Control de parámetros de transmisión en Redes WiFi 802.11

Artave, Jorge
Irland, Matías

Instituto de Computación

Facultad de Ingeniería

Universidad de la República

Tesis presentada en cumplimiento de los requerimientos para la obtención del título de

Ingeniero en Computación

TUTOR TESIS:
Dr. Javier Baliosian, Universidad de la República

COTUTOR TESIS:
Mtr. Matías Richart, Universidad de la República

Montevideo, Marzo 2017

Resumen

En los últimos años las redes *WiFi 802.11* se han convertido en una de las más utilizadas debido a su practicidad, bajo costo y alto rendimiento. Estas características propician escenarios en donde es posible encontrar en un espacio reducido, una gran cantidad de terminales y *access points* (*APs*), dando lugar a un ambiente conocido como de alta densidad. En condiciones normales, las redes *WiFi* suelen tener un buen rendimiento, pero éste se ve notoriamente deteriorado en escenarios de alta densidad, pudiendo agravarse aún más esta condición en los casos en los cuales el canal se encuentre en malas condiciones (por ejemplo, en casos en los que se encuentre un alto nivel de ruido, o interferencias generadas por aparatos cercanos como pueden ser otras redes *WiFi*, aparatos *bluetooth*, microondas, etc.). En particular estas interferencias son notorias cuando distintos *APs* transmiten en un mismo canal de forma simultánea. Es por esto que en trabajos anteriores se realiza un análisis de los posibles problemas en las redes *WiFi* de alta densidad, los distintos algoritmos que buscan mejorar el rendimiento en dichos escenarios y se plantean dos alternativas a los algoritmos ya existentes, *Robust Rate Adaptation Algorithm (RRPAA)* y *Power, Rate and Carrier Sense Threshold Control (PRCS)* . Dichas alternativas, en los distintos escenarios simulados, prueban mejorar el rendimiento de las redes *WiFi* de alta densidad, inclusive *PRCS* logra duplicar teóricamente el rendimiento global de la red en escenarios en donde otros mecanismos fallan.

RRPAA se basa en la manipulación del control del *power* y *rate*, mientras que *PRCS* agrega el control de *carrier sense* en los *APs* de la red con el fin de mitigar los posibles problemas de terminales ocultos y terminales expuestas. Partiendo de los trabajos anteriores, en el presente proyecto se plantea una posible implementación del algoritmo *RRPAA* directamente sobre el *driver ath9k* de la familia de dispositivos *Atheros* y el *framework mac80211* para redes *WiFi* en la frecuencia de 2.4GHz. Debido a que la implementación del control del *carrier sense* es específico de cada *hardware*, se desarrolla un prototipo de RRPAA, quedando la

posibilidad de incluir esta funcionalidad en una futura etapa obteniendo de esta forma una implementación completa de *PRCS*. En el presente trabajo, se presentan pruebas realizadas, las cuales muestran las ventajas de usar este protocolo en ciertos escenarios, en comparación a los protocolos originales.

Índice general

Índice de figuras

Glosario

Acces Point (AP) Dispositivo que permite conectar nodos inalámbricos a la red.

Ad-hoc Tipo de red WiFi en donde la comunicación se realiza directamente entre los nodos.

Attempts Cantidad de intentos de transmisión de una trama.

Canal Espectro de frecuencia por la cual son enviadas las tramas.

Carrier Sense Multiple Access With Collision Avoidance (CSMA/CA) Protocolo de acceso al medio.

Carrier Sense Threshold (CST) Valor que determina una cota inferior a la intensidad de que es posible tomar como válida. Todas las señales con una intensidad menor a esta, son consideradas cómo ruido.

Clear To Send (CTS) Mensaje del mecanismo de CTS/RTS el cual indica que el terminal puede comenzar a transmitir su trama.

Contention Window (CW) Valor que define el rango posible de tiempos de espera al intentar enviar una trama, utilizado por el mecanismo de acceso al medio.

CTS/RTS Mecanismo introducido en el estándar 802.11 con el fin de disminuir las colisiones de tramas asociadas al problema de la terminal oculta.

Data rate o rate Velocidad a la cual es posible introducir datos al ambiente.

Frame Loss-Rate (FLR) Porcentaje de pérdida de tramas.

GNU General Public Licence Licencia gratuita que permite la distribución de copias y versiones modificadas de la obra sobre la cual es aplicada.

IEEE 802.11 Estándar para redes de tipo WLAN.

Internet Of Things (IoT) Practica consistente en la interconexión de objetos cotidianos.

Maximum Tolerable Loss (MTL) Umbral definido por RRAA+ basic involucrado en el decremento del *rate*.

Multi Rate Retry (MRR) Mecanismo del protocolo *IEEE 802.11* que permite establecer distintos valores de data *rate* para distintos intentos de envío de las tramas.

Opportunistic Rate Increase (ORI) Umbral definido por RRAA+ basic involucrado en el incremento del *rate*.

PBUSY Probabilidad de encontrarse ocupado el ambiente al momento de comenzar una transmisión.

Physical Layer Convergence Protocol (PLCP) Capa de *WiFi* con rol de intermediario entre la capa *MAC* del *IP stack* y la *Physical Medium Dependent*.

Physical Medium Dependent (PMD) Esta capa es encargada del manejo de los distintos canales de frecuencia y de los esquemas de modulación disponibles.

Power Cantidad de energía consumida por unidad de tiempo.

Power, Rate and Carrier Sense Threshold Control (PRCS) Algritmo de control de *rate*, *power* y umbral de carrier sense.

Probabilistic Rate Increase Mecanismo utilizado para asegurar la convergencia de RRPAA, el mismo consiste en mantener una probabilidad de ingresar al próximo estado.

Probabilistic Rate Increase (PRI) Valor estadístico mantenido por RRPAA utilizado en el incremento del *rate*.

Rate Adaptation Algorithm Mecanismo responsable de determinar el valor de *rate* inicial al enviar una trama.

Received Signal Strength (RSS) Valor de *power* percibido por el receptor.

Redes de alta densidad Redes de computadoras con una gran cantidad de nodos conectados en un espacio reducido.

Request To Send (RTS) Mensaje del mecanismo de CTS/RTS el solicita al router la posibilidad de enviar una trama.

Retry chain Estructura de datos que almacena los parámetros a utilizar en cada intento de transmisión de una trama.

Robust Rate Adaptation Algorithm (RRPAA) Algritmo de control de *rate* y *power*.

Signal Detection Threshold (SDT) Valor mínimo de señal que debe ser recibido para diferenciar una trama de ruido.

Signal to Noise Ratio (SNR) Relación entre la intensidad de la señal y del ruido.

Starvation Problemática generada por los efectos de terminal oculta y terminal expuesta.

Success Tramas enviadas exitosamente.

Symbol Una transmisión se compone de varios symbols éstos a su vez puede contener varios bits.

Terminal expuesta Efecto que es posible encontrar en redes inalámbricas en donde una termina se abstiene de transmitir pudiendo haberlo hecho de forma exitosa.

Terminal oculta Efecto que es posible encontrar en redes inalámbricas en donde dos o mas transmisores no pueden detectarse y por esto se producen colisiones en el receptor.

Throughput Medida de la cantidad de datos transmitidos en un período de tiempo.

Trama Paquete de datos generado en la capa de enlace del *IP stack*.

Transmission Opportunity Fracción de tiempo en la cual el ambiente se encuentra disponible para transmitir.

WiFi Nombre con el que es normalmente conocido el estándar *IEEE 802.11*.

Wireless Local Area Network (WLAN) Sistema inalámbrico de interconexión entre dispositivos.

Capítulo 1

Introducción

En los últimos años ha habido un gran aumento de demanda en las redes inalámbricas, en particular las redes *WiFi 802.11*, las cuales son una de las más utilizadas debido a su practicidad, bajo costo y alto rendimiento. Desde sus inicios muchas investigaciones han surgido, principalmente para mejorar la cobertura y el aumento del *throughput* posible, intentando de esta forma convertir estas redes tan robustas y rápidas posibles como las redes *ethernet* con el beneficio de ser inalámbricas. Estas redes suelen funcionar correctamente en ambientes "controlados", pero su rendimiento es notoriamente deteriorado si el medio de transmisión se encuentra en malas condiciones, pudiendo encontrarse congestionado o con altos niveles de ruido. En particular estas interferencias son notorias cuando distintos AP transmiten en canales cercanos de forma simultánea. Este trabajo se basa principalmente en la tesis de maestría de Matías Richart[1], en donde se realiza un análisis exhaustivo del problema que sufren las redes *WiFi* en ambientes de alta densidad, esto es, espacios reducidos en donde es posible encontrar varios terminales y AP compitiendo por transmitir sus tramas. En dicho trabajo se propone un algoritmo para mitigar estos problemas, se realizan simulaciones y se comparan con algoritmos existentes. De estas simulaciones es posible observar que el algoritmo planteado obtiene mejores resultados que los anteriores, llegando a duplicar el rendimiento de la red en determinados escenarios. Dicho algoritmo es desarrollado en dos etapas, en la primer etapa se realizan modificaciones a un algoritmo existente, agregando a éste la característica de poder variar el *power* dependiendo de ciertos estados del canal y un mecanismo para asegurar su convergencia. La segunda etapa consiste en incluir, al resultado anterior, control de *carrier sense*. Por motivos de practicidad, en el presente trabajo se referencia a la versión del algoritmo con *power* control, como $RRPAA$ y a la versión final como *PRCS*.

Teniendo en cuenta que las simulaciones son aproximaciones a la realidad y que

éstas a su vez incluyen simplificaciones, resulta importante para evaluar el comportamiento de un algoritmo de este tipo, analizar su comportamiento en escenarios reales. Por lo mencionado anteriormente, en el presente proyecto se lleva a cabo una implementación de *RRPAA* y se evalúa su comportamiento en *hardware*, comparando sus resultado con el algoritmo de *rate control* incluido originalmente en el *driver ath9k*.

Para esto se realiza un conjunto de pruebas, las cuales consideran distintos escenarios en los cuales se evalúa el rendimiento de la implementación, incluyendo un escenario para el cual fue desarrollado, un escenario para los cuales no lo fue y un escenario neutro.

1.1. Objetivos

En el transcurso del presente proyecto se busca obtener una implementación funcional de *RRPAA*, la cual permita en un futuro extender su desarrollo con el fin de obtener una implementación funcional de *PRCS*. Junto a dicha implementación, teniendo en cuenta la escasa documentación existente referente al desarrollo de este tipo de *software*, se busca brindar un documento que pueda ser un punto de partida para futuros trabajos, en particular, indicando la arquitectura del sistema, las estructuras básicas del *driver*, las funciones que pueden ser modificadas, los requisitos necesarios para su compilación y ayudas posibles para futuros desarrolladores.

1.2. Contribución

En el marco del presente proyecto fue posible obtener una implementación completamente funcional del algoritmo *RRPAA*. Junto a esto, se tiene uno de los principales aportes: el de poder comparar una implementación real de un algoritmo que hasta el momento únicamente había sido simulado, contra un algoritmo estándar en los sistemas con *kernel Linux*.

Finalmente, se presenta el código producido junto a la documentación presentada en el presente informe, que puede ser considerado como un gran aporte debido a la escasa documentación existente. Dicho código se encuentra disponible de forma libre a quien desee acceder a él en *Github*[2].

1.3. Estructura

El presente documento se encuentra estructurado de la siguiente manera. El Capítulo 2 incluye conocimientos básicos de redes *WiFi*, con el fin de entender el problema

que busca resolver *RRPAA*. En el Capítulo 3 se presenta un conjunto de algoritmos que al igual que *RRPAA*, controlan parámetros del estándar *IEEE 802.11* con el fin de mejorar el rendimiento de la red. El Capítulo 4 presenta información referente al desarrollo propiamente dicho del algoritmo. El Capítulo 5 presenta información referente a las pruebas realizadas, los resultados obtenidos y un conjunto de pruebas que se consideran interesantes a realizar en un futuro proyecto. Finalmente el Capítulo 6 incluye un conjunto de conclusiones obtenidas.

Capítulo 2

Descripción del problema

2.1. Estándar *IEEE 802.11*

Gracias al bajo costo y buen desempeño que tienen este tipo de redes, es posible desplegarlas con el fin de interconectar diversos tipos de dispositivos, desde computadoras de escritorio hasta dispositivos de *IoT* como pueden ser cafeteras, calefones o diversos sensores, entre sí o a dispositivos conectados mediante internet.

Actualmente el estándar *IEEE 802.11* (normalmente conocido como *WiFi*) es el estándar de facto en las redes de tipo *WLAN*, permitiendo interconectar diversos dispositivos mediante la transmisión de ondas de radio. Dicho estándar especifica las capas de enlaces y físicas para redes *WLAN*.

Si bien existe una gran cantidad de redes *WiFi*, en [3] se detallan algunos de los inconvenientes que sufren este tipo de redes, entre ellos se encuentran:

- Interferencias debido a la cercanía de otras redes *WLAN*, ruido de ambiente o interferencia entre dispositivos de una misma red.
- Interferencias generadas por distintos *symbols* (una transmisión se compone de varios *symbols* y cada uno puede contener varios *bits*) de una misma transmisión. Esta forma de interferencia es debido que distintos *symbols* de una misma transmisión pueden recorrer distintos caminos, lo que posibilita que distintos *symbols* colisionen.
- Débil señal de las transmisiones en el receptor, debido al fenómeno denominado *path loss*, el cual provoca la pérdida de intensidad a medida que ésta atraviesa el ambiente.

Finalmente cabe mencionar que las redes de tipo *IEEE 802.11* pueden trabajar en alguno de los siguientes modos:

- Infraestructura, en donde se cuenta con un nodo central, comúnmente llamado *AP* o *Access Point*, al cual los nodos le envían las tramas y es el *AP* el encargado de enviar las tramas al destinatario.
- *Ad-hoc*, en este tipo de redes no existe un nodo central como el *AP*. En este tipo de redes, los terminales se comunican de forma directa.

De estos tipos de redes *IEEE 802.11*, únicamente se encuentran dentro del alcance del presente proyecto las redes de tipo infraestructura, debido a que tanto *RRPAA* como *PRCS* fueron desarrollados para este tipo.

2.1.1. Capa física

La capa física de *IEEE 802.11* se encuentra, a su vez, dividida en dos capas. La primera de estas subcapas es la *Physical Medium Dependent* (*PMD*). Esta capa es encargada del manejo de los distintos canales de frecuencia y de los esquemas de modulación disponibles (los cuales inciden directamente en el *rate* obtenido). La segunda de las subcapas es la llamada *Physical Layer Convergence Protocol* (*PLCP*). Esta subcapa actúa principalmente de intermediario entre la capa *MAC* del *stack IP* y la *PMD*. Adicionalmente la capa *PMD* es responsable de la inclusión de información de cabecera y cola a las tramas, necesaria tanto para los dispositivos receptores como emisores.

Dentro de esta capa, resulta interesante tener en cuenta algunos de los esquemas de modulación manejados. Dentro de éstos se encuentran:

- *Direct Sequence Spread Spectrum* (*DSSS*)
- *High Rate Direct Sequence Spread Spectrum* (*HR/DSSS*)
- *Orthogonal Frequency Division Multiplexing* (*OFDM*)
- *Extended Rate* PHY (*ERP*)
- *High Throughput* OFDM (*HT/OFDM*) *PHY*

Los valores de *rate* obtenidos por los distintos esquemas y en qué revisión del estándar *IEEE 802.11* son utilizados, se encuentran presentados en la tabla 2.1.

En esta capa resulta interesante tener en cuenta las particularidades de los formatos de modulación utilizados. Un formato de modulación define transformaciones físicas que serán necesarias realizar al enviar una tira de *bits* en un ambiente inalámbrico, dichas transformaciones al ambiente son normalmente conocidas como *symbols* [4].

Revision	Fecha	Frecuencia	Data rate (Mb/s)	PHY-layer
802.11	Jun. 1997	2.4 GHz	1, 2	DSSS, FHSS
802.11a	Sep. 1999	5 GHz	6, 9, 12, 18, 24, 36, 48, 54	OFDM
802.11b	Sep. 1999	2.4 GHz	1, 2, 5.5, 11	DSSS
802.11g	Jun. 2003	2.4 GHz	6, 9, 12, 18, 24, 36, 48, 54	DSSS, OFDM
802.11n	Oct. 2009	2.4 y 5 GHz	hasta 600	OFDM
802.11ac	Dic. 2012	5 GHz	hasta 3460	OFDM

Cuadro 2.1: Revisiones IEEE 802.11. Fuente [1].

Resulta necesario tener en cuenta que la cantidad de *symbols* posibles en un formato tiene algunas limitaciones, en particular, a medida que la cantidad de *symbols* posibles aumenta, la decodificación de éstos se torna más costosa. En particular es posible demostrar que la diferencia (por ejemplo, de frecuencia o de amplitud, dependiendo de la propiedad afectada) mínima entre dos valores posibles, tiene una incidencia directa sobre el valor mínimo de *Signal to Noise Ratio* (*SNR*) necesario con el fin de evitar errores de *bit*. Esta información resulta interesante debido a que establece una relación entre el *rate* y el *power*, en particular plantea que a medida que se aumenta el *rate* y se mantiene fijo el valor de *power*, es posible que el *SNR* obtenido sea menor al necesario para decodificar correctamente el mensaje, lo que implica la necesidad de aumentar el *power* del emisor buscando aumentar el *SNR*. Ejemplos de los valores de *SNR* necesarios a diferentes combinaciones de esquemas y formatos de modulación son presentados en la tabla 2.2.

2.1.2. Capa de enlace

La capa de enlace, en el estándar *IEEE 802.11* se encuentra principalmente dominada por una función de coordinación. Actualmente existen cuatro funciones de coordinación:

- *Distributed Coordination Function* (*DCF*)
- *Point Coordination Function* (*PCF*)
- *Hybrid Coordination Function* (*HCF*)
- *Mesh Coordination Function* (*MCF*)

En redes de tipo infraestructura, la función de coordinación utilizada por defecto es la *DCF*. Dicha función de coordinación es un mecanismo de *carrier sense* y prevención de colisiones (*CSMA/CA*) distribuido. Esta función indica que el terminal,

Data Rate	Standard	PHY-layer	Modulation format	SNR (dBm)
1	b	DSSS	BPSK	-2.92
2	b	DSSS	QPSK	1.59
5.5	b	DSSS	CCK	5.98
11	b	DSSS	CCK	6.99
6	g	OFDM	BPSK	6.02
9	g	OFDM	BPSK	7.78
12	g	OFDM	QPSK	9.03
18	g	OFDM	QPSK	10.79
24	g	OFDM	16-QAM	17.04
36	g	OFDM	16-QAM	18.80
48	g	OFDM	64-QAM	24.05
54	g	OFDM	64-QAM	24.56

Cuadro 2.2: IEEE 802.11b/g Parámetros de codificación y SNR mínimo. Fuente [1].

antes de enviar alguna trama, debe sensar el ambiente y de encontrarse el ambiente libre por un determinado periodo de tiempo, se procede con la transmisión. En caso de encontrar el canal ocupado, es necesario esperar a que dicha transmisión culmine. Luego de esto, antes de volver a intentar se espera un tiempo seleccionado de forma al azar (normalmente conocido como *back-off time* y seleccionado de un rango preestablecido y variable, acotado por una variable conocida como *Contention Window*).

Existe un modo alternativo al anteriormente mencionado, llamado *virtual carrier sense*. Esta alternativa consiste en el intercambio de dos tramas de escaso tamaño, llamadas *Request To Send* (*RTS*) y *Clear To Send* (*CTS*), con el fin de reservar el espacio por el tiempo necesario para enviar una trama. En un principio el nodo a transmitir envía una trama *RTS* y el destinatario responde con una *CTS*. De esta forma los restantes nodos, al recibir estas tramas, son conscientes de la reserva del ambiente. Este mecanismo tiene como cometido mitigar los problemas de terminal oculta y terminal expuestas que se explican en la sección 2.5.

Finalmente, para cada trama exitosamente transmitida, el receptor, en caso de recibirla correctamente, debe responder con una trama *ACK*. En caso de no recibir una trama de este tipo, el emisor asume que hubo algún tipo de error por lo que una retransmisión es necesaria.

2.2. Redes *WLAN* de alta densidad

Se puede decir que una red *WLAN* es de de alta densidad, cuando tiene un elevado número de nodos conectados a la misma en un espacio reducido, compitiendo por los recursos de la red. Este tipo de redes se caracterizan por la gran problemática que sufren en términos de performance, cuando el número de nodos es muy elevado o cuando éstos no se organizan de forma adecuada. En general, se puede decir que una red *WLAN* intenta optimizar dos métricas de importancia en las redes, el *throughput* y el *delay*. En cuanto al *throughput* es posible afirmar que éste se encuentra directamente relacionado a las condiciones del ambiente:

- Cantidad de tiempo que el nodo tiene acceso al medio, debido a que un tiempo mayor brinda más posibilidades de transmisión.
- El número de nodos que están vinculados al *AP*, debido a que el *AP* tiene que distribuir la carga entre todos los nodos
- La capacidad del enlace entre el nodo y el *AP*, siendo de suma importancia el data *rate*.

El estándar *IEEE 802.11* define un conjunto reducido de canales, que pueden ser utilizados por los nodos de la red. Sin embargo este tipo de redes cuentan con una gran cantidad de nodos transmitiendo en dichos canales, redes cercanas, por lo que no es posible aislar las comunicaciones, influyendo en el rendimiento de estas redes. Para mitigar estos problemas, es posible definir mecanismos tanto en el emisor como en el receptor. En cuanto al receptor, es posible definir mecanismos que pueden afectar la sensibilidad de éste, mediante la modificación de los parámetros del mecanismo de *carrier sense*. En cuanto al emisor, es posible trabajar con el *power* para aumentarlo o disminuirlo dependiendo el estado de la red.

Finalmente es importante destacar los estudios realizados en [5], los que indican que el efecto de incluir un nuevo *AP* a la red, es significativamente mayor que agregar un nuevo terminal.

