calls BuildPackage which in turn actually uses this information to
# build a package.
$(eval $(call BuildPackage,dron))
```

Bibliografía

[1] James F. Kurose y Keith W. Ross. *Redes de Computadores: Un Enfoque Descendente. Quinta edición*. Addison Wesley, 2010.

[2] Intel. Wireless ethernet lan (wlan) general 802.11a/802.11b/802.11g faq. Visitado en Octubre 2014. URL http://download.intel.com/support/wireless/wlan/wirelesslanfaq.pdf.

[3] Intel. Productos intel® wi-fi. http://www.intel.com/support/sp/wireless/wlan/sb/cs-025321.htm?wapkw=wlan, online. Visitado en Octubre 2014.

[4] Paul Vrancken Philips Research, CoSiNe. Ieee 802.11 medium access control (mac). http://www.wirelesscommunication.nl/reference/chaptr01/wrlslans/80211_page2.htm, online. Visitado en Octubre 2014.

[5] TechNet. How 802.11 wireless works. http://technet.microsoft.com/en-us/library/cc757419(v=ws.10).aspx, online. Visitado en Octubre 2014.

[6] Michael H. Warfield. Wireless security. 2006. URL http://www.wittsend.com/mhw/2006/Wireless-Security-ALE/Wireless-Security-ALE-2006.pdf.

[7] Jargen Bach Andersen, Theodore S. Rappaport, and Susumu Yoshida. Propagation measurements and models for wireless communications channels. 1995. URL http://www.cs.bilkent.edu.tr/~korpe/courses/cs515-fall2002/papers/radio-propogation-rappaport.pdf.

[8] IEEE. Modelo osi. 1994. URL http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip.

[9] R. Droms Network Working Group. Rfc 2131 - dynamic host configuration protocol. 2014. URL http://tools.ietf.org/html/rfc2131.

[10] IEEE. Cyclic codes for error detection. 2007. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=4066263.

[11] IEEE Ethernet Working Group. Ethernet 802.3. http://www.ieee802.org/3/, online. Visitado en Octubre 2014.

[12] Routerboard. About routerboard. http://routerboard.com/about, online. Visitado en Octubre 2014.

[13] MikroTik. R52. http://routerboard.com/R52, online. Visitado en Octubre 2014.

[14] MS Distribution. Mikrotik r52 802.11a/b/g mini-pci wireless. http://www.msdist.co.uk/product_MikroTik_R52_mini-PCI_wireless.php, online. Visitado en Octubre 2014.

[15] Linux Wireless. About ath5k. http://wireless.kernel.org/en/users/Drivers/ath5k, online. Visitado en Octubre 2014.

[16] Linux Wireless. Existing linux wireless drivers. http://wireless.kernel.org/en/users/Drivers, online. Visitado en Octubre 2014.

[17] Linux Wireless. Wireless operating modes. http://wireless.kernel.org/en/users/Documentation/modes, online. Visitado en Octubre 2014.

[18] TP-LINK. Router gigabit inalámbrico de banda dual n750 - tl-wdr4300. http://www.tp-link.es/products/details/?model=TL-WDR4300, online. Visitado en Abril 2015.

[19] OpenWRT. Tp-link tl-wdr4300. http://wiki.openwrt.org/toh/tp-link/tl-wdr4300, online. Visitado en Abril 2015.

[20] MikroTik. Routerboard 133 series - user's manual. 2006. URL http://routerboard.com/pdf/rb133ugA.pdf.

[21] MikroTik. Routerboard r52. http://i.mt.lv/routerboard/files/R52.pdf, online. Visitado en Octubre 2014.

[22] MikroTik. 2.4/5ghz 802.11a+b+g wireless mini-pci card (r52). http://www.mikrotik.com/pdf/R52.pdf, online. Visitado en Octubre 2014.

[23] TP-LINK. Tl-wdr4300 - user guide. http://www.tp-link.com/resources/document/TL-WDR4300_V1_User_Guide_19100.pdf, online. Visitado en Abril 2015.