2.2.1. Manipulación de parámetros

Una vez instalada las redes de tipo *IEEE 802.11*, es posible realizar ajustes al funcionamiento normal de la misma mediante la modificación de algunos parámetros de su funcionamiento. Entre los más utilizados en la literatura es posible encontrar:

- *Canal*

- *Power*
- *Data Rate*
- *CCA Sensitivity* (*Carrier Sense Threshold*, *CST*)
- *Receiver Sensitivity* (*Signal Detection Threshold*, *SDT*)
- *Contention Window* (*CW*)

Ajustando los valores de *CW* y *CST*, junto a *CSMA/CA*, es posible aislar las transmisiones en el tiempo. Modificar el canal utilizado permite aislar las transmisiones en el espacio, modificando la frecuencia utilizada. Por último, modificar el *data rate* establece los mecanismos de modulación y codificación que afectan los mecanismos de corrección de error y el mínimo *SNR* necesario para una correcta recepción de la trama.

Al modificar estas variables es necesario tomar precauciones, debido a que estos cambios pueden implicar efectos negativos. Como ejemplo de este comportamiento, modificar el *power* (uno de los parámetros más estudiados) permite disminuir la interferencia generada, en particular a los canales contiguos, logrando una mejora general en la red. Sin embargo, este cambio en caso de disminuir el valor utilizado, puede llevar a un deterioro de la *performance* del nodo, debido a un posible aumento en las pérdidas de tramas (por ejemplo debido a un menor *SNR* en el receptor).

Otro ejemplo posible es el de modificar el *rate* utilizado, el cual afecta directamente el *throughput* del enlace. Pero este cambio además de afectar al nodo modificado, también afecta a los cercanos, debido a que valores de rate más bajos necesitan más tiempo para transmitir, quitándole tiempo a los demás nodos de la red. Análogamente, valores de rates más altos implican menos tiempo de transmisión, pero las pérdidas del nodo pueden aumentar debido a que el *SNR* en el receptor puede no ser suficiente para el nuevo valor de *rate*.

Además de las concesiones anteriormente mencionadas, es necesario tener en cuenta que la manipulación de estos parámetros puede agravar algunos fenómenos como el de la terminal oculta y expuesta (explicados en la sección 2.5). En particular éstos son potenciados ante la presencia de enlaces heterogéneos en las redes *WLAN*.

Finalmente, resulta interesante poder determinar cuándo la densidad de una red se torna problemática y afecta el rendimiento de la misma. En particular, en este tipo de redes es común seguir alguno de los siguientes enfoques con el fin de detectar problemas de rendimiento y poder tomar acciones:

- Pérdidas de paquetes

- *Received Signal Strength* (*RSS*)

En el caso de seguir un enfoque basado en la pérdidas de paquetes, cabe recordar que luego de recibir una trama, el receptor debe responder con una trama de tipo *ACK*. En cambio, la no recepción de una de estas tramas, se toma como un indicio de una colisión con otras tramas, bajo *SNR* en el receptor, o la recepción defectuosa de la trama (debido a bits corruptos)

En caso de seguir un enfoque basado en un bajo *RSS* percibido en el receptor, se asume que esto es debido a la interferencia generado con otros nodos. Cabe mencionar que en este enfoque, la situación es detectada por el receptor y éste debe notificar de alguna forma al emisor.

2.3. *Multi-rate retry*

Multi-rate retry es un mecanismo introducido originalmente en el proyecto madwifi[6] y que actualmente se encuentra incluido dentro del *framework mac80211*[7]. Este mecanismo es responsable de seleccionar el valor de data rate a utilizar para los distintos reintentos de transmisión de tramas que puedan ser necesarios[8].

Es posible dividir el control de rate en dos mecanismos relacionados:

1. *Rate Adaptation Algorithm*, el cual es responsable de determinar el valor con el que se intentará enviar inicialmente las tramas.

2. *Multi-rate retry*, el cual es responsable de determinar los valores de rate a utilizar en las distintas retransmisiones que puedan surgir.

De esta forma el control de rate se divide en dos mecanismos, el primero encargado de la toma de decisiones a largo plazo, por esto es posible que deba mantener un registro de información estadística. El segundo mecanismo es el encargado de la toma de decisiones ante diversas variaciones que pueda sufrir el ambiente.

Para el funcionamiento de estos mecanismos es necesario mantener una estructura de datos con la información presente en la figura 2.1.

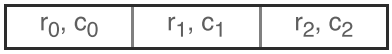

Figura 2.1: Diagrama de la retry chain.

Dicha estructura es conocida como ***multi retry chain*** o ***retry chain***. Inicialmente la trama es enviada utilizando el *rate* r_0 hasta un máximo de c_0 intentos. En caso que los c_0 intentos resulten fallidos, los siguientes son realizados con un *rate* r_1

un máximo de c_1 veces, así sucesivamente hasta consumir los registros almacenados en la estructura. En caso que todos los intentos definidos en la *retry chain* resulten fallidos, la trama es descartada y es tomada como un envío fallido. De esta forma es posible, definir futuras acciones para las tramas que no pudieron enviarse de forma exitosa, de forma que ésta pueda ser enviada exitosamente y se disminuya el *overhead* necesario para realizar ajustes en la transmisión (principalmente debido a que no es necesario ejecutar módulos del *kernel* buscando modificar los parámetros de la transmisión).

2.4. Algoritmos de *Power and Rate control*

IEEE 802.11 en su estándar incluye diversos mecanismos de modulación y codificación, los cuales brindarán distintos valores de *rate*. Estos valores de *rate* afectan directactamente el *throughput* de la red, es decir, cuanto mayor es el valor de *rate*, mayor es el *throughput* teórico al cual es posible enviar. Sin embargo, cuanto más altos son estos valores, mayor es la probabilidad que las transmisiones tengan *bits* corruptos. En contraposición, valores bajos de *rate*, disminuyen el máximo *throughput* posible, aunque aumentan la probabilidad de una recepción exitosa. De forma similar ocurre con el *power*, cuanto mayor es el *power* que se envía una trama, mayor es la probabilidad que esta alcance al terminal. Sin embargo, cuanto mayor es el *power*, mayor es la probabilidad de que la trama interfiera con una trama de otra terminal cercana, debido al aumento del alcance de la transmisión, pudiendo interferir en la transmisiones de más terminales. En cambio si el *power* es muy bajo, la señal percibida por el destinatario puede ser muy débil, por lo tanto, puede suceder que no sea posible recibir completamente la transmisión o diferenciarla del ruido del ambiente.

Por otro lado, como se ve en la sección 2.2.1, si se varía el *power* o el rate de una forma incorrecta, se pueden generar problemas de terminales ocultas o terminales expuestas.

2.5. Problemas de la Terminal Oculta y Terminal Expuesta

En esta sección se describen dos de los grandes problemas que pueden ocurrir en dispositivos en redes *WiFi*, los cuales degradan significativamente el rendimiento de la red. Como se muestra en [9], dichos problemas, introducen múltiples pérdidas y colisiones en la red.

2.5.1. Terminal oculta

Este problema ocurre cuando un transmisor no puede detectar la señal de otro transmisor, pero ambos se encuentran en el rango del receptor.

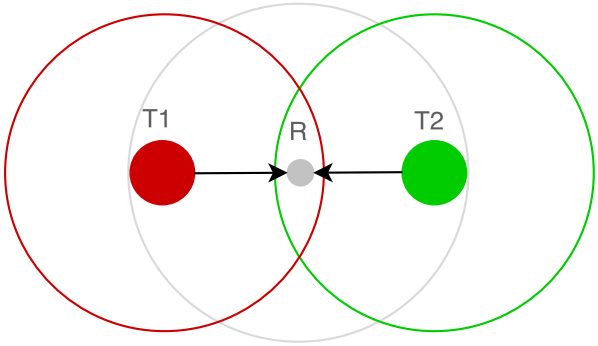

Figura 2.2: Diagrama de un escenario donde ocurre el problema de la terminal oculta.

La figura 2.2 ilustra gráficamente este escenario, en el cual los nodos T1 y T2 transmiten tramas a R. En dicho escenario, ambos nodos no se detectan entre sí, esto significa que el rango de *carrier sense* de cada nodo no incluye al otro. Por lo mencionado anteriormente, al momento de enviar las tramas, los nodos encuentran el canal en un estado libre, es por esto que ambos proceden a enviar las tramas. Sin embargo, al receptor llegan ambas tramas de forma simultánea, esto produce una colisión en el receptor, por lo que ambas tramas deben ser descartadas.

2.5.2. Terminal expuesta

Este escenario se da cuando existe un terminal el cual detecta que el estado del ambiente mayormente se encuentra ocupado. Debido a esto el mismo no logra enviar información a la red, sin embargo dicha transmisión podría haberse realizado satisfactoriamente. Dicho problema puede suceder en redes densas, debido a que existen muchos nodos en la red compitiendo para enviar información.

La figura 2.3 detalla gráficamente un posible escenario en donde sucede este problema. En dicho escenario T1 se encuentra en el rango de *carrier sense* de T2, gráficamente esto es posible observar por la línea punteada de color gris. De forma análoga, la líneas de color azul, verde y rojo indican el rango de alcance de sus respectivos centros.

Inicialmente el nodo T1 se encuentra transmitiendo a R1. En un momento dado, el nodo T2 intenta enviar tramas a R2, éste sensa el ambiente y detecta que el medio está ocupado, por lo que T2 se abstiene a enviar alguna trama. Sin embargo, R2 se encuentra fuera del alcance de T1, por lo que la transmisión de T2 a R2 podría

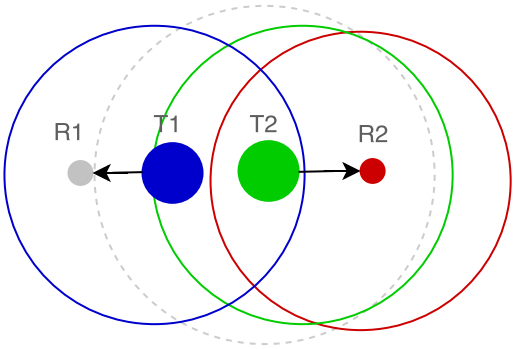

Figura 2.3: Diagrama de un escenario donde ocurre el problema de la terminal oculta.

haberse generado exitosamente.

2.6. *Starvation*

Relacionado a los problemas vistos, en [10] se define el concepto de *starvation*, el cual consiste en la problemática generada cuando un nodo puede transmitir pero no lo hace por alguna de las siguientes razones:

- Existe un nodo que sufre el problema de la terminal expuesta.
- Cuando un nodo sufre el problema de la terminal oculta.
- Cuando existe un medio donde se encuentra con asimetrías en los enlaces de los nodos transmisores o receptores, como puede ser por ejemplo el *powers* o los rangos de *carrier sense*.

Capítulo 3

Estado del arte

3.1. Algoritmos

En los últimos años se han realizado grandes avances en cuanto a investigaciones referentes a redes *WiFi* debido a su creciente demanda, enfocándose una gran parte del mismo en el área de control de parámetros de este tipo de redes. En dichos trabajos se plantea modificar, entre otros, los valores de *rate*, *power* y *carrier sense*, con el fin de mitigar los problemas que sufren este tipo de redes mencionados en el Capitulo 2. Existen a su vez propuestas que realizan modificaciones a la implementación del protocolo *IEEE 802.11*, que a pesar de conseguir buenos resultados, como se menciona en [11], tienen como desventaja tener que distribuir los cambios en todos los nodos de la red. En cuanto al tipo de red, existe una gran cantidad de trabajo realizado sobre redes de tipo *ad-hoc*, pero estas propuestas cuentan con el inconveniente de basarse en la comunicación nodo a nodo de este tipo de redes. Los algoritmos que se detallan en esta sección, se basan en ajustar los parámetros de capa física y de enlace del protocolo *IEEE 802.11* en redes de tipo infraestructura. Las diferentes propuestas consideradas se pueden clasificar según: los parámetros modificados, los mecanismos utilizados para estimar la calidad del canal, degradación de performance, el problema que intentan solucionar o si el control es realizado a nivel de enlace o de *AP*. En la tabla 3.1 se presenta un conjunto de algoritmos, basada en los analizados en [1], los cuales poseen como principal característica que cumplen el estándar de *WiFi*, aprovechando alguna de las siguientes características:

- *Power*, potencia de transmisión.
- *Rate*, tasa de transferencia de datos.
- *Carrier sense*, mecanismo de sensado de portadora.

- *Signal*, intensidad de la señal recibida, utilizado para estimar las condiciones del canal.
- *Frame Loss*, pérdida de paquetes, utilizado para estimar las condiciones del canal.
- *Per-link*, en caso que las modificaciones son realizadas por cada *link* con los *AP*.
- *Distributed*, en caso que las decisiones sean tomadas de forma distribuida.

Mechanism	Power	Rate	CST	Signal	Frame Loss	Per-link	Distributed
PARF	✓	✓			✓	✓	✓
APARF	✓	✓			✓	✓	✓
PASA	✓				✓	✓	✓
ConTPC	✓				✓	✓	✓
Symphony	✓	✓			✓	✓	✓
Minstrel		✓			✓	✓	✓
MP	✓	✓			✓	✓	✓
PMAC	✓			✓		✓	✓
DSB		✓	✓		✓		✓
Zhu			✓		✓		
ORCCA			✓	✓			

Cuadro 3.1: Algoritmos y las características aprovechadas.

3.1.1. Control de *rate* y *power*

3.1.1.1. Estimación en capa de enlace

***Auto Rate Fallback* (ARF)** utiliza la pérdida de paquetes para estimar el estado del canal, para esto, en caso de poder realizar satisfactoriamente una cierta cantidad de envíos correctos, aumenta la tasa de transferencia. En caso que se verifique una determinada cantidad de pérdidas, el algoritmo procede a disminuir la tasa de transferencia. En su versión original también utiliza un *timer*, en caso de que el resultado de dos transmisiones sea fallida, se disminuye el *rate* y se vuelve a iniciar el *timer*. Si sucede que el tiempo es consumido o se realizan diez envíos satisfactorios, el *rate* es nuevamente aumentado a no ser que la siguiente transmisión sea fallida. En dicho caso el *rate* es nuevamente disminuido y el *timer* inicializado nuevamente.

***Power-controlled Auto Rate Fallback* (PARF)** es una técnica basada en *ARF* para el control de *power* y *rate*. Esta técnica busca minimizar la interferencia

entre *APs* vecinos agregando control de potencia a *ARF*, disminuyendo la potencia en caso que el algoritmo se encuentre transmitiendo a máximo *rate* y no se registren pérdidas. Dicha reducción de potencia es realizada hasta alcanzar cierto umbral o hasta que se comiencen a registrar pérdidas, en cuyo caso se aumentará la potencia hasta que las pérdidas cesen o se alcance un valor máximo. En caso de encontrarse transmitiendo a máxima potencia y las pérdidas se continúen registrando, el algoritmo pasará a utilizar una estrategia de *fallback* sobre la tasa de transferencia.

***Adaptative PARF* (APARF)** se basa en *PARF* y plantea dos modos de ejecución: *High Performance*, el cual posee el mismo funcionamiento de *PARF* y *Low Power*, en donde la transmisión es realizada con la menor potencia posible y luego es optimizado el *rate*. Una diferencia interesante con respecto a *PARF*, es que el umbral utilizado para la toma de decisiones es modificado dinámicamente en base al estado del algoritmo.

***Power Adaptation for Starvation Avoidance* (PASA)** busca principalmente mitigar el problema de la terminal oculta, junto al problema de starvation que conlleva, basado en modificar la potencia de transmisión, utilizando una técnica similar a *PARF*.

***Conservative Transmit-Power Control* (ConTPC)** a diferencia de los anteriores utiliza datos estadísticos referente a la pérdida de tramas para controlar la potencia de transmisión.

Symphony es un mecanismo basado implícitamente en la pérdida de tramas. La idea básica es similar a la de mecanismos anteriores, modificar el *rate* y la potencia en cada enlace, manteniendo un resultado similar al obtenido a máxima potencia. Como principal desventaja se tiene que este mecanismo debe ser implementado en todos los nodos de la red y es necesario contar con un controlador central para sincronizar los *APs*. Este mecanismo cuenta con dos fases, *Reference* (*REF*) y Operational (*OPT*). En la fase *REF*, el rate control es realizado por los emisores en todos sus enlaces, utilizando para esto máximo *power*, tomando registros del rendimiento obtenido con el fin de utilizarlos como futura referencia para el control de potencia. En la siguiente fase se comienza a aplicar mecanismos de control de potencia y *rate* conjuntamente, en donde la potencia es disminuida siempre y cuando la performance no sea inferior a la registrada en la fase anterior.

Minstrel es un algoritmo del tipo *rate control*, el cual basa su funcionamiento en la recolección de estadísticas mediante el envío de tramas de sondeo y el mecanismo de *multi-rate retry*, presentado en la sección 2.3. Dichas tramas representan cerca de un 10 % del total de las tramas enviadas. Éstas son transferidas utilizando en primer instancia, el máximo *rate* entre un valor al azar y el que genere el máximo

throughput. El siguiente *rate* es seleccionado como el que genera el menor *throughput* y los restantes como los *rates* de mayores probabilidades. Para esto utiliza las tramas de sondeo con el fin de mantener, para cada *rate*, la probabilidad de realizar envíos exitosos, siendo a su vez, dicha probabilidad utilizada para la estimación del *throughput*.

Minstrel-Piano es un protocolo basado en *Minstrel*, el cual sigue la idea de los anteriores, utilizar el *rate* indicado por el mecanismo de *rate control*, utilizando el menor *power* posible, agregando información de los *ACK* recibidos con el fin estimar la interferencia y utiliza estos datos como insumo para el mecanismo de *power control*. Este algoritmo busca mejorar el anterior, agregando *power control* en cada trama. Para esto, las tramas son enviadas a distintas potencias y de esta manera se estima estadísticamente el comportamiento del impacto de la potencia en el *throughput*.

3.1.1.2. Estimación en capa física

Power control MAC (PMAC) es un algoritmo que busca mitigar el problema de la terminal oculta, basándose en el intercambio de tramas *RTS/CTS* y la medición de la intensidad de la señal. El funcionamiento consiste en enviar la trama *RTS* a menor *rate* y mayor *power* posible, buscando cubrir el mayor área posible. A continuación, se estima la potencia necesaria para enviar las tramas de datos basándose en la intensidad con la que fue recibida la trama *CTS*.

3.1.2. *Carrier-Sense Threshold Control*

3.1.2.1. Estimación en capa de enlace

Dynamic Spatial Backoff (DSB) es un algoritmo para controlar el *CST* y el *data rate* basado en los envíos exitosos o fallidos de las tramas de datos. El algoritmo se basa en la hipótesis que a un alto *rate* de emisión, es necesario un alto *power* en el receptor, por lo que un bajo *CST* es necesario. A grandes rasgos, el algoritmo consiste en comenzar utilizando el menor rate posible y máximo *CST*. A continuación, a medida que ocurran transmisiones satisfactorias, se aumenta el *rate*. En caso de haber pérdidas, dependiendo del estado actual, se procede a disminuir el *CST* o el *rate*.

Zhu es un mecanismo para balancear los problemas de la terminal oculta y la terminal expuesta controlando el *CST*. Este mecanismo se basa en la hipótesis que un bajo *Frame Lost Rate (FLR)* implica pocas colisiones, lo que conlleva a una cantidad baja de terminales ocultas, estableciendo una dualidad entre los problemas de terminal expuesta y oculta. Con esto, el mecanismo consiste en disminuir el *CST*

cuando el *FLR* es alto (lo que implicaría posibles terminales ocultos) e incrementando el *CST* cuando el *FLR* es pequeño.

3.1.2.2. Estimación en capa física

***Optimal-Rate CCA Adaptation* (*ORCCA*)** es un mecanismo para controlar el *CST* que utiliza un controlador central para seleccionar un *CST* global,

3.2. *RRPAA* y *PRCS*

Robust Rate Adaptation Algorithm (*RRAA+ Basic*) utiliza un enfoque estadístico para estimar las condiciones del canal en capa de enlace. Básicamente calcula el *FLR* de una ventana de tramas y adapta el *data rate* para mantener el *FLR* en un rango establecido. Este enfoque es similar al *Minstrel* pero sin utilizar tramas de sondeos. El protocolo cuenta principalmente con dos ideas, la primera es que define un umbral para decidir cuando se incrementa o decrementa el *rate*. La segunda idea, se basa en incrementar el *rate* en base a un dato estadístico (*Probabilistic Rate Increase*, PRI), lo que provoca que éste converja.

En base al algoritmo anterior, en [1] se presentan dos algoritmos. El primero de estos algoritmos es *RRPAA*, el cual consiste en agregar *power control* al funcionamiento de *RRAA+ basic*. En [1] se presentan un conjunto de simulaciones, las cuales prueban mejorar el rendimiento del algoritmo en un conjunto de escenarios. Este algoritmo tiene algunas ventajas sobre los anteriores (como por ejemplo, el de reducir el *power* utilizado manteniendo el *throughput*), a pesar de seguir sufriendo los problemas de *starvation*.

El segundo de los algoritmos a efectos del presente informe lo llamaremos *Power Rate Carrier Sense* (*PRCS*), el cual extiende el funcionamiento de *RRPAA* para incluir un control de *carrier sense*, buscando de esta forma, evitar o mitigar el problema de *starvation* que éste sufre. Por consiguiente, *PRCS* es un protocolo de tipo *power*, *rate* and *carrier sense control*, el cual implementa un control basado en mediciones de la capa de enlace.

3.2.1. *RRPAA*

3.2.1.1. Funcionamiento *RRAA+ basic*

Como fue mencionado anteriormente, *RRPAA* basa su funcionamiento en *RRAA+ basic*. *RRAA+ basic* centra su funcionamiento en el manejo del FLR, en una cierta ventana de tramas enviadas, buscando mantener su valor en un determinado rango.

Para esto, en lugar de utilizar tramas de sondeo, como lo hace por ejemplo *Minstrel*, recolecta datos estadísticos a lo largo de la sesión entre los terminales.

En su ejecución normal, *RRAA+ basic* define un conjunto de *thresholds* utilizados para decidir si incrementar o decrementar el valor del *rate* utilizado. El primero de estos *thresholds* es *Maximum Tolerable Loss* (*MTL*), el cual es utilizado para determinar un decremento en el *rate* utilizado. El segundo de los *thresholds* definidos es *Opportunistic Rate Increase* (*ORI*), el cual es utilizado para determinar un incremento del *rate* utilizado.

Para definir los valores de *MTL*, en [1] se define el *FLR* crítico de un *rate* R_i como el *FLR* que hace que R_i tenga el mismo *throughput* que el siguiente *rate* inferior (R_{i-1}) en caso de no haber pérdidas.

$$FLR_{critical}(R_i) = 1 - \frac{Throughput(R_{i-1})}{Throughput(R_i)} = 1 - \frac{TX_{time}(R_i)}{TX_{time}(R_{i-1})}$$

En base a dicho valor y teniendo en cuenta que es improbable que las pérdidas desaparezcan, se define el $MTL(R_i)$ de la siguiente forma:

$$MTL(R_i) = \alpha \times FLR_{critical}(R_i), \alpha \geq 1$$

Para definir el valor de *ORI*, se sigue un enfoque heurístico basado en la siguiente fórmula:

$ORI(R_i) = MTL(R_{i+1}) \times \beta$ donde R_{i+1} es el *rate* inmediatamente superior.

La idea principal consiste en que, para permitir un aumento de *rate*, el *FLR* debe ser menor al *MTL* del siguiente *rate*, de esta forma se evita que al aumentar el *rate*, el algoritmo deba decrementarlo inmediatamente por un aumento brusco del *FLR*. Este mecanismo, depende fuertemente del valor seleccionado de β, pudiendo generar oscilaciones en el *rate* si el valor es muy pequeño, o la imposibilidad de mejorar el *rate* si β es muy grande.

3.2.1.2. Funcionamiento *RRPAA*

Es posible considerar *RRPAA* como un algoritmo que utiliza estados basados en las distintas combinaciones de *power* y *rate* disponibles. Adicionalmente es necesario manejar un orden en éstos, cómo se define en los algoritmos 1 y 2.

Inicialmente *RRPAA* intenta encontrar el mejor *rate*, utilizando un *power* máximo, para luego, una vez que las pérdidas se encuentran estables, comenzar a decrementar el *power*.

Como se ha mencionado anteriormente, *RRPAA* está basado en *RRAA+ basic*, y es por esto que utiliza los mismos *thresholds* definidos por éste último. Dichos

1: **if** $p_0 == p_{max}$ **then**
2: $anterior(r_0, p_0) = (r_{-1}, p_0)$
3: **else**
4: $anterior(r_0, p_0) = (r_0, p_1)$
5: **end if**

Algoritmo 1: Pseudocódigo para obtener el estado anterior de RRPAA.

1: **if** $r_0 < r_{max}$ **then**
2: $siguiente(r_0, p_0) = (r_1, p_0)$
3: **else**
4: $siguiente(r_0, p_0) = (r_0, p_{-1})$
5: **end if**

Algoritmo 2: Pseudocódigo para obtener el estado siguente de RRPAA.

valores de *thresholds* son utilizados para cambiar el estado en el que se encuentra el algoritmo, es decir, en vez de aumentar o disminuir el *rate*, se avanza o retrocede el estado, pudiendo esto afectar tanto al *rate* como al *power*.

Finalmente, como fue mencionado anteriormente, la implementación de *RRAA+ basic* depende fuertemente del valor de β, pudiendo tener éste un impacto adverso para su ejecución. Por esto y con el fin de asegurar la convergencia del algoritmo, en [1] se incluye un mecanismo llamado *Probabilistic Rate Increase*. Dicho mecanismo consiste en mantener para cada estado del algoritmo una probabilidad de ingresar al mismo. Dicha estadística es decrementada cuando el algoritmo decide pasar a un estado anterior con el fin de dificultar volver fácilmente al estado saliente, mientras que es incrementada al ingresar a un estado siguiente. Cabe mencionar que para variar las probabilidades mencionadas anteriormente, se definen dos nuevos parámetros al algoritmos: γ, el cual es utilizado como un factor de decremento y δ el cual es utilizado como un factor de incremento. Finalmente, al momento de tomar acciones, el algoritmo incluye una variable aleatoria para controlar los cambios. En caso que dicha variable retorne un valor menor a la probabilidad del estado actual, la acción es tomada.

```
1: if loss > mtl(rate) and power < maxPower then
2:     priTable[rate][power] /= γ
3:     power++
4: else if loss > mtl(rate) and power == maxPower then
5:     priTable[rate][power] /= γ
6:     rate−−
7: end if
8: if loss < ori(rate) then
9:     for all r < rate do
10:        priTable[r][power] ∗= δ
11:    end for
12:    if rate < maxRate and power == maxPower and rand() <
   priTable[rate + 1][power] then
13:        rate++
14:    else
15:        for all p > power do
16:            priTable[rate][p] ∗= δ
17:        end for
18:        if rand() < priTable[rate][power + 1] then
19:            power−−
20:        end if
21:    end if
22: else if loss >= ori(rate) and loss < mtl(rate) and power > 0 then
23:    for all p > power do
24:        priTable[rate][p] ∗= δ
25:    end for
26:    if rand() < priTable[rate][power + 1] then
27:        power−−
28:    end if
29: end if
```

Algoritmo 3: Pseudocódigo RRPAA. Fuente [1].

El algoritmo completo se detalla en Algoritmo 3 y a modo de resumen, el algoritmo se divide en tres casos según el valor del *FLR*.

1. Si el *FLR* se encuentra entre los valores de *ORI* y *MTL*, se procede a decrecer el *power* seleccionado mientras el *FLR* no supere el *MTL*.

2. Si el *FLR* es superior al *MTL*, se procede a aumentar el *power* hasta

su máximo, en caso que el FLR no decremente, se procede a disminuir el *rate*.

3. Si el FLR se encuentra debajo del ORI, el algoritmo procede a aumentar el valor del *rate*, hasta que se alcance el valor máximo. En caso de alcanzar un *rate* máximo se procede a decrementar el *power*.

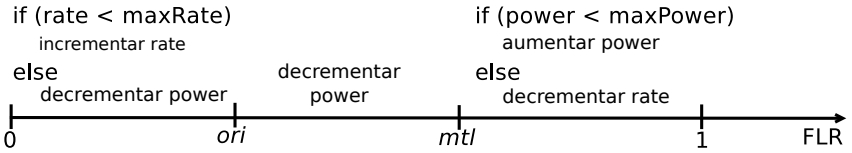

Figura 3.1: Decisiones tomadas por RRPAA. Fuente [1].

3.2.2. *PRCS*

En [1] se realizan simulaciones del algoritmo $RRPAA$, las cuales desprenden que dicho algoritmo puede sufrir problemas de *Starvation*. Para evitar dicho problema se plantea $PRCS$, una mejora a $RRPAA$. Esta mejora incluye el mecanismo de control de *Carrier Sense*, el cual consiste en modificar dicho parámetro buscando eliminar posibles asimetrías entre los enlaces.

Para su funcionamiento es necesario considerar una nueva variable, la probabilidad de encontrar el medio ocupado al intentar transmitir ($PBUSY$), adicionalmente se define un nuevo *thresholds* para el $PBUSY$. Una vez definido dicho *threshold*, en caso que el $PBUSY$ lo supere es aumentado el valor del CST, en caso negativo, si el FLR supera el MTL y es posible disminuir el CST, éste es disminuido.

Capítulo 4

Prototipo

4.1. Diseño e implementación

Debido a la gran variedad de dispositivos disponibles resulta necesario investigar la factibilidad y conveniencia de las posibles alternativas, no solo en *hardware*, sino que también en *software* disponibles para alterar su comportamiento. Un trabajo destacable es el realizado por Thomas Hühn[11], en el cual no solo realiza una investigación de las posibles plataformas, sino que también prueba varias alternativas sacando conclusiones interesantes.

4.1.1. Dispositivos

Para el desarrollo del prototipo es necesario realizar modificaciones al *driver* del dispositivo, por lo que deben ser descartados los dispositivos en los cuales la lógica de capa de enlace se encuentre implementada en *hardware*. Otras consideraciones a tener en cuenta son[12]:

- Soporte de variantes de la norma 802.11 tales como 802.11a, 802.11n, etc.
- Contar con suficientes recursos de memoria y *CPU* para poder realizar mediciones estadísticas.
- La cantidad de dispositivos que se pueden conectar al *AP*.

Otro punto a evaluar es el tiempo disponible del *AP* para realizar cálculos extra sin degradar considerablemente su funcionamiento, con el fin de realizar las estimaciones requeridas, pudiendo llegar a ser necesario agregar algún dispositivo adicional en una etapa inicial.

Con lo anteriormente mencionado, si se compara los controladores, es posible observar que hay dispositivos como por ejemplo *Broadcom BCM4712*, o *Intel 5300* que poseen un *firmware* en el mismo *chip* del controlador, lo que dificulta o hasta imposibilita realizar modificaciones a algunos comportamientos desde el *driver*. Por otra parte, dispositivos como los de *Atheros* poseen las funcionalidades para las cuales el tiempo es crítico implementados directamente en *hardware*, dejando el resto del comportamiento en el microcontrolador. A su vez si se compara implementaciones *open source*, se puede apreciar que este último es el que posee más implementaciones de este tipo, lo que resulta conveniente debido al acceso a documentación y ayuda por parte de la comunidad que permiten dichos proyectos.

Por último, es importante mencionar la utilización de un sistema operativo que permita las flexibilidades que son requeridas para la realización del prototipo, en este sentido se encuentran alternativas como *OpenWRT*[13], *Lede*[14], *DD-WRT*[15], *pfSense*[16], entre otros.

Con el fin de cumplir las restricciones anteriormente mencionadas y teniendo en cuenta los recursos a los que fue posible acceder, el algoritmo es desarrollado para el *driver* ath9k para dispositivos *Atheros*. Dicho *driver* se encuentra publicado bajo licencias de código abierto, por lo que es posible acceder al código y modificarlo con pocas restricciones.

4.2. Mac80211

Mac80211 es un *framework* utilizado para el desarrollo de drivers *WiFi* de tipo *softMAC*. Este tipo de *drivers*, a diferencia del anterior paradigma (*fullMAC*), delega una parte de la funcionalidad necesaria para el funcionamiento del dispositivo al *kernel* del sistema operativo. En particular, *mac80211* presenta un conjunto de primitivas que permiten principalmente:

- Resolver la traducción de paquetes de capas superiores a tramas *IEEE 802.11* y viceversa.
- Implementar un conjunto de operaciones de control referidas al estándar *IEEE 802.11*, como podría ser la asignación de *rate*, ahorro de energía, agregación de tramas, entre otros.

4.3. *Ath9k*

Ath9k es un *driver* para controladores *WiFi* de una gran variedad de dispositivos desarrollados por la empresa *Qualcomm Atheros*. Dicho *software* se encuentra bajo

una licencia de código abierto y fue incluido por primera vez en el *kernel* de *linux* en agosto del 2008[17].

Una de las características principales por la cual utilizar este *driver* en el transcurso del proyecto, es la dependencia que tiene con *mac80211*, permitiendo entre otras cosas, realizar modificaciones al algoritmo de control de *rate* simplemente modificando módulos incluidos en el *kernel* de *Linux*, sin necesidad de modificar la lógica incluida en los controladores.

4.4. Arquitectura

A nivel general, la arquitectura en la que es posible localizar a *ath9k*, es presentada en la figura 4.1. En esta arquitectura, las capas superiores del *IP stack* de *Linux* se comunican con *mac80211* para el envío de tramas a través de *WiFi*, mientras que *mac80211* sirve de intermediario entre las capas superiores y el controlador de red[18].

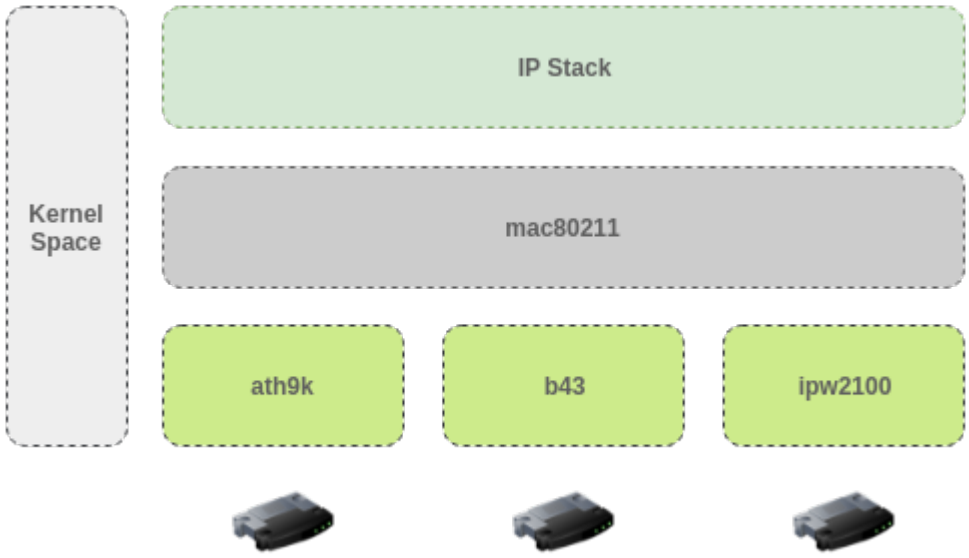

Figura 4.1: Diagrama de la arquitectura de ath9k.

En esta arquitectura las funciones de cada capa quedan bien definidas y con responsabilidades claras. La capa inferior, donde residen los *drivers* de los controladores, tiene como cometido principal el manejo de la comunicación entre la interfaz física y los módulos del sistema operativo.

En una capa intermedia es posible encontrar a *mac80211*, el cual resuelve tareas de control propias del protocolo *IEEE 802.11* como se mencionó anteriormente.

Finalmente es posible encontrar el resto del *IP stack* sobre *mac80211*. Esto es debido a que los sistemas *Linux* cuentan con una interfaz virtual, la cual recibe

los paquetes provenientes del *IP stack*, siendo mac80211 quien implementa dicha interfaz.

4.5. Análisis

Considerada la arquitectura utilizada por *Linux* para el control de dispositivos que cumplen el estándar *IEEE 802.11*, la implementación se centra principalmente en la capa de *mac80211*, teniendo como ventaja destacable, que al ser implementado en una capa intermedia y respetando las interfaces definidas, es posible portar la implementación generada a otros *drivers* que utilicen *mac80211* como *framework* base.

4.5.1. Rate control

La mayoría de los *drivers* que utilizan *mac80211* utilizan los algoritmos de control de *rate* provistos por el *framework*. *Mac80211* implementa dos algoritmos de adaptación de rate los cuales son, *Minstrel* y *PID*[11], permitiendo a su vez la incorporación de nuevos algoritmos y delegando la elección del mismo al *driver*. Todos los algoritmos de control de *rate* son registrados en *mac80211*, los cuales deben implementar una interfaz con un conjunto de operaciones base definidas en la estructura *rate_control_ops* [19]. Del *set* de operaciones disponibles cabe destacar las siguientes operaciones:

- ***alloc_sta*:** es invocada en el momento que un cliente se vincula al dispositivo, tiene como cometido la reserva del espacio de memoria para la estación y la inicialización de algunas variables referentes a la sesión.
- ***rate_init*:** inicializa información referente a los *rates* a utilizar en el transcurso de la sesión.
- ***tx_status_noskb*:** es llamada luego del envío de un paquete, la cual recibe información estadística del resultado del mismo.
- ***free_sta*:** es invocada al momento de la desvinculación del terminal

4.5.2. *Retry Chain*

Los *drivers* que basan su funcionamiento en el *framework mac80211* tienen a su disposición la posibilidad de definir, para cada trama a enviar, una cantidad de reintentos posibles y para cada uno de estos, los valores de *rate* a utilizar. Con

dicho fin se utiliza la estructura *ieee*80211_*sta_rates*, la cual contiene los siguientes atributos destacables:

- ***idx*:** identificador del *rate* a utilizar.
- ***count*:** la máxima cantidad de intentos que serán enviados con dicho *rate*
- ***count_cts*:** cantidad de intentos de tramas *clear to send*.
- ***count_rts*:** cantidad de intentos de tramas *request to send*.

Cabe mencionar que la cantidad máxima de valores posibles de este arreglo es definido por el *hardware*, el cual se lo comunica al *framework* a través del *driver*.

Para indicar el fin de los valores seleccionados, en la *retry chain* es posible ingresar una cantidad de intentos a realizar con valor -1, pudiendo estar este valor en la primer posición, de esta forma únicamente se intentaría un envío con el *rate* definido por el mecanismo de *rate control*.

Como parte del trabajo realizado y con el fin de poder definir un nuevo valor de *power* para cada reintento, se agrega un nuevo atributo a la mencionada estructura. De esta forma, tanto la decisión del *rate* y *power* principales, como el de los sucesivos reintentos a utilizar en caso de falla, es manejado dentro de *mac80211*. Gráficamente esto es posible verlo en la figura 4.2, en donde en cada posición de la *retry chain* contiene el valor de *rate* (r_i), *power* (p_i) y la cantidad (c_i) de intentos con dicha configuración.

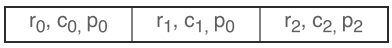

Figura 4.2: Diagrama de la nueva retry chain.

4.5.3. *Power*

Originalmente el *power control* es realizado directamente por el *driver* del dispositivo de red, el cual originalmente utiliza la información provista en la configuración de la interfaz de red para definir dicho valor. Cada país regula las frecuencias y el *power* máximo que pueden ser utilizados[20], es por esto que los posibles valores de *power* dependen principalmente del país en donde será utilizado el dispositivo.

Debido a la información necesaria para la toma de decisiones y la separación en capas mencionada, se decidió extender la interfaz de *mac80211* con el fin de incluir *power control* a éste, en otras palabras, el *framework* es el encargado de tomar la decisión sobre el valor de *power* a utilizar en el envío de las distintas tramas.

Conjuntamente, debido a que ath9k es el encargado de comunicarle dicho valor a la interfaz física, resulta necesario realizar cambios en el *driver* con el fin de respetar el *power* notificado por la nueva versión de *mac80211*.

Para lograr dicho comportamiento, se utilizó como base el código desarrollado en [11], el cual modifica algunas de las estructuras de *mac80211* para incluir un valor de *power*. El mismo se encuentra actualmente publicado bajo la licencia *GNU General Public Licence* en un repositorio público en *GitHub*[21]. Partiendo de la mencionada implementación, se procedió a vincular las estructuras modificadas, con el *driver ath9k*.

En esta implementación se tomó la decisión de diseño de trabajar con un conjunto de *powers* estático, tomando como valor máximo 19 dBm. Dicha decisión fue tomada para poder realizar las pruebas en un ambiente reducido y controlado, ya que aún utilizando dicho *power* máximo, la distancia a la cual es posible recibir las transmisiones es cercana a los 100m.

Cabe mencionar que el *driver* ath9k, utiliza la unidad decibelio-milivatio (dBm) para expresar los distintos valores de *powers*. Dado que esta unidad está definida de forma no lineal, al expresarla en mW existen valores que no presentan cambios significativos entre sí, por lo se decidió excluirlos tal como lo hace la interfaz gráfica de *OpenWRT* (Luci), al momento de seleccionar el valor de *power*. Los valores de *power* utilizados en este prototipo se encuentran en el cuadro 4.1

Power(dBm)	Power(mW)
0	1
4	2.512
5	3.162
7	5.012
8	6.310
9	7.643
10	10
11	12.589
12	15.849
13	19.953
14	25.119
15	31.623
16	39.811
17	50.119
18	63.096
19	79.433

Cuadro 4.1: Equivalencia de valores de power en dBm y mW.

4.5.4. *Carrier-Sense*

El *carrier sense* fue investigado y excluido del alcance del presente proyecto, debido a que la información necesaria para implementar dicho control depende del *hardware*.

Como resultado de la investigación realizada fue posible concluir que para aplicar el control de *carrier sense*, resulta necesario acceder y modificar ciertos registros del *hardware*, teniendo en cuenta que las compañías responsables de dichos dispositivos suelen pactar acuerdos de confidencialidad con los involucrados en el desarrollo[1], lo cual dificulta considerablemente el acceso a la misma.

En caso de incluir el control de *carrier sense* en un futuro, es posible realizar modificaciones similares a las implementadas para el control de *power*, logrando una comunicación entre el *driver* y *mac80211*. Posteriormente sería necesario realizar modificaciones en el *driver* correspondiente con el fin de modificar los valores de los registros involucrados. Incluyendo este control de la forma mencionada, se podría obtener un algoritmo completo de control de *power*, *rate* y *carrier sense*, totalmente incluido en el *framework mac80211* y aumentando su portabilidad, ya que únicamente sería necesario ajustar nuevos *drivers* a la interfaz de *mac80211* para consumir la información proporcionada por *PRCS*.

4.6. Implementación

En esta sección se detallan las etapas principales de la ejecución del algoritmo, brindando en cada una de ellas un detalle del desarrollo realizado para la misma.

Para la implementación del código se utilizó como base la implementación existente de *Minstrel* incluida en el *kernel* de *OpenWRT* trabajado (3.18.29), en particular el prototipo es basado en la modificación de dos módulos del *kernel* de *OpenWRT,* los cuales son *kmod-mac80211* y *kmod-ath9k.* En el primero se encuentra propiamente el algoritmo desarrollado, mientras que en el segundo se realizan cambios para respetar cuestiones tales como el manejo de distintos valores de *power*.

En *kmod-mac80211* se destacan tres nuevos archivos desarrollados, las cuales son:

- ***rc80211_rrpaa.h***: Se definen las constantes y estructuras a utilizar por el algoritmo, así como algunas funciones que deben ser utilizadas por otros archivos del módulos.

[1] Esta información fue proporcionada por miembros de la comunidad *open source* que estuvieron vinculados a *Atheros* y nos proporcionaron información referente a dispositivos que funcionan con *ath5k*

- ***rc80211_rrpaa.c***: En este archivo se encuentra el algoritmo desarrollado.

- ***rc80211_rrpaa_debugfs.c***: Se define las funciones para la obtención de los datos estadísticos, en el cual destacan dos funciónes: la primera puede ser utilizada para visualizar los datos estadísticos en tiempo real (*rrpaa_stats_open*) y la segunda función permite obtener los datos estadísticos recolectados en un periodo de tiempo (*rrpaa_time_state_open*).