[24] MikroTik. Routerboard product catalog q2 2014. 2014. URL http://download2.mikrotik.com/2014-Q2.pdf.

[25] Intel. ¿qué es un router inalámbrico? http://www.hp.com/global/uy/es/wireless/wireless-network-help4.html, online. Visitado en Octubre 2014.

[26] Intel. Redes inalámbricas, productos intel wi-fi. http://www.intel.com/support/sp/wireless/wlan/sb/cs-030505.htm?wapkw=wlan, online. Visitado en Octubre 2014.

[27] Oracle. Java plataform, micro edition (java me). http://www.oracle.com/technetwork/java/embedded/javame/overview/index.html, online. Visitado en Octubre 2014.

[28] Qualcomm. Qualcomm atheros. http://www.qca.qualcomm.com/, online. Visitado en Octubre 2014.

[29] Qualcomm. Product collateral. http://www.qca.qualcomm.com/resources/product-collateral/, online. Visitado en Octubre 2014.

[30] WikiDEV. Atheros. https://wikidevi.com/wiki/Atheros, online. Visitado en Octubre 2014.

[31] Intel. Intel wi-fi products. http://ark.intel.com/es/products/family/59484/Intel-Wi-Fi-Products], online. Visitado en Octubre 2014.

[32] Intel. Intel. http://www.intel.la/content/www/xl/es/homepage.html, online. Visitado en Octubre 2014.

[33] Intel. Intel wireless technology and products information. http://www.intel.la/content/www/xl/es/wireless-products/WirelessTechDocs.html?wapkw=wlan, online. Visitado en Octubre 2014.

[34] Broadcom. Broadcom company. http://www.broadcom.com/company/, online. Visitado en Octubre 2014.

[35] Broadcom. 802.11 wireless lan solutions. http://www.broadcom.com/products/Wireless-LAN/802.11-Wireless-LAN-Solutions, online. Visitado en Octubre 2014.

[36] Broadcom. Broadcom corporation quick facts. http://www.broadcom.com/docs/company/BroadcomQuickFacts.pdf, online. Visitado en Octubre 2014.

[37] Microsoft. ¿qué es un proveedor de acceso a internet (isp)? http://windows.microsoft.com/es-es/windows/what-is-internet-service-provider#1TC=windows-7, online. Visitado en Octubre 2014.

[38] Jesús Moreno León and Raúl Ruiz Padilla. Planificación y administración de redes: Network address traslation. http://www.exa.unicen.edu.ar/catedras/comdat1/material/NAT.pdf, online. Visitado en Octubre 2014.

[39] Pablo Estrada. Mimo: Why multiple antennas matter. https://meraki.cisco.com/blog/2011/02/mimo-why-multiple-antennas-matter/, online. Visitado en Octubre 2014.

[40] Karima Velásquez and Eric Gamess. Network performance evaluation based on three processes. 2014. URL http://pubs.sciepub.com/jcsa/2/2/1/jcsa-2-2-1.pdf.

[41] Francisco Ortuño López. Implementación openwrt del protocolo de comunicaciones w2lan. 2010. URL http://repositorio.bib.upct.es/dspace/bitstream/10317/1958/1/pfc3514.pdf.

[42] D. Grari Hicham. Plataforma móvil para aula virtual basada en wrt160nl-openwrt-usb. 2012. URL http://repositorio.bib.upct.es/dspace/bitstream/10317/3083/1/pfc4550.pdf.

[43] Florian Fainelli. The openwrt embedded development framework. 2008. URL ftp://193.206.140.34/pub/1/openwrt/people/florian/fosdem/fosdem.pdf.

[44] Aaron Weiss. The open source wrt54g story. http://www.wi-fiplanet.com/tutorials/article.php/3562391, online. Visitado en Octubre 2014.

[45] TheIndividual. The individual's sveasoft wrt54g firmwares. http://wrt54g.thermoman.de/, online. Visitado en Octubre 2014.