El prototipo desarrollado fue basado en el estándar *IEEE 802.11g*, sin embargo este protocolo puede ser utilizado en los otros estándares de *WiFi*, debido a que todos ellos deben implementar la misma interfaz.

El código desarrollado es posible dividirlo en tres etapas principales:

1. Inicialización de estructuras de datos a utilizar a lo largo de la sesión

2. Relevamiento de datos estadísticos

3. Toma de decisiones en base a los datos recolectados y asignación de parámetros a utilizar en las futuras transmisiones.

4.6.1. Inicialización

Esta es la primer etapa del algoritmo, la cual es invocada al momento en que una estación es asociada al *AP*. En esta etapa se utilizan dos de las funciones provistas por *mac*80211, *alloc_sta* y *rate_init*.

De las funciones anteriormente mencionadas, *alloc_sta* es la primera en ser ejecutada y es en donde es reservada la memoria a utilizar a lo largo de la sesión entre ambos dispositivos. Dicha sesión es almacenada en la estructura $rrpaa\_sta\_info$, la cual contiene, entre otra información, la tabla de *thersholds*, la tabla de probabilidades, marcas de tiempo, tabla de *rates* disponibles. De esta información almacenada, cabe destacar la estructura *rrpaa_rate*, la cual cuenta con los siguientes atributos destacables:

- ***bitrate*:** almacena el valor del bitrate multiplicado por una constante.

- ***rix***: identificador del *rate* en el *driver*.

- ***rrpaa_rate_stats***: almacena información estadística asociada al *rate*. Dentro de esta información es posible encontrar la última cantidad de intentos y transmisiones exitosas realizado con dicho *rate* (*attempts* y *success* respectivamente) y un contador utilizado para el control del valor de la ventana definido por *RRPAA* (*ewnd_count*).

Cabe destacar que al momento de invocar a *alloc_sta*, es el único momento en donde se reserva memoria y aún no se han definidos algunos parámetros a utilizar. Por ejemplo, no se han definido los valores de *rates* válidos a lo largo de la sesión. Esto implica que es necesario reservar memoria extra con el cometido de prever todos los casos posibles.

Luego de ser reservada la memoria, es invocada por el *driver*. La función *rate_init*, la cual en una primera instancia filtra los rates que pueden ser utilizados, obteniendo únicamente los rates soportados por ambas terminales involucradas en la sesión. En esta etapa se asocian los rates utilizados por *mac80211*, con los rates utilizados por el *driver*. Adicionalmente se los ordena en forma decreciente, con el fin de simplificar su consumo al momento de calcular la tabla de *thresholds* y asignación de *rate* a utilizar.

Luego de inicializados los *rates*, se procede al cálculo de la tabla de *thresholds*, siendo definida con este fin la función *threshold_init*. Dicha función tiene como cometido calcular los valores de *ORI* y *MTL* partiendo de los *rates* emparejados y previamente ordenados. Con este fin se define la estructura *threshold*, la cual cuenta con los siguientes atributos:

- ***mr_idx*:** posición del *rate* dentro del arreglo de *rates* disponibles almacenado en $rrpaa\_sta\_info$.
- ***ori*:** valor de *ORI* definido por el algoritmo para el rate mr_idx.
- ***mtl*:** valor del *MTL* definido por el algoritmo para el rate mr_idx.
- ***ewnd*:** tamaño de la ventana definido por el algoritmo para el rate mr_idx.

Cabe mencionar que los valores de la tabla de *thresholds* no son establecidos de forma estática, sino que son calculados en dicha función en base a los *rates* disponibles. De esta forma se brinda soporte a distintos conjuntos de *rates*, ya que éstos pueden variar dependiendo de los terminales involucrados en la sesión.

4.6.2. Recolección de datos

Esta es una de las etapas principales del algoritmo, debido a que en ella se recaban todos los datos a ser utilizados en la toma de decisiones. Para esta etapa se utiliza la función tx_status_noskb de *mac80211*, la cual es invocada cuando el proceso de envío de una trama es finalizado. Esta función recibe como parámetro una estructura $ieee80211\_tx\_info$, la cual contiene, para cada *rate*, la información de la cantidad de intentos realizados, intentos exitosos y el valor de *power* utilizado en dichos intentos. Esta información es almacenada en $rrpaa\_rate\_stats$, la cual se encuentra dentro de la estructura $rrpaa\_rate$.

Por otro lado, con el fin de aumentar la información disponible al realizar tareas de *debugging*, se almacena el *power* y *rate* del estado actual del algoritmo, junto con los intentos necesarios para el envío de la trama en un arreglo con tope. Estas estadísticas son las utilizadas para generar las gráficas del presente informe. Dicha recolección de datos puede ser habilitada o deshabilitada fácilmente en tiempo de compilación.

4.6.3. Toma de decisiones

Luego de actualizar los valores estadísticos, es invocada la función $rrpaa\_update\_stats$, la cual tiene como cometido principal, aplicar el algoritmo propiamente dicho. Con el mencionado fin, esta función primeramente verifica para cada valor de *rate*, si se ha alcanzado el límite de transmisiones definidos por la ventana de *threshold* para el *rate* analizado, aplicando para cada *rate* que cumpla con dicha condición las acciones definidas por *RRPAA*. Esto implica que el algoritmo puede cambiar de estado de forma consecutiva, múltiples veces en un mismo paso de la iteración.

4.6.4. Asignación de *power*, *rate* y *retry chain*

Una vez que se ha ejecutado *RRPAA* y se han definido los nuevos valores de *power* y *rate* a utilizar, se procede a cargar la estructura la cual es utilizada para informar al *driver* los nuevos valores a utilizar. Para esto es utilizada la función $rrpaa\_update\_rates$. Dichos valores deben ser asignados en la *retry chain*, tanto los valores a utilizar por el primer envío, como los correspondientes a los reintentos.

Una vez definidos los parámetros a utilizar, es necesario decidir qué hacer con los distintos reintentos. Es decir, si se permitirán, cuantos se harán y con qué parámetros. En esta primer versión de *RRPAA* se ha decidido no deshabilitar dicha funcionalidad debido a las mejoras sustanciales que este mecanismo busca aportar en cuanto al rendimiento.

Con el fin de aumentar la probabilidad del envío de las tramas, los siguientes valores de la *retry chain* son cargados siguiendo el algoritmo y asumiendo el peor de los casos posibles (es decir que manteniendo el *rate* y *power* fallarán las retransmisiones), de esta forma se sigue una lógica como la presente en el algoritmo 4.

```
1: while can_add_items_to_retrychain do
2:     if power < p_max then
3:         power = next_power
4:     else if rate > rate_min then
5:         rate = previous_rate
6:     end if
7: end while
```

Algoritmo 4: Pseudocódigo para asignación de valores en *retry chain*.

En otras palabras, en cada posición de la *retry chain*, se intenta utilizar el *power* superior siguiente, en caso de no ser posible se intenta disminuir el *rate*, siendo esta acción realizada como máximo tres veces en la actual implementación, dejando una cuarta posición con un valor de rate mínimo y power máximo, buscando asegurar la transmisión de la trama. De esta forma se busca respetar las decisiones del algoritmo en caso de transmisiones fallidas.

Cabe mencionar que es posible definir un único valor en la *retry chain* y obtener una implementación aún más fiel a la definición del algoritmo, pero dado que se cuenta con esta funcionalidad, se decide utilizarla y disminuir el *overhead* que implicaría el procesamiento de las retransmisiones. De no utilizar esta funcionalidad, en casos donde todas los intentos de la *retry chain* resultan fallidos, se ejecutarán las funciones asociadas al fin de la transmisión de una trama, generando un *overhead* innecesario y posiblemente degradando la performance del algoritmo.

Finalmente, debido al impacto que puede tener esta decisión en la *performance* del algoritmo, resulta necesario realizar, en un trabajo futuro, un conjunto de pruebas destinadas a definir los valores a utilizar en esta sección y definir la permanencia de este mecanismo.

Capítulo 5

Pruebas realizadas

Con el fin de evaluar el comportamiento y rendimiento del algoritmo desarrollado (*RRPAA*), se plantean distintos escenarios en los cuales se ejecuta el algoritmo y se compara con la implementación de *Minstrel*[22] incluido en la distribución *OpenWRT*[13].

5.1. Objetivos

Se plantea como objetivo principal de las pruebas la evaluación del correcto funcionamiento del algoritmo, buscando comprobar que es fiel a su especificación.

Como segundo objetivo, se plantea la comparación de *RRPAA* frente a un algoritmo de similares caracteristicas, estable, estudiado y muy utilizado en la actualidad, como es Minstrel.

Un tercer objetivo planteado consiste en someter el algoritmo a distintos escenarios, con el fin de evaluar su comportamiento en ambientes para los cuales fue pensado, como ambientes para los que no. Con esto último se busca evaluar la viabilidad de utilizar *RRPAA* en distintos escenarios y obtener datos que permitan comparar el comportamiento del algoritmo con una implementación futura de *PRCS*.

Como objetivo final, se busca analizar los resultados obtenidos, con el fin de plantear posibles mejoras y adaptaciones, buscando la evolución del algoritmo a una versión estable.

5.2. Mediciones obtenidas

Para cada prueba se generó un tráfico constante *UDP* a 54Mb/s, siendo dicho *rate* el máximo al cual puede trabajar el estándar *WiFi* en el que fue trabajado, *802.11g*[23]. De esta forma se busca que el dispositivo transmita en todo momento.

Por otro lado se utilizan paquetes *UDP* con el fin de evitar el *overhead* que implican las retransmisiones *TCP* asociadas a las pérdidas de mencionado tráfico fue generado mediante la herramienta *Iperf3*[24].

Durante las pruebas se registraron los siguientes datos:

- *Throughput*, para obtener este dato fue utilizada la misma herramienta de generación de tráfico, la cual brinda reportes a intervalos regulares, en este caso el intervalo utilizado es de 1 segundo.
- *Rate*, *power*, *attempts* y *success*, estos fueron obtenidos directamente del código ejecutable con el fin de tener una mejor visión del comportamiento de los algoritmos, siendo éstos obtenidos, luego que el sistema descarta o envía exitosamente una trama. Estos datos estadísticos son alojados en memoria *RAM* del módulo del *kernel*, por lo cual fue necesario la utilización del sistema de archivos virtual para lograr la comunicación entre el kernel y el proceso que se encarga de alojar estos datos. Dicho proceso se encarga de recolectar todos los datos estadísticos y almacenarlos en una memoria externa debido a los escasos recursos del router. Es de destacar que es posible deshabilitar la recolección de estos datos estadísticos en tiempo de compilación.
- *Busy ratio*, para obtener el porcentaje de tiempo en el cual el medio se encuentra ocupado, es utilizado en los routers un *script* que obtiene esta medida mediante la herramienta iw[25], con reportes a intervalos de 1 segundo.
- *Loss-rate*, en base a los datos obtenidos de *attempts* y *success* de las distintas ejecuciones, se agrupan los registros y se calcula el porcentaje de intentos exitosos y fallidos para cada trama.
- Consumo de energía, en base a los datos obtenidos de *attemps*, *rate* y *power*, se calcula la variación porcentual de energía utilizada en cada una de las ejecuciones en ambos protocolos. Para esto se asume que todas las tramas tienen igual tamaño, debido a que el generador de tráfico envía paquetes iguales en cada momento. Teniendo en cuenta que la energía insumida para el envío de una trama es posible calcularla como:

$$\frac{power \times tamaño}{rate}$$

es posible calcular la variación porcentual de energía como:

$$\left(\sum \frac{Power_{RRPAA}(i)}{Rate_{RRPAA}(i)} - \sum \frac{Power_{Minstrel}(i)}{Rate_{Minstrel}(i)} \right) \times \frac{1}{\sum \frac{Power_{Minstrel}(i)}{Rate_{Minstrel}(i)}}$$

Cabe destacar que para cada escenario y algoritmo analizado, se realizan tres ejecuciones del mismo y los datos obtenidos de éstas son agrupados con el fin de obtener los datos estadísticos anteriormente mencionados. Tomando en cuenta la reducida cantidad de ejecuciones, los posibles resultados no tienen valor estadístico, sin embargo permiten tener un panorama del comportamiento de los algoritmos.

En la presente sección, como regla general, para los gráficos referentes al *throughput* obtenido por los algoritmos, se presentan gráficos de tipo *boxplot*, en el cual se grafica la mediana, el primer cuartil, tercer cuartil, los valores máximos y mínimos, luego de eliminar los valores atípicos.

Finalmente es necesario aclarar que los valores de *rate* y *power* obtenidos con el fin de generar los siguientes gráficos son asociados al estado en el cual se encuentra el algoritmo, en lugar de asociarlos a los valores de *power* y *rate* con el cual fueron enviados. Sin embargo, al momento de registrarlos para la toma de decisiones, estos datos son correctamente asociados a los valores con los cuales fueron enviados.

5.3. Configuración de los escenarios

Para la ejecución de las diferentes pruebas, se obtuvo acceso a las instalaciones del Instituto de computación (InCo) de la Facultad de Ingeniería. De esta forma fue posible acceder a un ambiente con pocas interferencias de redes *WLAN* (debido a que no había redes en canales cercanos), llevando a cabo las pruebas en momentos con escasa cantidad de gente dentro del instituto.

En cuanto al equipamiento empleado para las mismas, fueron utilizados routers *TP-Link WDR4300* con *OpenWRT,* a los cuales se les fue instalado el algoritmo a evaluar por cada escenario. Adicionalmente como nodos conectados a los routers fueron empleados ordenadores personales, ambos con tarjetas de red de marca *Intel* de similares características.

5.4. Escenarios analizados

En esta sección se analiza el comportamiento de *RRPAA* en comparación con *Minstrel* en tres escenarios muy distintos entre sí. En una primera instancia se busca analizar el comportamiento de *RRPAA* en un escenario para el cual el algoritmo no fue desarrollado. En él se plantea evaluar el comportamiento de un router emisor con un nodo receptor a medida que se aumenta la distancia entre ellos.

En una segunda prueba se plantea un escenario favorable a *RRPAA*, en el cual existen dos routers transmitiendo información, cada uno a un nodo particular, de forma tal que ambos *routers* generan interferencias entre sí. En dicho escenario se busca evaluar la reducción del power utilizado por parte de *RRPAA* con el fin de generar menos interferencia, dejando un canal más limpio y libre para enviar la información a los respectivos nodos.

En una última prueba, se plantea un escenario equitativo a ambos protocolos, en los cuales se cuenta con un *router* transmitiendo información a dos nodos distintos ubicados a diferentes distancias del *router*. En dicho escenario se busca comprobar el comportamiento del algoritmo en los distintos nodos, comparando los resultados de ambos protocolos y permitiendo confirmar la utilización de parámetros independientes para cada enlace.

5.4.1. Diferentes distancias

Con este escenario se busca analizar el comportamiento de la implementación de *RRPAA* en función de la distancia respecto al *router*, así como evaluar el algoritmo en un escenario para el cual no fue optimizado. Para esto se plantea un escenario en donde se coloca un *router* y una terminal a una distancia inicial de 3 metros. Posteriormente a intervalos de 3 minutos se aleja el terminal 6m hasta llegar a una distancia final de 63m. Esta prueba es realizada 3 veces con *RRPAA* y 3 veces con la implementación original de Minstrel.

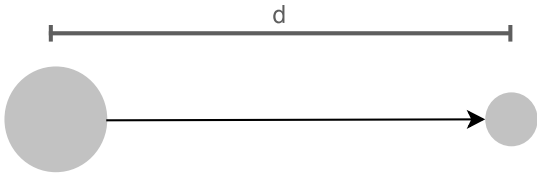

Figura 5.1: Diagrama del escenario de múltiples distancias.

5.4.1.1. Resultado esperado

En este escenario es esperable que el rendimiento de ambos algoritmos sea similar, es decir, que ambos tengan un *throughput* similar. En cuanto al *power* utilizado, es esperable que en las distancias iniciales éste sea mínimo y a medida que se aumente la distancia éste sea aumentado. Se espera que la tasa de pérdidas sea levemente superior en el caso de *RRPAA* debido a la disminución del *power* y tiempo de adaptación del mismo.

5.4.1.2. Resultado obtenido

Cómo es posible observar en el gráfico 5.2, el *throughput* obtenido por *Minstrel* es significativamente superior al obtenido por *RRPAA*, llegando en el peor de los casos a una variación del 50 %. En dichas gráficas, la diferencia de *throughput* no es significativa hasta los 21 metros. Luego de este valor, la diferencia tiende a aumentar casi de forma constante. Este comportamiento es posible explicarlo observando la pérdida de paquetes y el *rate* utilizado, como se pueden apreciar en las figuras 5.9 y 5.6 respectivamente. Debido a la cantidad de pérdidas, *RRPAA* busca aumentar el *power* y disminuir el *rate* hasta llegar a un equilibrio y estabilizarse. Debido al bajo *rate* en comparación con *Minstrel*, la cantidad de datos transferidos es significativamente menor lo que conlleva a un menor *throughput*.

Por otro lado si se compara la distancia entre los cuartiles, se observa que *RRPAA* posee una dispersión superior a *Minstrel*. Como se observa en los graficos 5.3 y 5.4, es posible observar que este efecto es debido al procesamiento de los datos realizados, es decir, el agrupamiento de los datos de las distintas ejecuciones.

Como se observa en la figura 5.3, las ejecuciones poseen un comportamiento similar, pero con diferencias que llegan hasta cerca de los 400 B/s. Por lo anteriormente mencionado, al agrupar los datos para su posterior procesamiento, según los distintos intervalos de tiempo, es que se presentan las diferencias entre los cuartiles anteriormente mencionadas.

En la figura 5.4 se puede observar un gráfico similar al de la figura 5.2, pero únicamente con información de una de las ejecuciones. Se presenta dicho gráfico con el fin de mostrar, que dentro de las distintas ejecuciones, la variación del *throughput* no es tan importante como en el gráfico agrupado con los datos de todas las ejecuciones (figura 5.2). En particular la figura 5.4, muestra una menor variación en el *throughput* obtenido,lo cual es posible observar en cada una de las ejecuciones realizadas. En cambio, al observar el gráfico con los datos agrupados 5.2, se obtiene una variación mayor debido a que distintas ejecuciones obtienen distintos resultados de *throughput* a nivel general. A pesar de la reducción de la variación del *throughput*, es posible

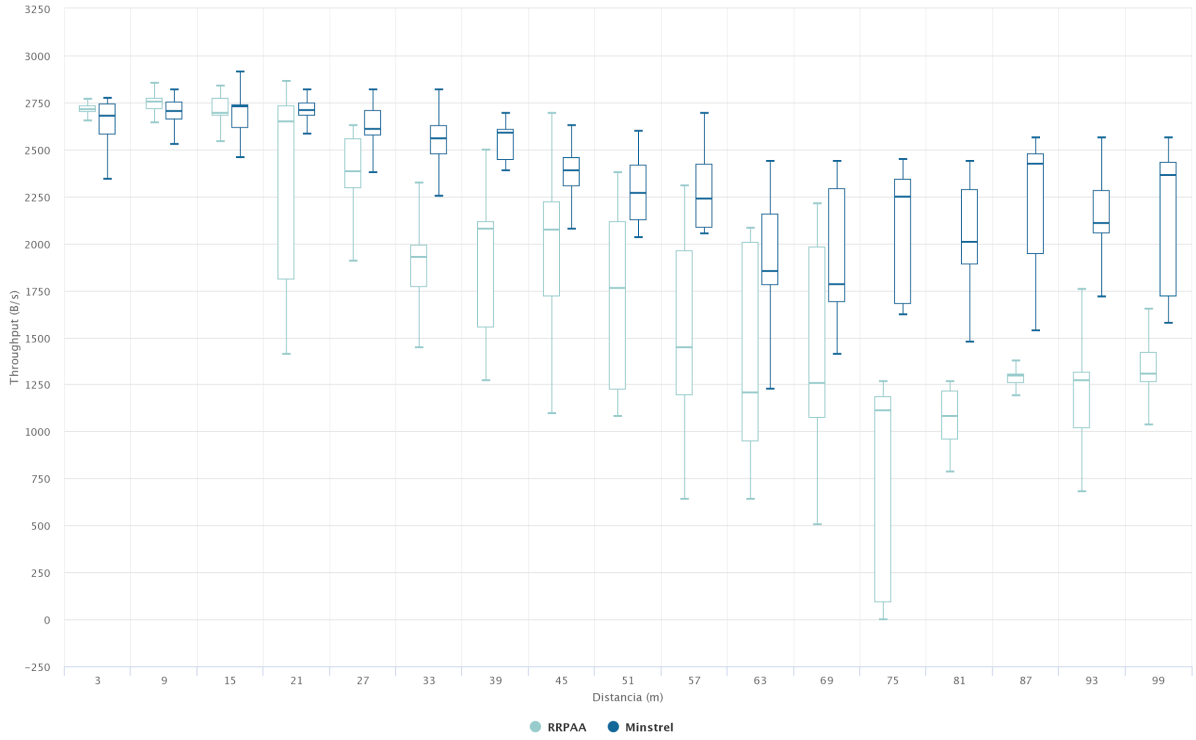

Figura 5.2: En este gráfico se presenta, agrupado por distancia y juntando los datos de todas las ejecuciones, los cuartiles de los throughputs obtenidos de los algoritmos.

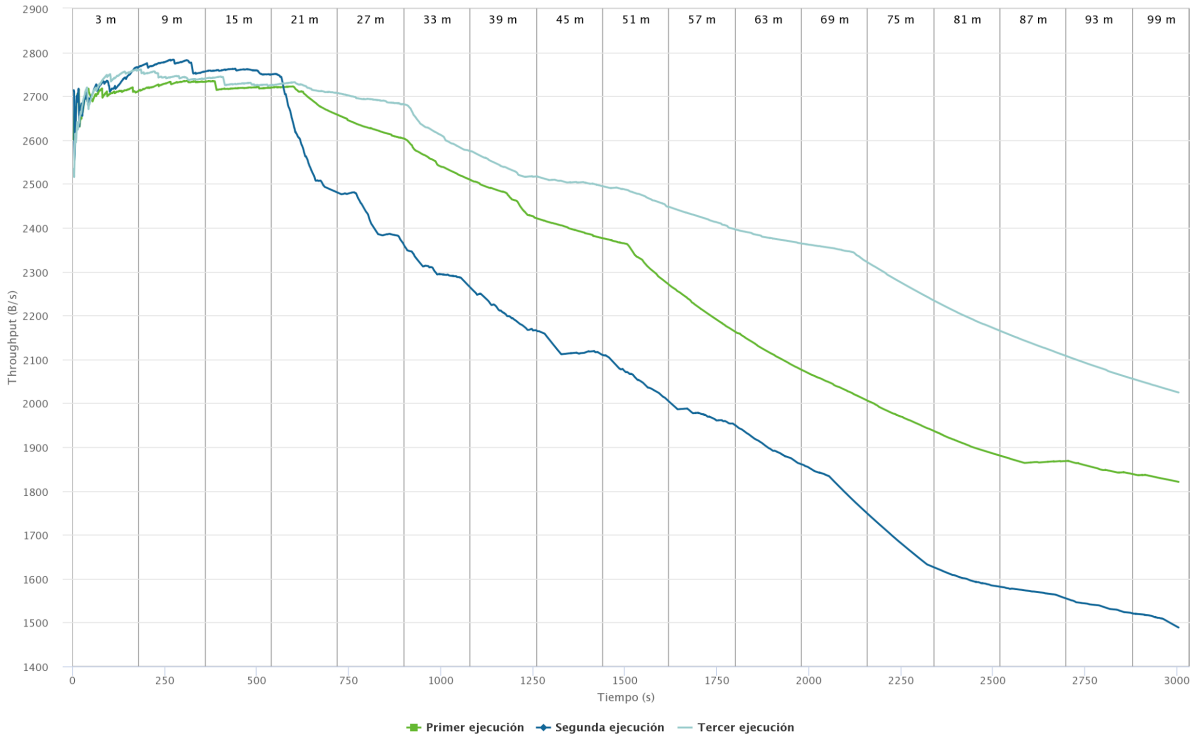

Figura 5.3: Este gráfico presenta el throughput de las tres ejecuciones del algoritmo RRPAA, en función del tiempo.

ver que el comportamiento general se condice con el gráfico sumarizado, en donde, en términos generales, luego de los 21m de distancia, *RRPAA* tiene un resultado significativamente inferior al reportado por *Minstrel*.

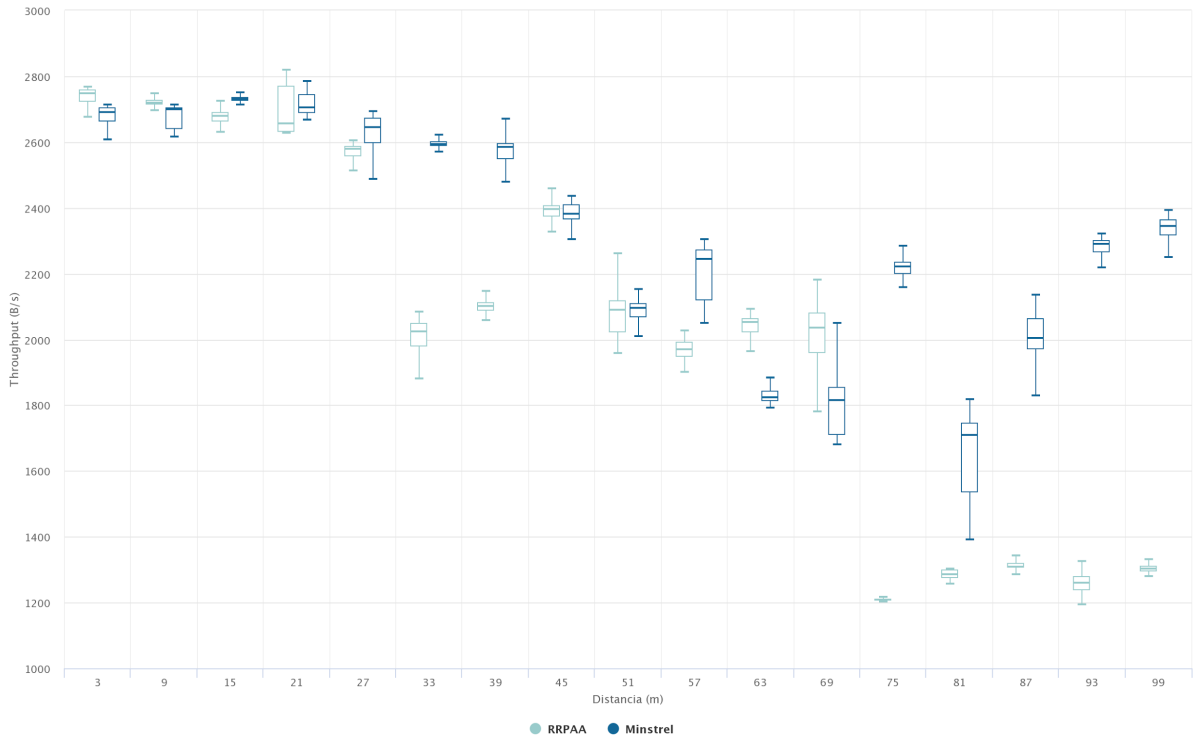

Figura 5.4: Throughput, en este gráfico se presenta para una ejecución de los algoritmos, agrupado por distancia, los cuartiles de los throughputs obtenidos de los algoritmos.

Analizando los valores de *power* (figura 5.5) y *rate* (figura 5.6) utilizados, es posible observar que los valores se encuentran cercanos a los esperados, pudiendo en algunos casos ser deseable un valor superior de *power*. En particular, es deseable que en las distancias mayores, se utilice principalmente los *power* mayores, buscando disminuir las pérdidas obtenidas.

En las figuras 5.7 y 5.8 se presenta para las distintas ejecuciones de *RRPAA* y para cada valor de *power* posible, el porcentaje de veces en que fue utilizado dicho valor. De esta forma es posible observar que únicamente en una de las ejecuciones se utiliza principalmente el *power* máximo. Este comportamiento es posible explicar por el mecanismo de recolección de información implementado para generar las gráficas. Como fue mencionado anteriormente, al momento de recolectar dicha información, se almacenan los valores de *power* y *rate* asociados al estado en el que se encuentra el algoritmo, sin embargo, al momento de tomar las decisiones, son utilizados los

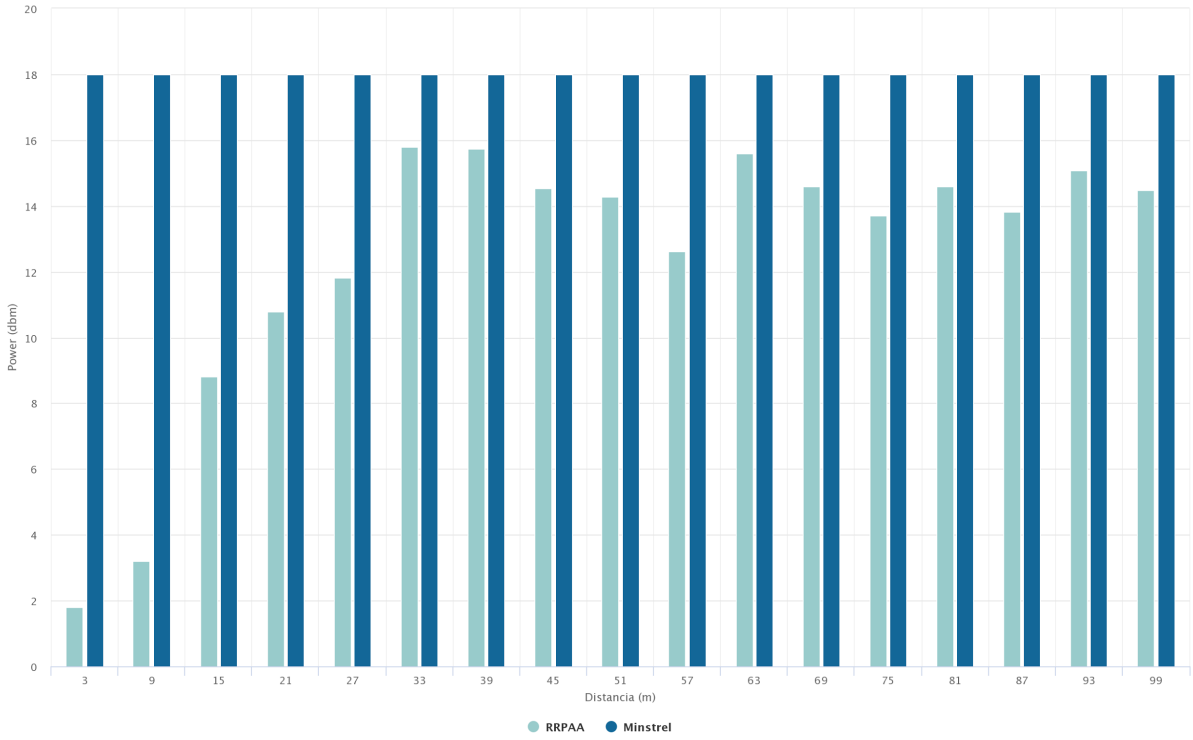

Figura 5.5: Power, en este gráfico se presenta, agrupado por distancia, el promedio del power obtenido de todas las ejecuciones.

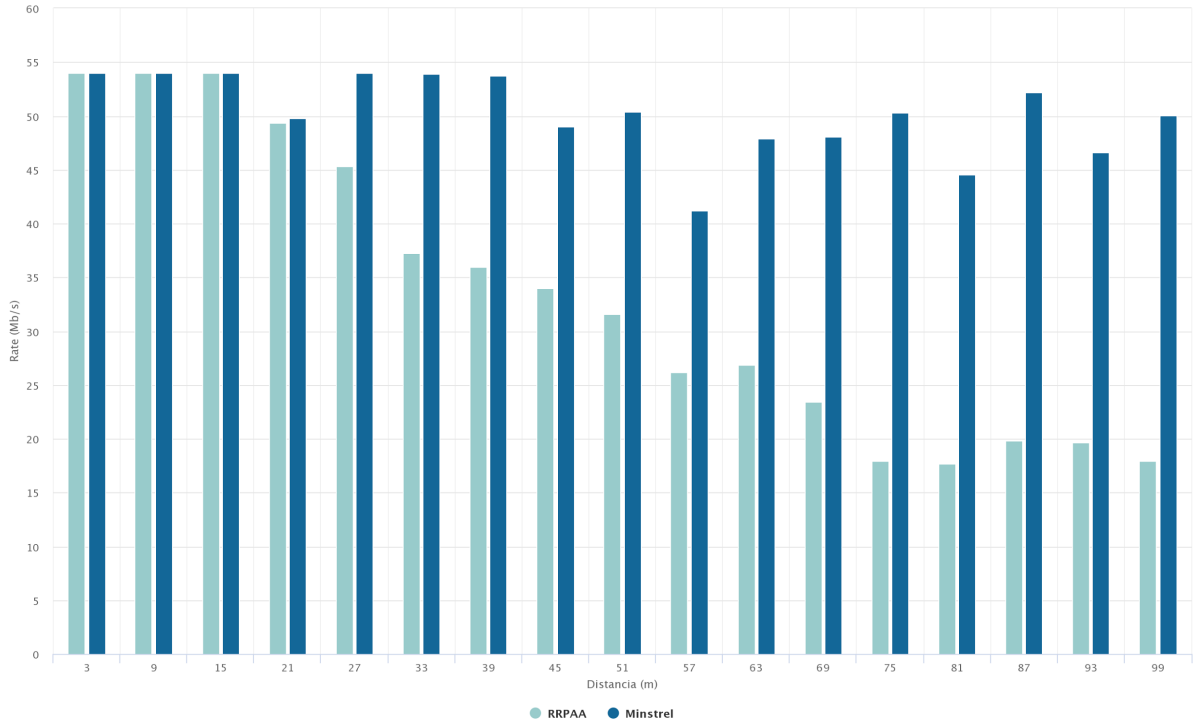

Figura 5.6: Rate, en este gráfico se presenta, agrupado por distancia, el promedio del rate obtenido de todas las ejecuciones.

valores con los que fueron enviadas las tramas. Teniendo en cuenta esto, es posible que se logeen valores de *power* menores a los finalmente utilizados.

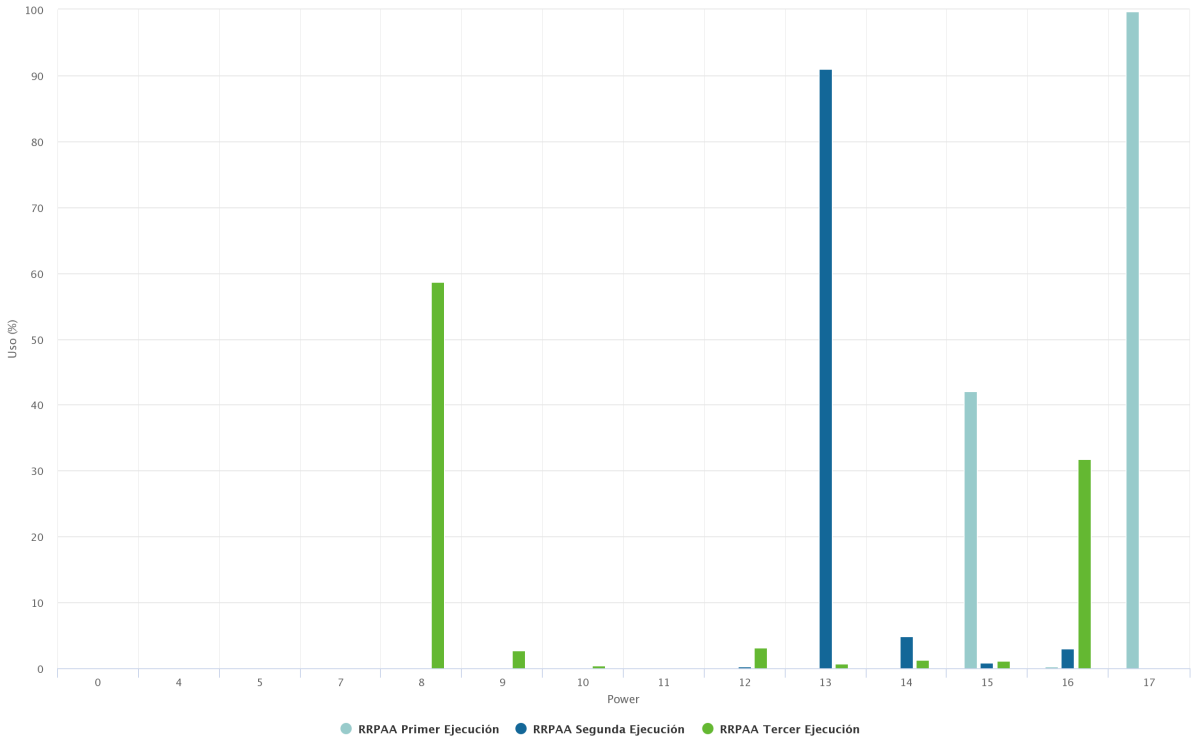

Figura 5.7: Histograma de los valores de power utilizados a 93m.

Como se puede observar en el gráfico de la figura 5.9, a pesar de que el *throughput* de *RRPAA* es significativamente menor a distancias superiores a 21m en comparación con *Minstrel*, el porcentaje de pérdidas de *RRPAA* es inferior al de *Minstrel*. Esto puede explicarse por el hecho que *RRPAA* utiliza rates significativamente inferiores a los utilizados por *Minstrel*, enviando menos información por unidad de tiempo pero con una fiabilidad superior. Con el fin de mejorar este tipo de escenarios, en la sección 6.2 se plantea estudiar la posibilidad de incluir el *throughput* generado en la decisiones de aumentar o disminuir el *rate*.

Si se analiza los reintentos que fueron necesarios a lo largo de las distintas ejecuciones de las pruebas (figura 5.10), se observa que a una distancia de 75m es cuando se obtiene un mayor porcentaje de pérdidas, por ende una cantidad de intentos mayor a las restantes distancias. En la figura 5.10, es posible observar el porcentaje de tramas enviadas exitosamente luego de una cierta cantidad de reintentos (sin tener en cuenta las tramas enviadas correctamente al primer intento). Este gráfico muestra que, de los paquetes que no fue posible enviar en el primer intento, un porcentaje cercano al 45 % se logra enviar en el segundo intento, seguido de un 30 % para los

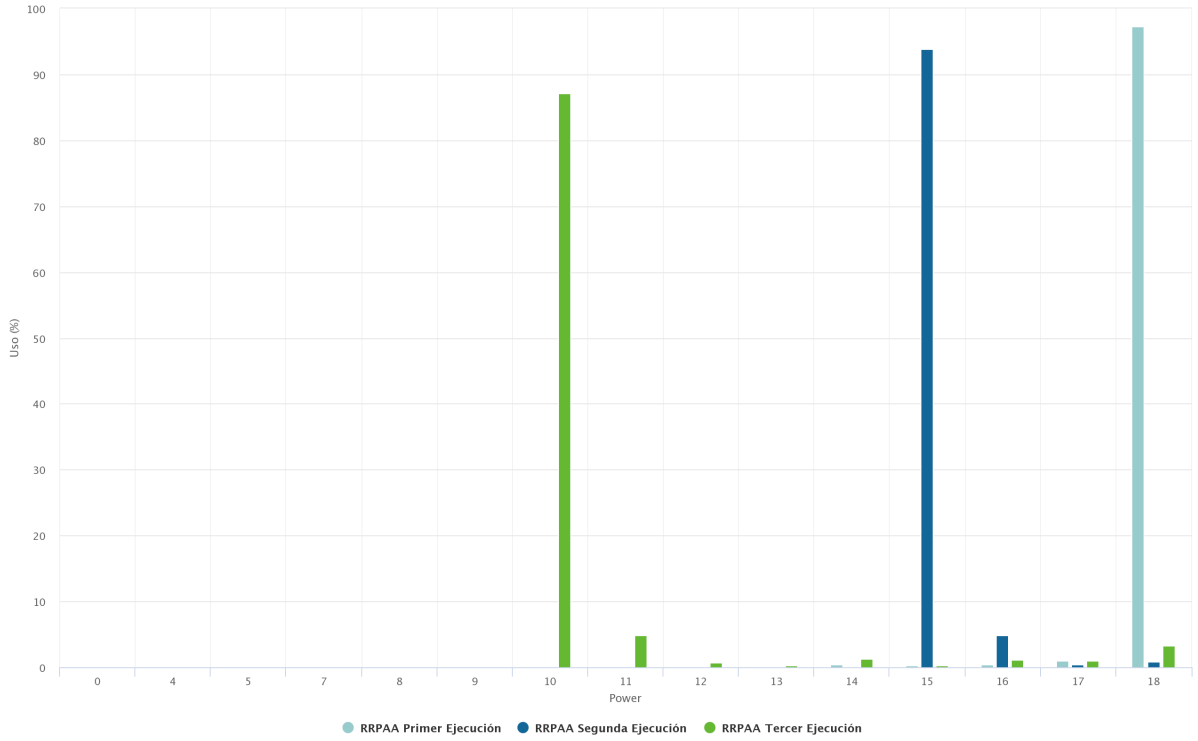

Figura 5.8: Histograma de los valores de power utilizados a 99m.

paquetes que fue necesario realizar un tercer intento. Por otro lado, es posible visualizar en la gráfica, mediante las líneas verticales, cada sección de la *retry chain* utilizado en RRPAA. Esto último, sumado al comportamiento anteriormente mencionado, permite observar el beneficio de utilizar el mecanismo de *multi-rate retry*, debido a que para la totalidad de las tramas enviadas, no resulta necesario realizar un procesamiento de las estadísticas ni ejecución del algoritmo de *rate control*, evitando el *overhead* que implicaría la ejecución del mismo. En términos cuantitativos se tiene que del total de las tramas enviadas, un porcentaje cercano al 87 % fueron enviadas exitosamente en el primer intento, por lo que un porcentaje cercano al 13 % del total genera retransmisiones. De estas retransmisiones, un porcentaje superior al 90 % son logrados retransmitir con los mismos valores de *rate y power* que fueron enviados el primer intento.

Cabe mencionar que a pesar de no ser ejecutado el algoritmo de control de *power* y *rate* para cada intento, las estadísticas generadas por estos intentos son reportadas luego de realizar un envío exitoso o hasta que se consume completamente el *retry chain*. De esta forma se permite procesar las estadísticas de todos los intentos como si se hubiera aplicado el algoritmo de forma individual, permitiendo mantener el correcto mantenimiento estadístico para el control de *rate* y *power*, evitando el

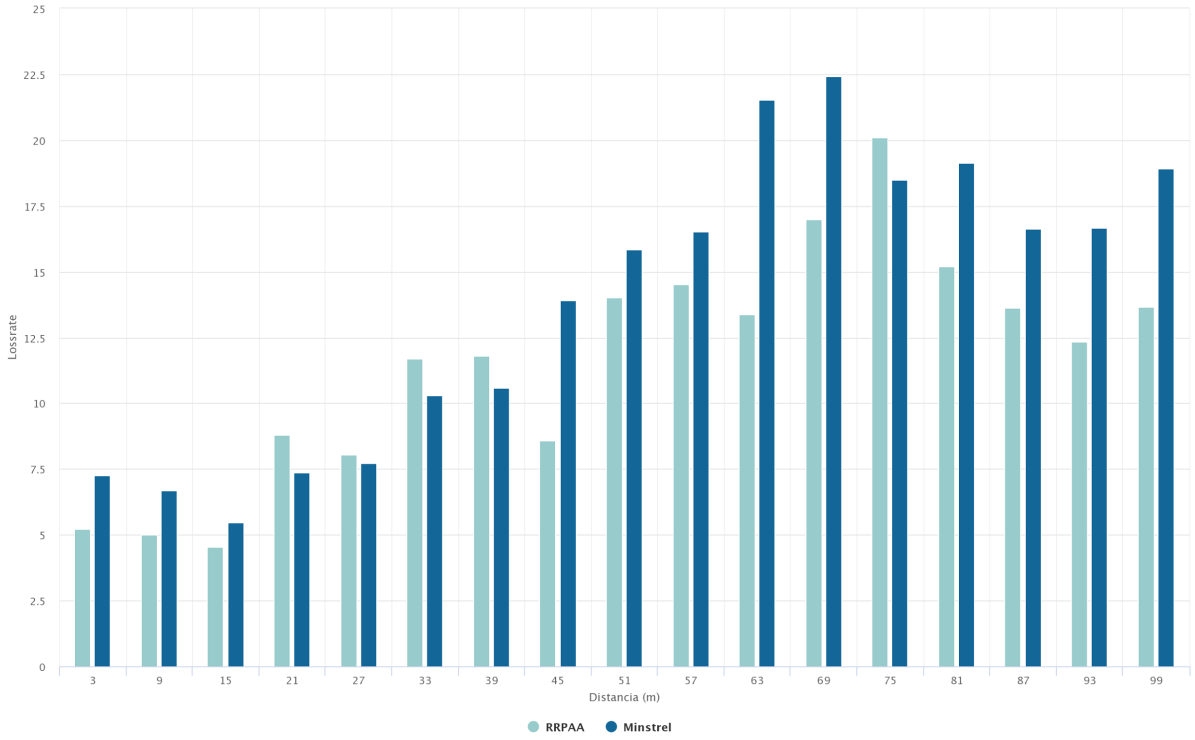

Figura 5.9: Loss-rate, en este gráfico se presenta, agrupado por distancia, el promedio del loss-rate de todas las ejecuciones.

overhead que implicaría la ejecución de algoritmo asociado.