[46] Narod. Freww wrt initiative. http://freewrt.narod.ru/, online. Visitado en Octubre 2014.

[47] Sveasoft. Sveasoft. http://www.sveasoft.com/, online. Visitado en Octubre 2014.

[48] Sveasoft forum. None of the downloads are working. http://www.sveasoft.com/index.php?option=com_kunena&view=topic&catid=26&id=126156&Itemid=18, online. Visitado en Octubre 2014.

[49] Polarcloud. Tomato firmware. http://www.polarcloud.com/tomato, online. Visitado en Octubre 2014.

[50] Dd-wrt. About dd-wrt. http://www.dd-wrt.com/site/content/about, online. Visitado en Octubre 2014.

[51] Dd-wrt. What is dd-wrt? http://www.dd-wrt.com/wiki/index.php/What_is_DD-WRT%3F, online. Visitado en Octubre 2014.

[52] Dd-wrt. Dd-wrt supported devices. http://www.dd-wrt.com/wiki/index.php/Supported_Devices, online. Visitado en Octubre 2014.

[53] Dd-wrt. Router database. http://www.dd-wrt.com/site/support/router-database, online. Visitado en Octubre 2014.

[54] OpenWRT. Openwrt. https://openwrt.org/, online. Visitado en Octubre 2014.

[55] Jeremy Collake and Kemen. X-wrt. https://code.google.com/p/x-wrt/, online. Visitado en Octubre 2014.

[56] Tarifa. The wrt54gl enhanced firmware. http://tarifa.sourceforge.net/, online. Visitado en Octubre 2014.

[57] Michael Barr and Anthony Massa. *Programming Embedded Systems, Second Edition with C and GNU Development Tools*. O'Reilly, 2006.

[58] Peter Marwedel. *Embedded Systems Design, Second Edition*. Springer, 2011.

[59] Michael Barr. *Programming Embedded Systems in C and C++*. O'Reilly, 1999.

[60] Eugenio Villar (editor). *Embedded Systems Specification and Design Languages - Selected contributions from FDL'07*. Springer, 2007.

[61] Stephen A. Edwards. Design languages for embedded systems. 2003. URL http://www.cs.columbia.edu/~sedwards/papers/edwards2003design.pdf.

[62] A. Jantsch and I. Sander. Models of computation and languages for embedded system design. 2005. URL http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1454195&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F2192%2F31230%2F01454195.pdf%3Farnumber%3D1454195.

[63] Lua. Lua. http://www.lua.org/about.html, online. Visitado en Octubre 2014.

[64] J. Visca. Introducción a lua. https://eva.fing.edu.uy/pluginfile.php/58939/mod_resource/content/1/introduccionALua-v1.pdf, online. Visitado en Octubre 2014.

[65] Python. Python. https://www.python.org/, online. Visitado en Octubre 2014.

[66] David Boddie. Embedded python. https://wiki.python.org/moin/EmbeddedPython, online. Visitado en Octubre 2014.

[67] Lua Users. Lua multi tasking. http://lua-users.org/wiki/MultiTasking, online. Visitado en Octubre 2014.

[68] Mary Bellis. Inventors of the modern computer. http://inventors.about.com/od/mstartinventions/a/microprocessor.htm, online. Visitado en Octubre 2014.

[69] Adrián Granados. Diagnosticando problemas en redes inalámbricas. http://www.adriangranados.com/content/help/wifiexplorer/1.5/es.lproj/pgs/troubleshoot.html, online. Visitado en Octubre 2014.

[70] Alex Gzz. Redes de telecomunicaciones. http://redestele.blogspot.com/2013_02_01_archive.html, online. Visitado en Octubre 2014.

[71] Margaret Rouse. signal-to-noise ratio (s/n or snr). http://searchnetworking.techtarget.com/definition/signal-to-noise-ratio, online. Visitado en Octubre 2014.

[72] Xataka. Qué son los canales wi-fi y cómo escoger el mejor para nuestra red. http://www.xatakaon.com/optimizacion-del-adsl/que-son-los-canales-wi-fi-y-como-escoger-el-mejor-para-nuestra-red, online. Visitado en Octubre 2014.