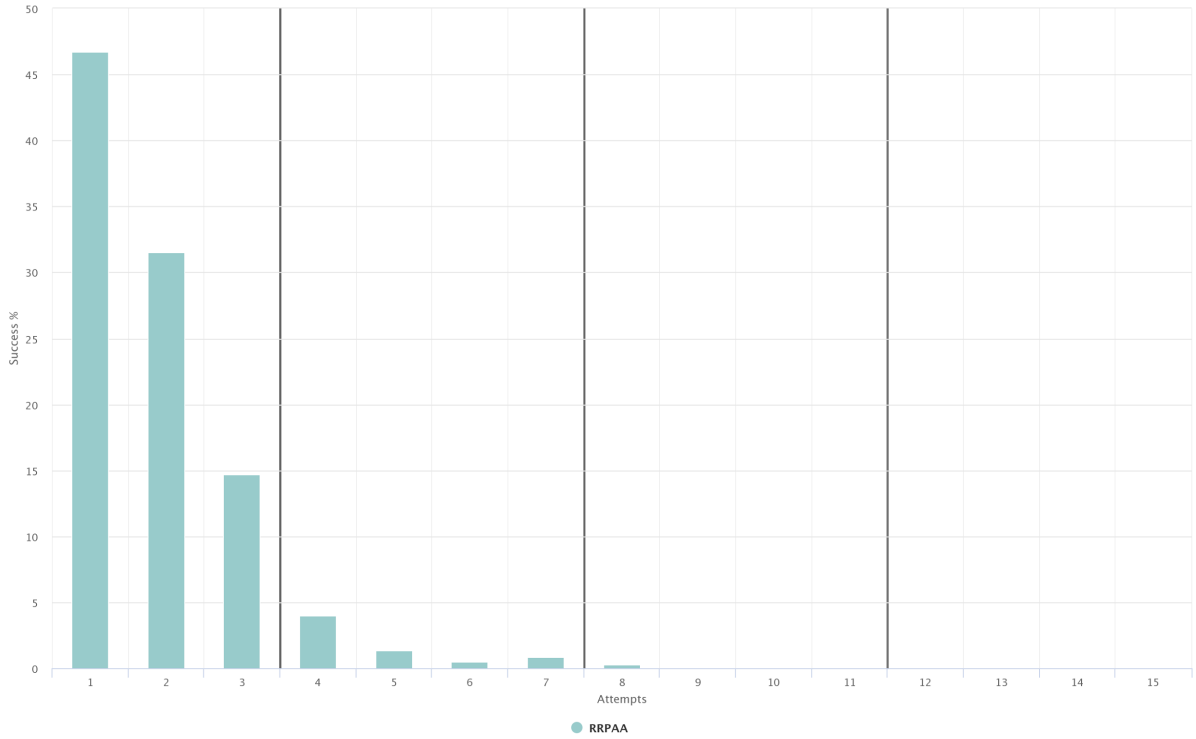

Figura 5.10: Porcentaje de reintentos exitosos, para una distancia de 75m, según attempts.

Finalmente resulta interesante considerar el consumo final de energía para ambos algoritmos. En este escenario, luego de agrupar todos los registros con el fin de considerar las pruebas de forma global, es posible constatar un ahorro cercano al 73 % de energía de *RRPAA* frente a Minstrel. En este escenario cabe recordar que *Minstrel* obtiene un *throughput* de hasta un 50 % superior y que este resultado se relaciona al bajo rate utilizado. Teniendo en cuenta esto, podría ser interesante que *RRPAA* utilice un *power* mayor, buscando disminuir las pérdidas, permitiendo eventualmente aumentar el *rate* empleado.

5.4.2. *Overlapping*

En este escenario se busca analizar el comportamiento de RRPAA en escenarios en donde se encuentran varios terminales, asociados cada uno a un *router* distinto, como puede suceder en ambientes densos. Este escenario se divide en los siguientes casos:

1. En un primer caso, se colocan las terminales de forma tal que no es

posible aislar las transmisiones.

2. En un segundo caso, los terminales con su *router* asociado se encuentran a una distancia tal que es posible aislar las transmisiones.

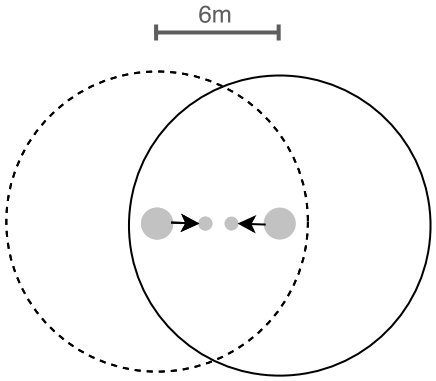

Figura 5.11: Diagrama del caso en donde no es posible aislar las transmisiones.

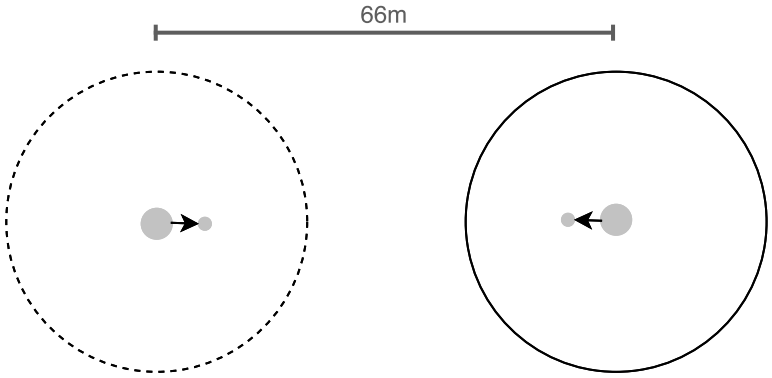

Figura 5.12: Diagrama del caso en donde es posible aislar las transmisiones.

Para conseguir los casos planteados se colocan los terminales con sus *routers* a varias distancias, en particular se decidió por tomar los siguientes valores:

- 6 metros
- 18 metros
- 36 metros
- 66 metros

Separando los terminales de su respectivo router 60cm. Adicionalmente se define un escenario similar, el cual cuenta con la particularidad de no incluir un terminal

generando interferencias. Dicho escenario, se denomina escenario base, el cual es tenido en cuenta como un escenario ideal de ejecución de los algoritmos.

5.4.2.1. Resultado esperado

En estos escenarios se espera que a una distancia inicial, tanto en RRPAA como en *Minstrel*, las transmisiones se encuentren superpuestas, pero a medida que la distancia aumenta se espera que las transmisiones mejoren, disminuyendo la interferencia entre sí. Esto es debido a que aún con *power* mínimo, las transmisiones siguen generando interferencias a cortas distancias.

Se espera que al aumentar la distancia, *RRPAA* mejore su rendimiento a una menor distancia que *Minstrel*, debido a que éste último utiliza *power* máximo.

Para detectar los casos en donde las transmisiones se desacoplan se tomará como medida el *loss-rate* obtenido en cada router, el cual debe disminuir una vez que se desacoplan las transmisiones.

5.4.2.2. Resultado obtenido

Como es posible observar en la figura 5.13 los resultados obtenidos coinciden con los resultados esperados. En particular es posible ver que *RRPAA*, a una distancia de 18m, obtiene un *throughput* 12 % superior al obtenido con *Minstrel*. Esta mejora es incrementada al alejar los nodos a 36m hasta alcanzar un 20 % y finalmente la mejora desciende a un 10 % a 66m. Este descenso de la mejora es explicable por la mejora de *Minstrel* al encontrarse a distancias tales que la interferencia entre las transmisiones disminuye.

En cuanto a los escenarios bases es posible observar un *throughput* medio de 2952 B/s para *Minstrel y* 2976 B/s *para RRPAA*. Sin embargo al incorporar interferencias a ambos protocolos, es posible obtener valores de throughput entre 1591 B/s y 1968 B/s para *RRPAA*, mientras que para Minstrel estos valores se encuentran entre 1488 *B/s* y 1768 *B/s*. Esto implica una disminución porcentual entre 45 % y 30 % del *throughput* base para *RRPAA* a medida que se aumenta la distancia y entre 50 % y 40 % para *Minstrel*. Teniendo en cuenta estos valores, es posible observar nuevamente el efecto de la disminución de la interferencia debido al aumento del *throughput*, a pesar de encontrarse aún presente.

Nuevamente en la figura 5.14 es posible observar el efecto que tiene la disminución del *power* en la separación de las transmisiones, observando la disminución del *loss-rate* de *RRPAA* frente a *Minstrel*. Al disminuir el *power*, la interferencia entre los nodos disminuye, lo que genera una tasa de pérdidas menor. Dicho efecto en una etapa inicial es inexistente debido a la cercanía que existe entre los nodos, a medida que se

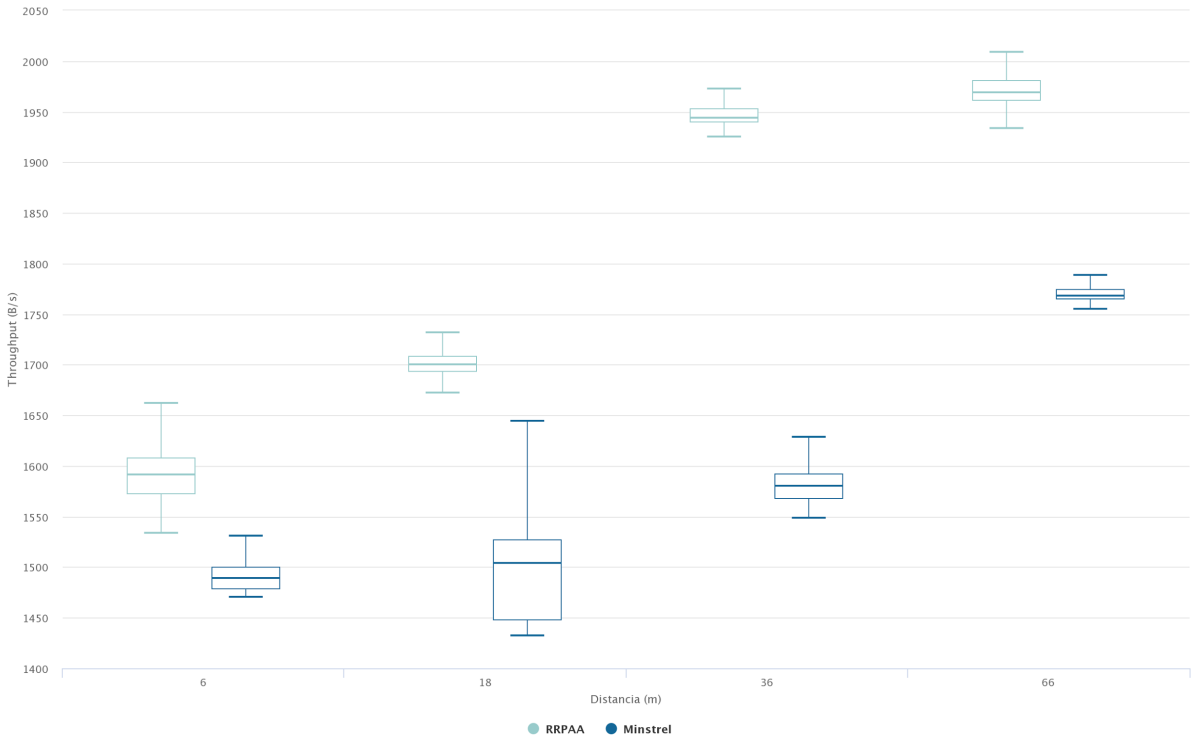

Figura 5.13: Throughput, en este gráfico se presenta, para una distancia fija, el throughput medio obtenido de todas las ejecuciones.

alejan, el efecto de la separación entre los nodos se torna visible hasta alcanzar una distancia tal que ambas transmisiones tienden a desacoplarse aún a máximo *power*.

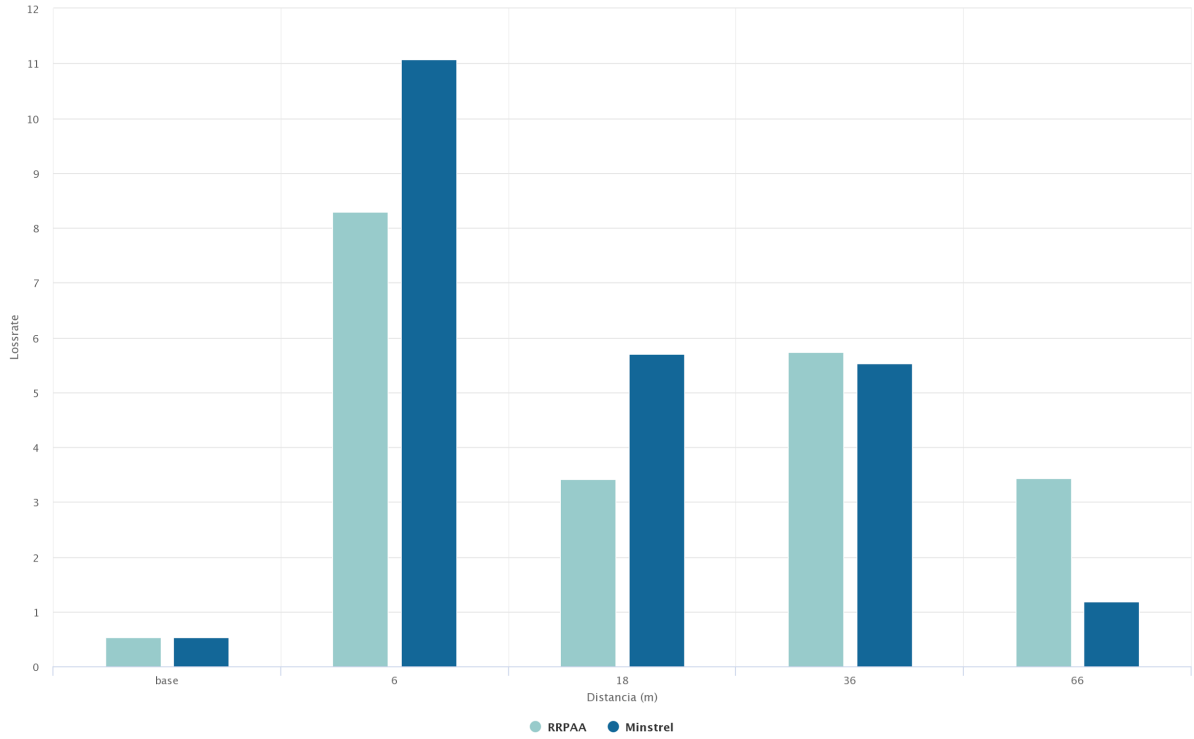

Figura 5.14: Loss-rate, en este gráfico se presenta, agrupado por distancia, el promedio del loss-rate obtenido de todas las ejecuciones.

De forma similar, en el gráfico de la figura 5.15 es posible ver la mejora de *RRPAA* en el manejo del ambiente debido a la disminución del *busy ratio* obtenido con éste. En particular es posible observar que el *busy ratio* que obtiene *Minstrel* se mantiene prácticamente constante, mientras que con *RRPAA* se obtiene una disminución considerable a partir de los 36m, siendo esta mejora cercana a un 10 % en comparación con *Minstrel*.

Relacionado al *busy ratio*, es posible observar en la figura 5.16 un aumento de la cantidad de intentos, asociado a una mayor oportunidad de envío. Cómo es posible observar en las figuras 5.17 y 5.14, *RRPAA* emplea un *rate* similar, sufre de un *loss-rate* similar a *Minstrel* y los paquetes enviados son del mismo tamaño (de lo contrario podría explicarse un aumento en el tiempo de transferencia o recepción). Por lo mencionado anteriormente y la baja del *busy ratio*, el terminal cuenta con una oportunidad de envío superior, lo que conlleva a una mayor cantidad de envíos de tramas.

Si se compara el *throughput* de ambos algoritmos con respecto al *throughput* del

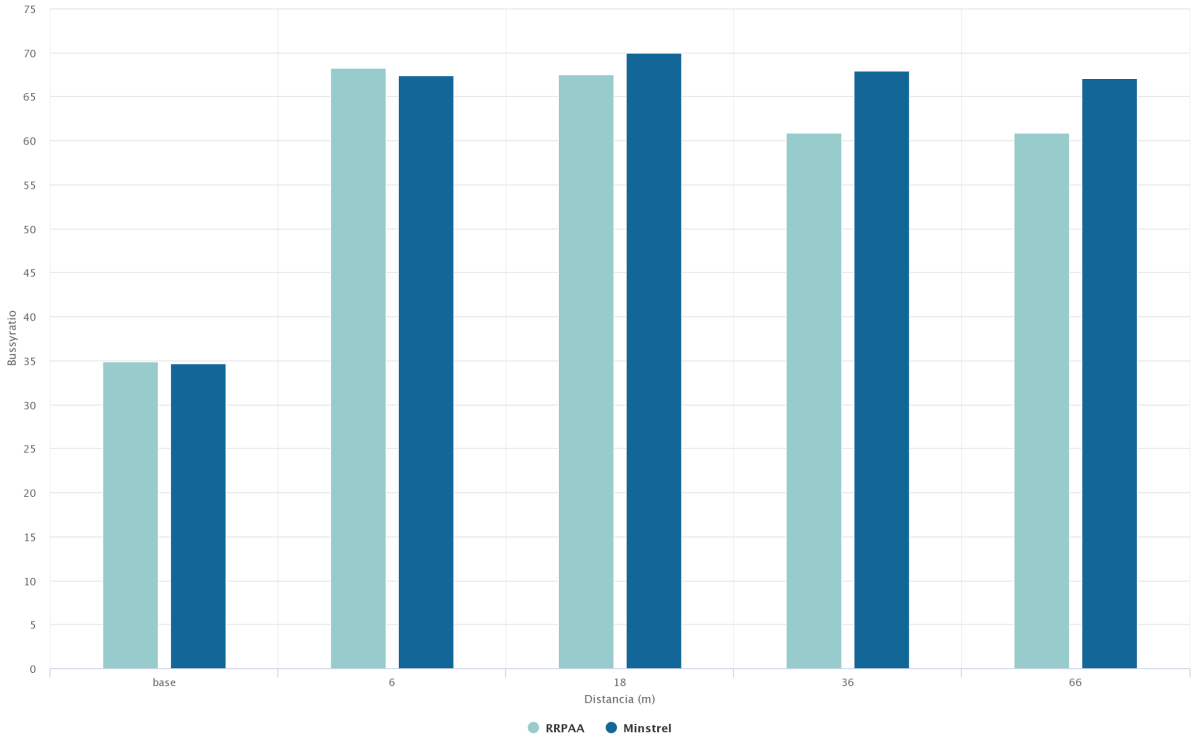

Figura 5.15: Busy ratio, en este gráfico se presenta, agrupado por distancia, el busy ratio obtenido de todas las ejecuciones.

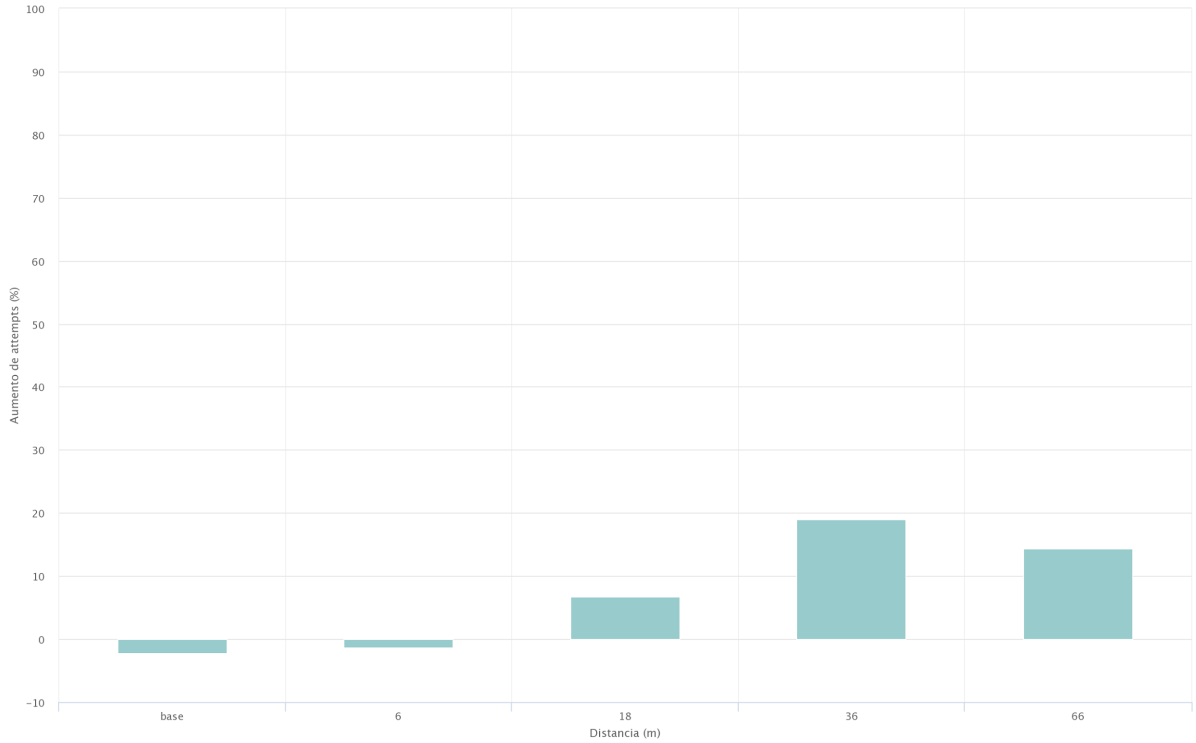

Figura 5.16: Aumento porcentual de attemps entre RRPAA y Minstrel.

escenario base, es posible observar que en ambos casos se obtiene un deterioro considerable (cercano al 35 %) inclusive a una distancia superior a los 60m. Esto indica que la implementación generada no logra aislar completamente las transmisiones a esta distancia pero logra una mejora considerable con respecto a *Minstrel*.

Si se comparan las pérdidas de ambos algoritmos, no es posible observar una diferencia significativa en las distintas ejecuciones. Si se considera la potencia utilizada, es posible observar una menor potencia generalizada para la implementación de *RRPAA* (Figura 5.18). Esto junto a una mayor distancia al AP, implica que los terminales más lejanos perciben una disminución de la interferencia debido a que la cantidad de paquetes enviados no aumenta (debido a un rate y *loss-rate* similares).

Ambos casos planteados anteriormente permiten observar una mejora en el manejo del ambiente, entendiendo dicha mejora como una menor interferencia enviada a los demás terminales.

Finalmente, si se compara la variación porcentual de energía entre ambos protocolos (figura 5.19), es posible observar que *RRPAA* consume un porcentaje de energía significativamente menor en comparación con *Minstrel*. Cabe mencionar que a pesar de mantener valores similares de *rate* y *loss-rate* (esto en principio permitiría mantener una cantidad de paquetes enviados similar) se obtiene un aumento en el

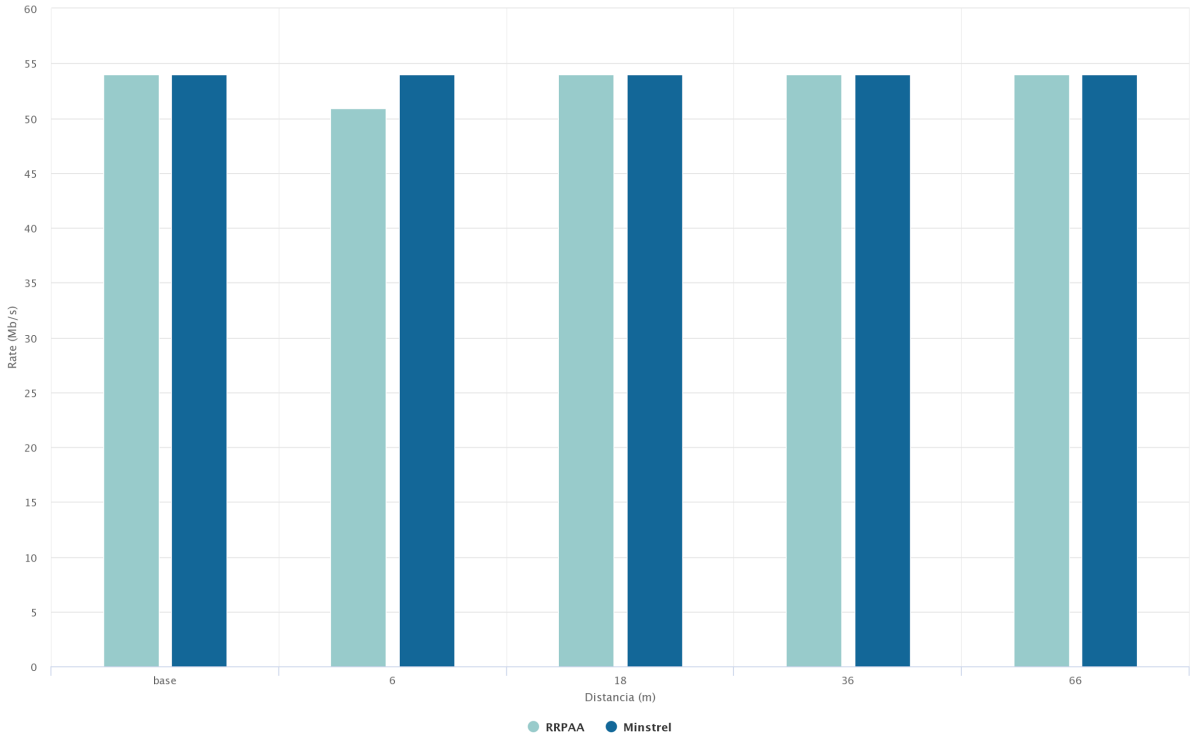

Figura 5.17: Rate, en este gráfico se presenta, agrupado por distancia, el promedio del rate obtenido de todas las ejecuciones.

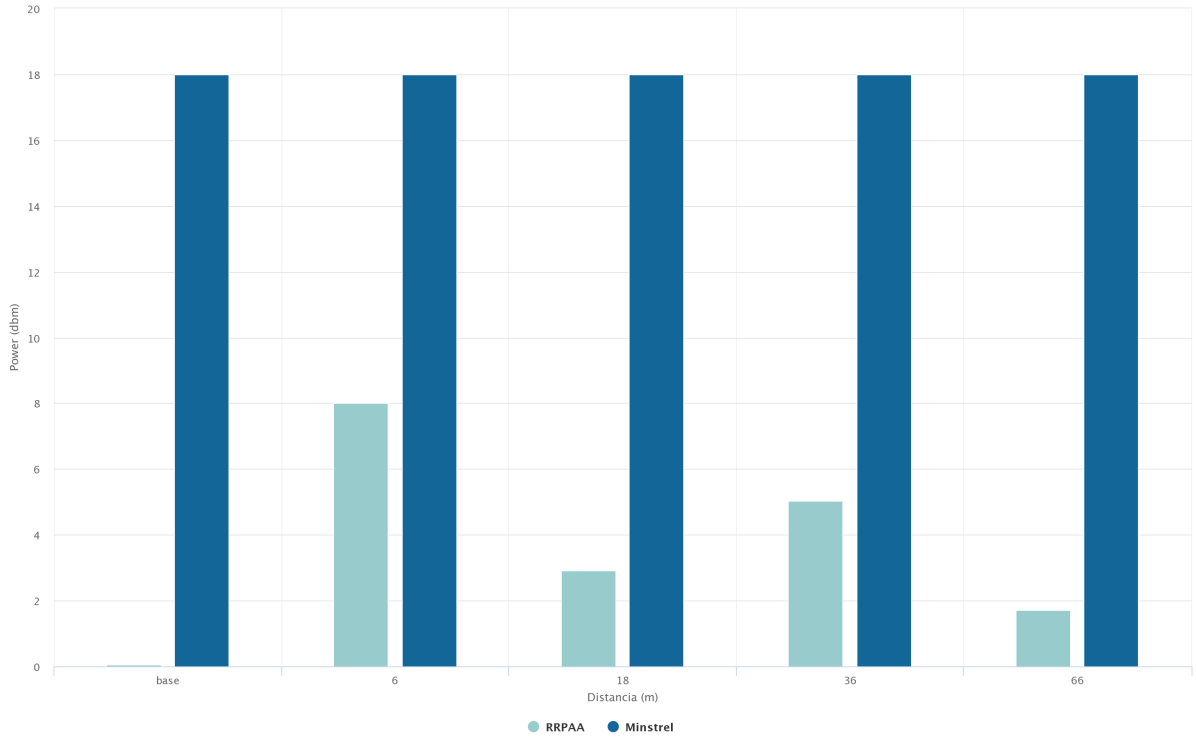

Figura 5.18: Power, en este gráfico se presenta, agrupado por distancia, el promedio del power obtenido de todas las ejecuciones.

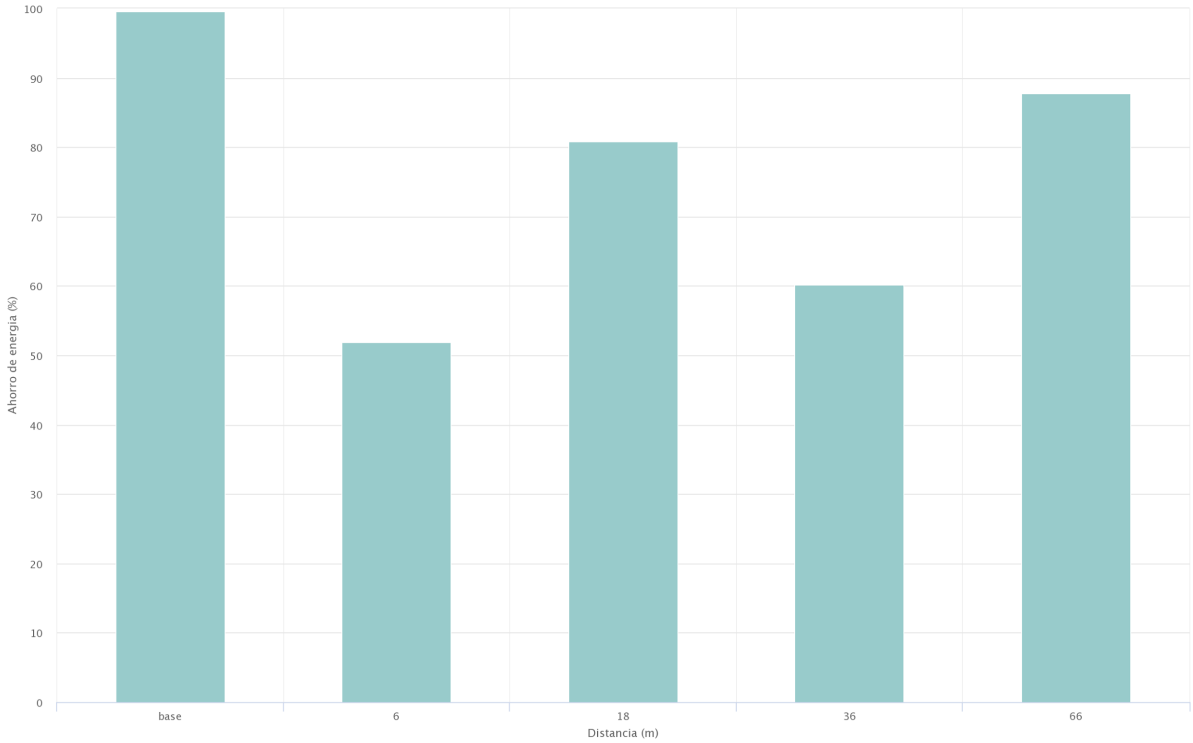

Figura 5.19: Ahorro porcentual de energía de RRPAA frente a Minstrel.

throughput, como fue mencionado anteriormente. Esto es posible explicarlo debido a la disminución del *busy ratio*, lo que permite una mayor oportunidad de transmisión y una cantidad de tramas enviadas.

5.4.3. Múltiples *Links*

En este escenario se busca analizar el comportamiento de *RRPAA* en presencia de varios clientes a distintas distancias, como sería un escenario cotidiano de ejecución de un AP. Para esto se coloca un *router* transmitiendo a dos terminales, la primera de ellas a 12 metros y la restante a 36 metros.

5.4.3.1. Resultado esperado

En este caso se espera que los parámetros de *power* y *rate* varíen según el terminal y que el *throughput* obtenido por *RRPAA* sea similar o superior al obtenido con *Minstrel*

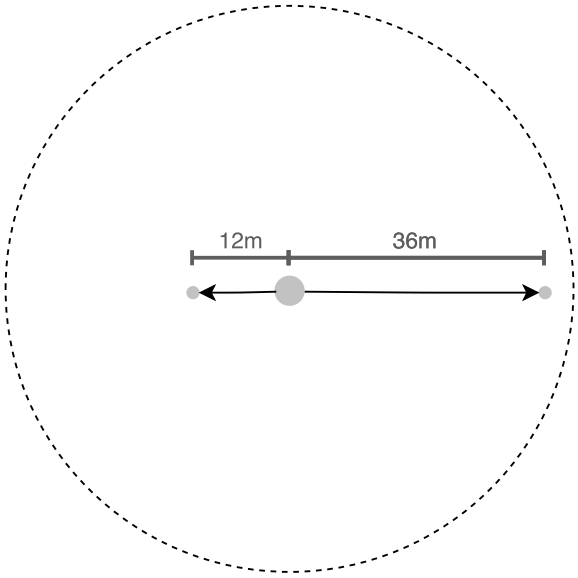

Figura 5.20: Diagrama del escenario Múltiples *Links*.

5.4.3.2. Resultados obtenidos

Como se aprecia en la figura 5.21, el algoritmo de *Minstrel* tiene un rendimiento levemente superior en throughput que *RRPAA*. Este comportamiento se condice con el aumento de pérdida que se da al disminuir el *power*, no obstante dicha variación es ínfima, cercana a un %3.

En cuanto al rate utilizado, como es posible observar en la figura 5.22, ambos algoritmos utilizan valores similares.

En cuanto al *power* utilizado, como se aprecia en la figura 5.23, en ambos casos *RRPAA* utiliza un *power* muy inferior al utilizado por *Minstrel*. El terminal de menor distancia utiliza un *power* mínimo, mientras que el restante utiliza un power mayor, pero aún así se obtiene una disminución de *power* superior al 75 % en comparación con *Minstrel*.

Como resultado favorable de estas ejecuciones se tiene que *RRPAA* utiliza *power* mínimo en la mayor parte del tiempo, sin disminuir considerablemente el *throughput*.

Si se compara la variación porcentual de la energía entre los distintos protocolos (figura 5.24) se obtiene que *RRPAA* utiliza un porcentaje sumamente inferior de energía en comparación con *Minstrel*. Como se mencionó anteriormente, el *throughput* generado por ambos protocolos es similar, por lo que *RRPAA* logra un resultado similar al de *Minstrel* con la ventaja adicional de un ahorro de energía superior al 75 %.

Como conclusión de este escenario es posible observar que el control del algorit-

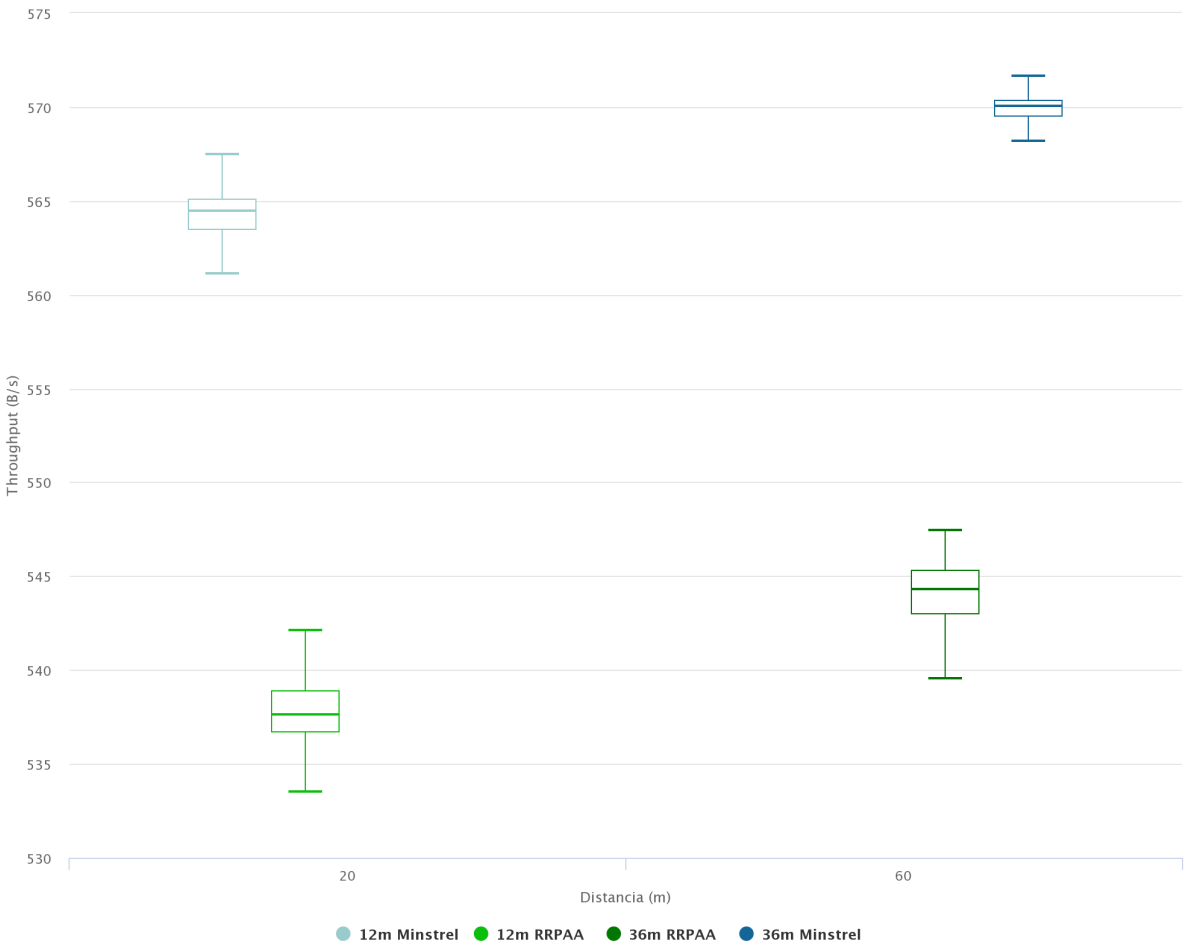

Figura 5.21: Throughput, en este gráfico se presenta, agrupado por distancia, el throughput medio obtenido de todas las ejecuciones.

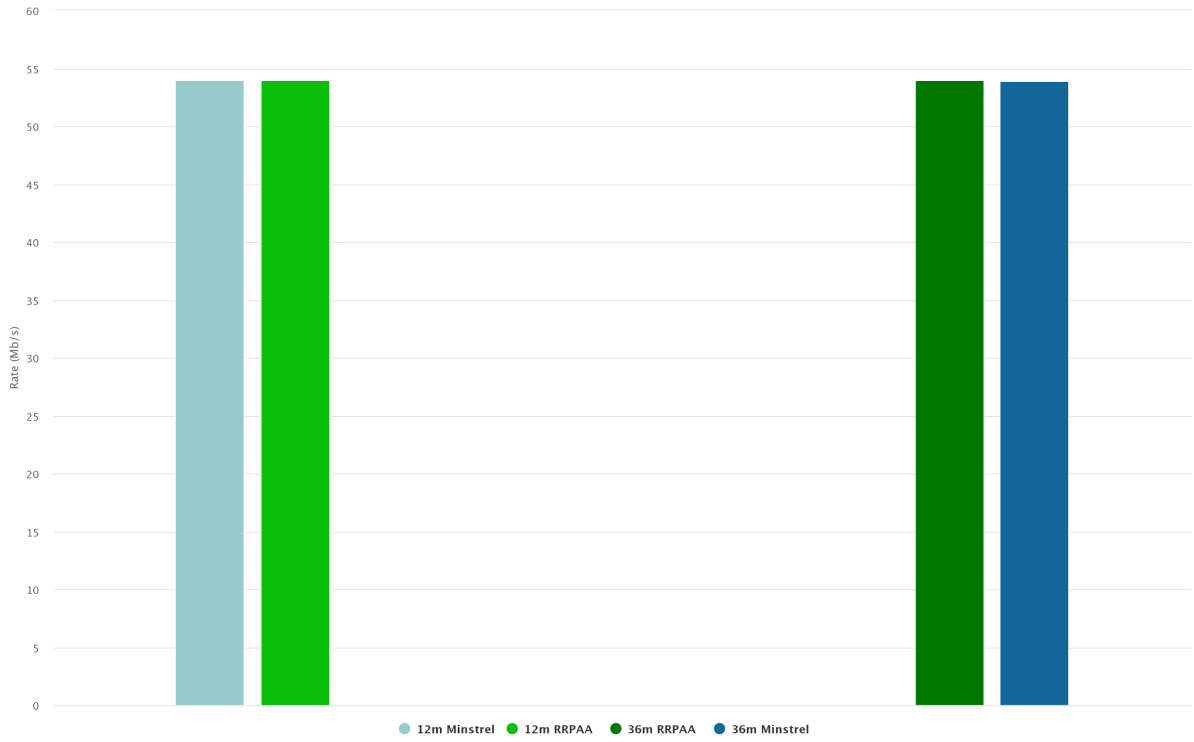

Figura 5.22: Rate, en este gráfico se presenta, agrupado por distancia, el promedio del rate obtenido de todas las ejecuciones.