[73] Juan V. Puertos. Montaje de un router wireless en una maquina gnu/linux. URL http://www.uv.es/~montanan/ampliacion/trabajos/nat_802.11_linux.pdf.

[74] Ambili Thottam Parameswaran, Mohammad Iftekhar Husain, and Shambhu Upadhyaya. Is rssi a reliable parameter in sensor localization algorithms – an experimental study. 2009. URL http://www.cse.buffalo.edu/srds2009/F2DA/f2da09_RSSI_Parameswaran.pdf.

[75] Angelos Vlavianos, Lap Kong Law, Ioannis Broustis, Srikanth V. Krishnamurthy, and Michalis Faloutsos. Assessing link quality in ieee 802.11 wireless networks: Which is the right metric? URL http://www2.research.att.com/~broustis/research/link_pimrc08.pdf.

[76] Daniel Aguayo, John Bicket, Sanjit Biswas, Glenn Judd, and Robert Morris. Link-level measurements from an 802.11b mesh network. URL http://pdos.csail.mit.edu/rtm/papers/p442-aguayo.pdf.

[77] Kameswari Chebrolu, Bhaskaran Raman, and Sayandeep Sen. Long-distance 802.11b links: Performance measurements and experience. 2006. URL http://pages.cs.wisc.edu/~sdsen/papers/2006dgpmeas.pdf.

[78] Susan Martínez Cordero. Análisis de la calidad de señal en una red wifi con la herramienta netstumbler. 2005. URL http://www.redalyc.org/articulo.oa?id=30400708.

[79] Jian Zhang and Ivan Marsic. Link quality and signal-to-noise ratio in 802.11 wlan with fading: A time-series analysis. 2006. URL http://www.ece.rutgers.edu/~marsic/Publications/vtc2006.pdf.

[80] T. Balla Z. Gal and A. Sz. Karsai. On the wifi interference analysis based on sensor network measurements. 2013.

[81] Thomas Hühn. A measurement-based joint power and rate controller for ieee 802.11 networks, 2013. URL http://d-nb.info/106738460X/34.

[82] IEEE Standard Asociation. Ieee std 802.11^{TM}-2012. 2012.

[83] Talia Odete Ledesma Quiñones, Lucy Coya Rey, and Luis Alberto Marichal Alcántara. Herramientas de monitorización y análisis del tráfico en redes de datos. 2012. URL http://revistatelematica.cujae.edu.cu/index.php/tele/article/download/62/61.

[84] Ramesh Babu H. Siddamallaiah, Gowrishankar Subramanian, and Piriyapatna S. Satyanarayana. A perspective on traffic measurement tools in wireless networks. 2010. URL http://www.scirp.org/journal/PaperDownload.aspx?paperID=2337.

[85] Acrylic. Análisis de cobertura wifi y seguridad. https://www.acrylicwifi.com/, online. Visitado en Octubre 2014.

[86] Cisco. The next generation meraki wifi. https://meraki.cisco.com/, online. Visitado en Octubre 2014.

[87] Orion. Acerca de cisco meraki. http://www.solucionesorion.com/partners/meraki, online. Visitado en Octubre 2014.

[88] Cisco Merkai. Meraki wifi stumbler. https://play.google.com/store/apps/details?id=com.meraki.wifistumbler, online. Visitado en Octubre 2014.

[89] Cisco. Try cisco meraki cloud networking for free. https://meraki.cisco.com/es/form/trial, online. Visitado en Octubre 2014.

[90] Cisco. Meraki dashboard demo. https://meraki.cisco.com/es/form/demo, online. Visitado en Octubre 2014.

[91] Merkai. Introducción a la tecnología de red en la nube. https://meraki.cisco.com/lib/pdf/meraki_overview_es.pdf, online. Visitado en Octubre 2014.

[92] Merkai. Merkai. https://meraki.cisco.com/lib/pdf/meraki_datasheet_trust_es.pdf, online. Visitado en Octubre 2014.

[93] OpenWRT. Openwrt howtos. http://wiki.openwrt.org/doc/howto/start, online. Visitado en Enero 2015.

[94] OpenWRT. Wireless utilities. http://wiki.openwrt.org/doc/howto/wireless.utilities, online. Visitado en Enero 2015.

[95] OpenWRT. Wireless overview. http://wiki.openwrt.org/doc/howto/wireless.overview, online. Visitado en Enero 2015.

[96] OpenWRT. Beginners' guide to openwrt. http://wiki.openwrt.org/doc/howto/user.beginner, online. Visitado en Enero 2015.

[97] OpenWRT. Kamikaze. http://downloads.openwrt.org/kamikaze/docs/openwrt.html, online. Visitado en Enero 2015.

[98] OpenWRT. Advanced user. http://wiki.openwrt.org/doc/howto/user.advanced, online. Visitado en Enero 2015.

[99] OpenWRT. Opkg package manager. http://wiki.openwrt.org/doc/techref/opkg, online. Visitado en Enero 2015.

[100] OpenWRT. Openwrt wireless faq. http://wiki.openwrt.org/doc/faq/faq.wireless, online. Visitado en Enero 2015.

[101] OpenWRT. Faq after installation of openwrt. http://wiki.openwrt.org/doc/faq/after.installation, online. Visitado en Enero 2015.

[102] OpenWRT. Openwrt buildroot – about. http://wiki.openwrt.org/about/toolchain, online. Visitado en Enero 2015.

[103] OpenWRT. Routed ap. http://wiki.openwrt.org/doc/recipes/routedap, online. Visitado en Enero 2015.

[104] OpenWRT. Linux network interfaces. http://wiki.openwrt.org/doc/networking/network.interfaces, online. Visitado en Enero 2015.

[105] OpenWRT. Bandwidthd. http://wiki.openwrt.org/doc/howto/bandwidthd, online. Visitado en Enero 2015.

[106] OpenWRT. Bandwidth monitoring. http://wiki.openwrt.org/doc/howto/bwmon, online. Visitado en Enero 2015.

[107] OpenWRT. Kismet. http://wiki.openwrt.org/doc/howto/wireless.tool.kismet, online. Visitado en Enero 2015.

[108] OpenWRT. Configure a guest wlan. http://wiki.openwrt.org/doc/recipes/guest-wlan, online. Visitado en Enero 2015.

[109] OpenWRT. Openwrt sysupgrade. http://wiki.openwrt.org/doc/howto/generic.sysupgrade, online. Visitado en Enero 2015.

[110] OpenWRT. Command-line interpreter. http://wiki.openwrt.org/doc/howto/user.beginner.cli, online. Visitado en Enero 2015.

[111] OpenWRT. Wireless configuration. http://wiki.openwrt.org/doc/uci/wireless, online. Visitado en Enero 2015.

[112] OpenWRT. Cross compile. http://wiki.openwrt.org/doc/devel/crosscompile, online. Visitado en Enero 2015.

[113] Javier Rodriguez. Portando openwrt a un mikrotik routerboard rb133c. http://www.javirodriguez.com.es/2011/06/08/portando-openwrt-a-un-mikrotik-routerboard-rb133c/, online. Visitado en Octubre 2014.

[114] Rb1xx goes openwrt. http://rb1xx.ozo.com/doku.php, online. Visitado en Octubre 2014.

[115] Lachlanmiskin. Using minicom yo interface with serial devices on linux. http://lachlanmiskin.com/blog/2012/08/03/using-minicom-to-interface-with-serial-devices-on-linux/, online. Visitado en Octubre 2014.

[116] nixCraft. Linux / unix minicom serial communication program. http://www.cyberciti.biz/tips/connect-soekris-single-board-computer-using-minicom.html, online. Visitado en Octubre 2014.