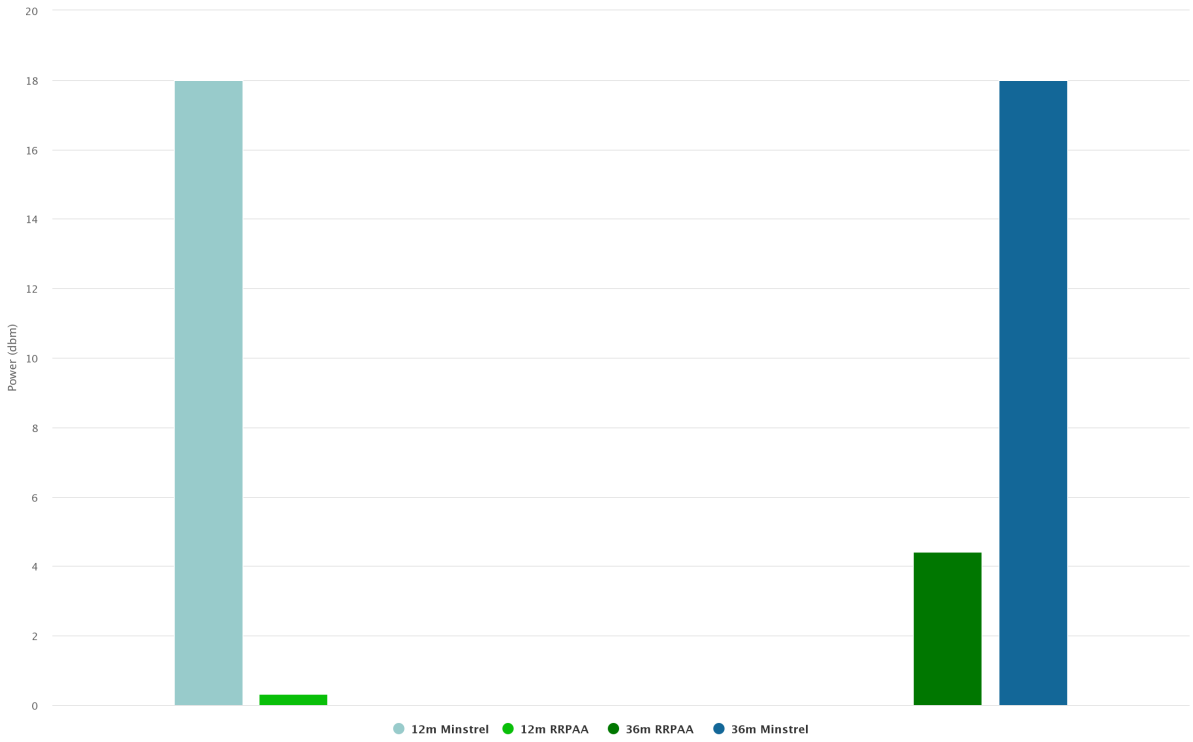

Figura 5.23: Power, en este gráfico se presenta, agrupado por distancia, el promedio del power obtenido de todas las ejecuciones.

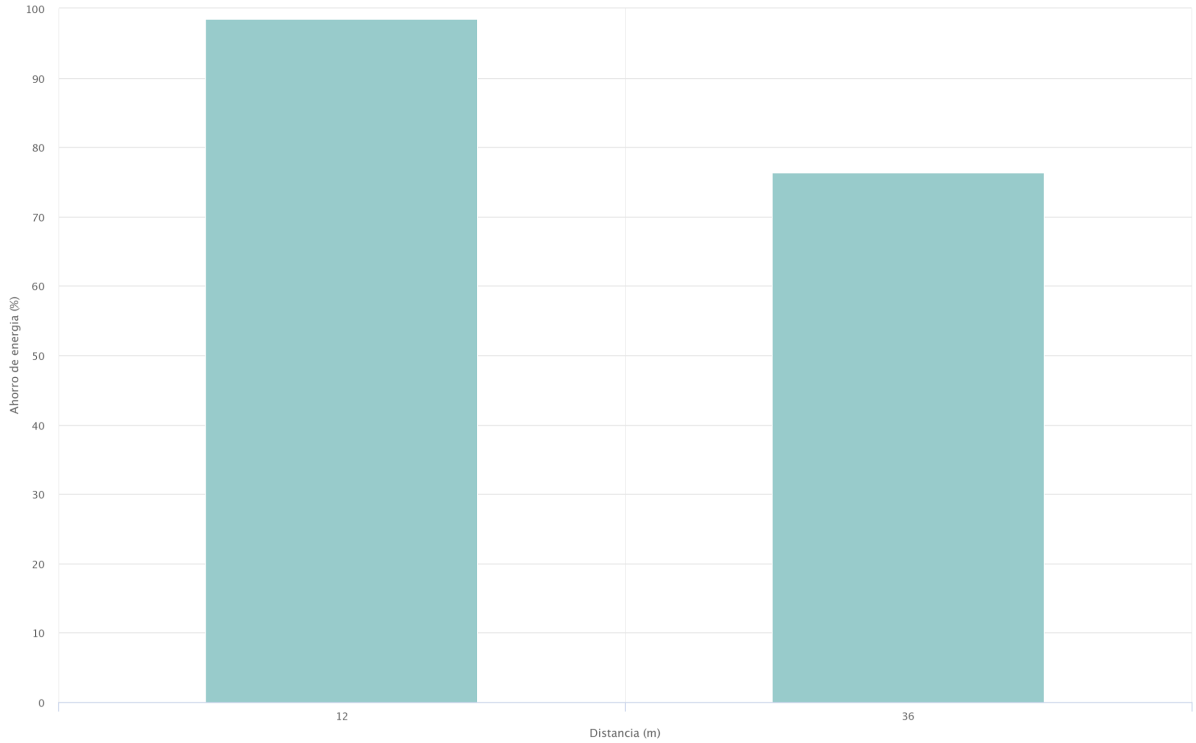

Figura 5.24: Ahorro porcentual de energía de RRPAA frente a Minstrel.

mo es aplicado a cada enlace virtual del *router* con los respectivos nodos de forma independiente.

5.5. Problemas encontrados

Uno de los principales problemas encontrados en la realización de las pruebas fue la dificultad de encontrar disponible un ambiente libre de interferencias y con distancias tales que permita aislar las transmisiones realizadas. Para esto se obtuvo acceso a las instalaciones del InCo y fue posible dar de baja algunas redes inalámbricas, dejando libre los canales superiores, evitando de esta forma interferencias con redes inalámbricas. Por otro lado, las pruebas fueron realizadas en momentos de poca concurrencia de personas en el InCo, principalmente fines de semanas, buscando minimizar las interferencias que éstas pudieran provocar.

Otro problema encontrado es que, a pesar de que el escenario de prueba fuera el mismo, ambos algoritmos presentan un comportamiento diferente con la variación del ambiente, pudiendo llegar a tener un deterioro del 80 %, aun en escenarios sin interferencias generadas y en canales aislados. Buscando mitigar este inconveniente, las pruebas fueron llevadas a cabo utilizando intervalos mínimos de tiempo e inter-

calando las distintas ejecuciones de las pruebas entre las distintas implementaciones, buscando que ante estos cambios, el impacto sea el menor posible.

Capítulo 6

Conclusiones y trabajo a futuro

6.1. Conclusiones

En el presente trabajo se buscó plasmar el proceso realizado para la implementación de un algoritmo de control de *rate* y *power*, incluyendo las tareas de investigación tanto del ambiente en el que debe convivir la implementación, como el conjunto de herramientas necesarias para su desarrollo. Dicha implementación cuenta con un rendimiento tal que es posible tomarlo como una alternativa real a los protocolos por defecto que tienen los routers hoy en día.

Dicho algoritmo cuenta con algunas características que lo hacen interesante, en particular respeta el protocolo *IEEE 802.11*, por lo que puede ser utilizado en los APs sin la necesidad de alterar ninguno de los terminales que se conectan a éstos.

Si bien en el marco del presente proyecto se obtiene un prototipo completamente funcional de *RRPAA*, por cuestiones de tiempo, no fue posible incluir algunas de las pruebas planteadas a futuro, en particular, realizar pruebas más exhaustivas, refinar los valores de los parámetros. Sin embargo las pruebas realizadas muestran un prometedor algoritmo, que si se sigue trabajando puede ser presentado como alternativa real a *Minstrel*.

Cabe destacar que el desarrollo de este tipo de *software* cuenta con muchos retos, en particular la escasa documentación sobre el *kernel* y herramientas como *mac80211*, por lo que resulta necesario basar el desarrollo en implementaciones ya existentes para poder realizar el prototipo. En cuanto a la depuración del código, no se cuenta con herramientas como *debuggers* que indiquen posiciones de excepciones o errores en el manejo de memoria, por esto, las pruebas y depuración del código insumen un tiempo mayor, teniendo que restaurar el sistema operativo ante ciertos errores, debido a que principalmente se trabaja dentro del *kernel* del mismo.

En cuanto al código desarrollado, se trabajó en base al protocolo planteado por

Matías Richart[1], teniendo la necesidad de tomar decisiones exclusivas del desarrollo del *software*, como puede ser la forma de manejar la variable aleatoria o el control de la *retry chain* y restricciones propias del *kernel* de *Linux* como puede ser, no contar con operaciones de punto flotante u optimizaciones a nivel de código y memoria. Dichas decisiones deben ser sometidas a evaluación pudiendo ser una fuente de mejora al rendimiento del mismo.

Finalmente cabe destacar la obtención final de una implementación funcional de *RRPAA*, junto a la evaluación del mismo en un ambiente real y frente a un algoritmo estable y ampliamente utilizado, debiendo enfrentarse a las variaciones y demás contraposiciones que agrega la ejecución en un ambiente real, dejando de lado las simplificaciones que implican las simulaciones.

6.2. Mejoras a futuro

Si bien en este proyecto se logró obtener un prototipo funcional del protocolo *RRPAA* se plantean algunas mejoras para continuar el desarrollo, con el fin de alcanzar una versión final, estable y fiel al protocolo *PRCS*.

- Como uno de los *ítems* más importante se encuentra la inclusión del control del *Carrier Sense* al prototipo desarrollado. Como fue mencionado en la sección 4.5.4, para obtener esto, se tendría que trabajar en conseguir las cartillas de registros de *Atheros*, para luego evaluar si existe algún registro que cumpla los requerimientos buscados.

- Si bien este prototipo desarrollado sirve para todas las revisiones de *IEEE 802.11*, fue desarrollado y probado sobre la base de *Minstrel* legado. *Minstrel* incluye optimizaciones para los dispositivos que trabajan a 5Ghz, especialmente para los que utilizan la norma *IEEE 802.11n*, los cuales utilizan la implementación con el nombre *Minstrel_ht*. Como mejora se propone evaluar las optimizaciones realizadas e incluirlas en caso que aporten valor.

- Como fue mencionado en la seccion 4.6, el proyecto fue implementado utilizando la distribución *OpenWRT*. Actualmente, algunos desarrolladores de dicho proyecto crearon una nueva distribución llamada *Lede*[26], la cual propone ser más abierta a los cambios. Podría resultar interesante evaluar la utilización de esta versión y proponer *RRPAA* como una alternativa a *Minstrel*.

- En cuanto al código desarrollado, se plantea un refinamiento del mismo, siguiendo buenas prácticas de desarrollo en C y de módulos del *kernel* de *Linux*, así como el desacoplamiento a la implementación de *Minstrel*.

- Relacionado al punto anterior, resulta necesario realizar una evaluación de la aproximación empleada para el manejo de las probabilidades, en particular, se debería asegurar que estados con baja probabilidad puedan volver a ser utilizados con el paso del tiempo.

- En cuanto a los parámetros del algoritmo se plantea evaluar los valores de los parámetros (α, β, γ, δ y τ) del protocolo, con el fin de modificar la tabla de *threshold*, buscando generar un conjunto de valores para distintos escenarios.

- Similar al punto anterior, se propone evaluar el mecanismo de selección de rate con el fin de incluir el valor de *throughput* registrado por el nodo. El cometido de esta propuesta es mitigar el bajo *throughput* observado en el escenario de múltiples distancias 5.4.3.

- Evaluar la decisión tomada con respecto a la optimización realizada al incluir el uso de la *retry chain*. Se podría evaluar la utilización de una cantidad de reintentos variable dependiendo del estado del algoritmo, ajustar la cantidad máxima de intentos para cada *rate*, o evaluar alternativas a la carga de dicha estructura.

- Incluir en la implementación la posibilidad de utilizar un conjunto de *powers* dinámicos, dependiendo de las restricciones asociadas al país en donde será utilizado dicha implementación. Para esto, el *driver* tendría que comunicarle a *mac80211*, los *powers* disponibles al momento de la inicialización.

- La utilización de un sistema de recolección de información con los valores reales de *rate* y *power*, es decir sin importar el estado actual del protocolo.

- Incluir la posibilidad de habilitar o deshabilitar la recolección de datos estadísticos en tiempo de ejecución, para un conjunto de terminales conectados al *AP*, utilizando el sistema de archivos virtual, para la configuración de esta funcionalidad.

6.3. Pruebas

Se entienden que las pruebas realizadas sirven para comprobar el correcto funcionamiento del algoritmo, no obstante se proponen un conjunto de pruebas a futuro que resultan interesantes para la mejora de éste.

6.3.1. Pruebas de estrés

Las pruebas realizadas en el marco del presente trabajo fueron realizadas en un ambiente en el cual el *router* conectaba con a lo sumo dos clientes. En pruebas futuras se plantean llevarlas a cabo con un número superior de dispositivos conectados al mismo *router*. El objetivo de estas pruebas es comprobar el comportamiento de *RRPAA* al tener que manejar un gran número de estadísticas para tomar decisiones en un tiempo razonable y mantener un buen rendimiento de la red.

Teniendo en cuenta que cada *AP* puede soportar una cantidad cercana a los 25 clientes sin verse considerablemente degradado el rendimiento de la red[27], se podrían definir rangos de cantidad de clientes cercanos a este valor y enviar un tráfico similar entre éstos.

6.3.2. Pruebas de escenario real

Tomando en consideración que las pruebas realizadas fueron de forma controlada y alejadas a los escenarios normales de ejecución de este tipo de algoritmos, se plantea definir un conjunto de escenarios similares a los que se someten estos algoritmos al encontrarse en ambientes de producción.

Teniendo en cuenta esto y el trabajo realizado en [28] se plantean los siguientes escenarios:

- Un salón de clases de facultad con, por ejemplo 100 alumnos, en donde cada uno cuenta con un dispositivo conectado a la red de forma inalámbrica y no se cuentan con superficies como paredes que puedan deteriorar la conectividad.
- Un hogar de familia en donde cada integrante cuenta con al menos un dispositivo y en donde la señal debe traspasar obstáculos como pueden ser las paredes de las distintas habitaciones.

6.3.3. Pruebas de terminal expuesta y oculta

Cabe recordar que estos problemas mencionados, no son resueltos únicamente con la inclusión de control de potencia pudiendo hasta ser agravados, con el fin de

reducir su impacto resulta necesario incluir el control de carrier sense. A pesar de ésto y contemplando una futura implementación de *PRCS*, podría resultar interesante someter *RRPAA* a los escenarios planteados en [1] con el fin de generar un conjunto de resultados y evaluar el impacto de la inclusión del mecanismo de control de *carrier sense.*

Apéndice A

Anexos

A.1. Utilización del prototipo generado

El código generado para el prototipo de *RRPAA*, es posible obtenerlos en forma de parches que son acoplados al sistema de *OpenWRT*. Para la utilización de dicho código es necesario incluirlos al código fuente de dicho sistema y compilarlo. Esto puede ser realizado siguiendo los siguientes pasos:

1. Descargar el código fuente de *OpenWRT*, el mismo se encuentra disponible en github[29].

2. Descargar y copiar los parches generados a la carpeta "*openwrt/package/kernel/mac80211/patches/*"

3. Agregar las constantes "*MAC80211_RC_RRPAA*" y "*MAC80211_RC_DEFAULT_RRPAA*" debajo de "*MAC80211_RC_MINSTREL_VHT*" dentro del *Makefile* del módulo *mac80211* ubicado en "*openwrt/package/kernel/mac80211/Makefile*". El propósito de dichas variables es la compilación del módulo y selección del algoritmo *RRPAA* como algoritmo de control de *rate* por defecto (suplantando *Minstrel*).

4. Compilar el código de *OpenWRT* siguiendo la guía de compilación[30].

Una vez concluido este proceso, se puede utilizar el binario de *OpenWRT*, el cual es generado dentro la carpeta "*openwrt/bin*".

Dicho proceso de compilación puede ser realizado de forma automática utilizando el siguiente *script*:

```
#!/bin/bash

REPOSITORY_PATH="https://github.com/matir91/rrpaa.git"
REPOSITORY_PATCH_PATH='src'
REPOSITORY_CONFIG_FILE_FOLDER='01
    ↪ a95dfb90d4908549108f42239ad281'
REPOSITORY_CONFIG_FILE="https://gist.github.com/
    ↪ $REPOSITORY_CONFIG_FILE_FOLDER.git"

sudo apt-get update
sudo apt-get install subversion build-essential libncurses5-
    ↪ dev zlib1g-dev gawk git ccache gettext libssl-dev
    ↪ xsltproc wget unzip --yes

git clone -b chaos_calmer git://github.com/openwrt/openwrt.
    ↪ git
cd openwrt/
git checkout 2052672 # Version utilizada para desarrollo

git submodule add $REPOSITORY_PATH
cp $REPOSITORY_PATCH_PATH/* package/kernel/mac80211/patches/

sed -i -e '1452i\\tMAC80211_RC_RRPAA \\' package/kernel/
    ↪ mac80211/Makefile
sed -i -e '1452i\\tMAC80211_RC_DEFAULT_RRPAA \\' package/
    ↪ kernel/mac80211/Makefile



# La siguientes lineas puede ser removidas y aplicar el
    ↪ comando 'make menuconfig'
# para configurar la instalacion de openwrt como se desee
# mas informacion en https://wiki.openwrt.org/es/doc/howto/
    ↪ build
git submodule add $REPOSITORY_CONFIG_FILE
cp $REPOSITORY_CONFIG_FILE_FOLDER/.config .

make defconfig
make -j3
```

Bibliografía

[1] Matías Richart. Ieee 802.11 parameters adaptation for performance enhancement in high density wireless networks. Master's thesis, UR. FI-INCO, 2014. VII, 1, 7, 8, 17, 21, 22, 23, 24, 25, 70, 73

[2] Rrpaa. `https://github.com/matir91/rrpaa`. (Accessed on 02/25/2017). 2

[3] A Mcgregor and D Smithies. Rate adaptation for 802.11 wireless networks: Minstrel. *ACM SIGCOMM*, 2010. 5

[4] Pejman Roshan and Jonathan Leary. *802.11 Wireless LAN fundamentals*. Cisco press, 2004. 6

[5] Mesut Ali Ergin, Kishore Ramachandran, and Marco Gruteser. Understanding the effect of access point density on wireless lan performance. In *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, MobiCom '07, pages 350–353, New York, NY, USA, 2007. ACM. 9

[6] madwifi-project.org - trac. `http://madwifi-project.org/`. (Accessed on 02/01/2017). 11

[7] en:developers:documentation:mac80211 [linux wireless]. `https://wireless.wiki.kernel.org/en/developers/documentation/mac80211`. (Accessed on 02/07/2017). 11

[8] Mahesh K Marina et al. Understanding the role of multi-rate retry mechanism for effective rate control in 802.11 wireless lans. In *2009 IEEE 34th Conference on Local Computer Networks*, pages 305–308. IEEE, 2009. 11

[9] Bahareh Sadeghi. On hidden and exposed terminal problems. *IEEE 802.11-05/0065r0*, 9, 2005. 12

[10] Cunqing Hua and Rong Zheng. On link-level starvation in dense 802.11 wireless community networks. *Computer Networks*, 54(17):3159–3172, 2010. 15

[11] Thomas Hühn. *A Measurement-Based Joint Power and Rate Controller for IEEE 802.11 Networks*. PhD thesis, Oxford Brookes University, 2013. 17, 27, 30, 32

[12] Harald Schiöberg. *Interactions of legacy internet traffic with multihop wireless networks*. PhD thesis, Harvard University, 2012. 27

[13] Openwrt. `https://openwrt.org/`. (Accessed on 02/01/2017). 28, 39

[14] Lede project: Welcome to the lede project. `https://lede-project.org/`. (Accessed on 04/03/2017). 28

[15] www.dd-wrt.com | unleash your router. `http://www.dd-wrt.com/site/index`. (Accessed on 04/03/2017). 28

[16] pfsense® - world's most trusted open source firewall. `https://www.pfsense.org/`. (Accessed on 04/03/2017). 28

[17] Linux 2.6.27-rc3 [lwn.net]. `https://lwn.net/Articles/293784/`. (Accessed on 01/31/2017). 29

[18] Linux ip networking: A guide to the implementation and modification of the linux protocol stack. `https://www.cs.unh.edu/cnrg/people/gherrin/linux-net.html`. (Accessed on 01/31/2017). 29

[19] Linux/include/net/mac80211.h - linux cross reference - free electrons. `http://lxr.free-electrons.com/source/include/net/mac80211.h#L5430`. (Accessed on 01/31/2017). 30

[20] Cisco wireless control system configuration guide, release 3.2 - appendix b - supported country codes [cisco wireless control system] - cisco. `http://www.cisco.com/c/en/us/td/docs/wireless/wcs/3-2/configuration/guide/wcscfg32/wcscod.html`. (Accessed on 01/31/2017). 31

[21] thuehn/minstrel-blues: joint rate & power control within linux mac80211. `https://github.com/thuehn/Minstrel-Blues`. (Accessed on 02/10/2017). 32

[22] en:developers:documentation:mac80211:ratecontrol:minstrel [linux wireless]. `https://wireless.wiki.kernel.org/en/developers/documentation/mac80211/ratecontrol/minstrel`. (Accessed on 02/01/2017). 39

[23] Guido R Hiertz, Dee Denteneer, Lothar Stibor, Yunpeng Zang, Xavier Pérez Costa, and Bernhard Walke. The ieee 802.11 universe. *IEEE Communications Magazine*, 48(1), 2010. 39

[24] iperf - the tcp, udp and sctp network bandwidth measurement tool. https://iperf.fr/. (Accessed on 02/01/2017). 40

[25] en:users:documentation:iw [linux wireless]. https://wireless.wiki.kernel.org/en/users/documentation/iw. (Accessed on 02/01/2017). 40

[26] Lede project: Welcome to the lede project. https://lede-project.org/. (Accessed on 01/31/2017). 70

[27] Maximum recommended clients per access point | getting started with wireless | cisco support community. https://supportforums.cisco.com/discussion/11718356/maximum-recommended-clients-access-point. (Accessed on 02/01/2017). 72

[28] How real-world stress test of high-density wi-fi deployment worked. http://boundless.aerohive.com/experts/Testing-Quality-of-Experience-is-Testing-What-Matters.html. (Accessed on 02/01/2017). 72

[29] openwrt/openwrt: Linux distribution for embedded devices. https://github.com/openwrt/openwrt. (Accessed on 02/16/2017). 75

[30] Openwrt buildroot – uso [openwrt wiki]. https://wiki.openwrt.org/es/doc/howto/build. (Accessed on 02/16/2017). 75