[117] Kuyné. Instalar y configurar servidor dhcp en ubuntu y derivados. http://kuyne.blogspot.com/2013/09/instalar-y-configurar-servidor-dhcp-en.html, online. Visitado en Octubre 2014.

[118] Linux Hispano. Instalar y configurar servidor tftp en ubuntu. http://www.linuxhispano.net/2012/02/09/instalar-y-configurar-servidor-tftp-en-ubuntu/, online. Visitado en Octubre 2014.

[119] kernel.org. Replacing iwconfig with iw. https://wireless.wiki.kernel.org/en/users/Documentation/iw/replace-iwconfig, online. Visitado en Enero 2015.

[120] kernel.org. Iw. https://wireless.wiki.kernel.org/en/users/Documentation/iw, online. Visitado en Enero 2015.

[121] kernel.org. Linux wireless. https://wireless.wiki.kernel.org/en/users, online. Visitado en Enero 2015.

[122] man.cx. Iwconfig manpage. http://man.cx/iwconfig, online. Visitado en Enero 2015.

[123] dragorn. Kismet. https://www.kismetwireless.net/, online. Visitado en Enero 2015.

[124] Kismet. Kismet.git. https://www.kismetwireless.net/kismet.git/, online. Visitado en Enero 2015.

[125] Mike Kershaw. Kismet github. https://github.com/ahendrix/kismet, online. Visitado en Enero 2015.

[126] Joe Barr. An introduction to the kismet packet sniffer. 2008. URL http://archive09.linux.com/feature/139754.

[127] Johannes Berg. iw linux github. https://github.com/dickychiang/iw-linux, online. Visitado en Enero 2015.

[128] HP. Wireless tools for linux. http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html#latest, online. Visitado en Enero 2015.

[129] Vladimir A. Pshenkin. Bandwidth monitoring tools for linux. http://dynacont.net/documentation/linux/network_monitoring/, online. Visitado en Enero 2015.

[130] Mike Kershaw. Kismet readme. 2011. URL https://www.kismetwireless.net/documentation.shtml.

[131] Componentality Oy. Step by step guide for starting hello, world! on openwrt. 2013. URL https://www.componentality.com/res/Step-By-Step-Instruction-To-Run-Apps-On-FlexRoad-HW.en.pdf.

[132] Mike Kershaw. Linux and 802.11 wireless. 2004. URL https://www.kismetwireless.net/presentations/mhvlug-wireless.pdf.

[133] Matthew S. Gast. *802.11ac: A Survival Guide*. O'Reilly Media, 2013.

[134] Luis Martín García. Programing with libpcap - sniffing the network. 2008. URL http://recursos.aldabaknocking.com/libpcapHakin9LuisMartinGarcia.pdf.

[135] Alberto Dainotti y Antonio Pescapé. Plab: a packet capture and analysis architecture. 2004. URL http://traffic.comics.unina.it/software/ITG/D-ITGpublications/TR-DIS-122004.pdf.

[136] Tcpdump. Tcpdump and libpcap. http://www.tcpdump.org/#latest-release, online. Visitado en Abril 2015.

[137] Tim Carstens. Programming with pcap. http://www.tcpdump.org/pcap.html, online. Visitado en Abril 2015.

[138] Tcpdump. Tcpdump and libpcap. http://www.tcpdump.org/#latest-release, online. Visitado en Abril 2015.

[139] Jonathan Huang. How to install libpcap on latest ubuntu. http://xgjonathan.blogspot.com/2011/04/how-to-install-libpcap-on-ubuntu.html, online. Visitado en Abril 2015.

[140] Alejandro López Monge. Aprendiendo a programar con libpcap. 2005. URL http://www.e-ghost.deusto.es/docs/2005/conferencias/pcap.pdf.

[141] tcpdump. pcap loop. http://www.tcpdump.org/manpages/pcap_loop.3pcap.html, online. Visitado en Abril 2015.

[142] cet.nau. Packet capture with libpcap and other low level network tricks. http://eecs.wsu.edu/~sshaikot/docs/lbpcap/libpcap-tutorial.pdf, online. Visitado en Abril 2015.

[143] University Of California Riverside. Tcpdump filters. http://www.cs.ucr.edu/~marios/ethereal-tcpdump.pdf, online. Visitado en Abril 2015.

[144] tcpdump.org. pcap(3) - linux man page. http://linux.die.net/man/3/pcap, online. Visitado en Abril 2015.

[145] GuyHarris. Libpcap file format. https://wiki.wireshark.org/Development/LibpcapFileFormat, online. Visitado en Abril 2015.

[146] Hani Benhabiles. A look at the pcap file format. http://www.kroosec.com/2012/10/a-look-at-pcap-file-format.html, online. Visitado en Abril 2015.

[147] WinPcap. Unix-compatible functions. https://www.winpcap.org/docs/docs_40_2/html/group__wpcapfunc.html, online. Visitado en Abril 2015.

[148] manoftoday. Writing and compiling a simple program for openwrt. https://manoftoday.wordpress.com/2007/10/11/writing-and-compiling-a-simple-program-for-openwrt/, online. Visitado en Abril 2015.

[149] OpenWRT. Openwrt buildroot – installation. http://wiki.openwrt.org/doc/howto/buildroot.exigence, online. Visitado en Abril 2015.

[150] OpenWRT. Using the sdk. http://wiki.openwrt.org/doc/howto/obtain.firmware.sdk, online. Visitado en Abril 2015.

[151] nao15irikiin. Cross compiling for openwrt. https://github.com/airplug/airplug/wiki/Cross-compiling-for-OpenWRT, online. Visitado en Abril 2015.

[152] Advanxer. Openwrt: Monitoring using collectd. https://advanxer.com/blog/2013/02/openwrt-monitoring-using-collectd/, online. Visitado en Abril 2015.

[153] OpenWRT. Using dependencies. http://wiki.openwrt.org/doc/devel/dependencies, online. Visitado en Abril 2015.

[154] yorkspace.com. Using public keys with dropbear ssh client. https://yorkspace.wordpress.com/2009/04/08/using-public-keys-with-dropbear-ssh-client/, online. Visitado en Abril 2015.

[155] jkini. How to scp, ssh and rsync without prompting for password. https://blogs.oracle.com/jkini/entry/how_to_scp_scp_and, online. Visitado en Abril 2015.

[156] yolinux. C/c++ signal handling. http://www.yolinux.com/TUTORIALS/C++Signals.html, online. Visitado en Abril 2015.

[157] Alexander Sandler. Signal handling in linux. http://www.alexonlinux.com/signal-handling-in-linux, online. Visitado en Abril 2015.

[158] tutorialspoint. Postgresql - c/c++ interface. http://www.tutorialspoint.com/postgresql/postgresql_c_cpp.htm, online. Visitado en Abril 2015.

[159] Sean Hamlin. Installing and configuring postgresql 9.1 on ubuntu 12.04 for local drupal development. http://www.pixelite.co.nz/article/installing-and-configuring-postgresql-91-ubuntu-1204-local-drupal-development/, online. Visitado en Abril 2015.

[160] PostgreSQL. Postgresql 9.1.15 documentation. http://www.postgresql.org/docs/9.1/static/, online. Visitado en Abril 2015.

[161] Linux Wireless. Automatic channel selection. https://wireless.wiki.kernel.org/en/users/documentation/acs, online. Visitado en Abril 2015.

[162] INC WILDPACKETS. 802.11 wlan packet types. http://www.wildpackets.com/resources/compendium/wireless_lan/wlan_packet_types, online. Visitado en Abril 2015.

[163] Nicolas Darchis. 802.11 frames : A starter guide to learn wireless sniffer traces. https://supportforums.cisco.com/document/52391/80211-frames-starter-guide-learn-wireless-sniffer-traces, online. Visitado en Abril 2015.

[164] Shyam Parekh. Ieee 802.11 wireless lans. http://inst.eecs.berkeley.edu/~ee122/sp07/80211.pdf, online. Visitado en Abril 2015.