Universidad de la República
Facultad de Ingeniería

Redes Cognitivas: Estudio de la Movilidad en el Espectro

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Germán Cuña, Rafael Durán, Mauricio Olivera

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutores

Pablo Belzarena ..... Universidad de la República
Federico Larroca ..... Universidad de la República
Germán Capdehourat ..... Universidad de la República

Tribunal

Eduardo Cota ..... Universidad de la República
Gabriel Gómez ..... Universidad de la República
Haldo Sponton ..... Universidad de la República

Montevideo
jueves 10 julio, 2014

Redes Cognitivas: Estudio de la Movilidad en el Espectro, Germán Cuña, Rafael Durán, Mauricio Olivera.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 119 páginas.
Compilada el jueves 10 julio, 2014.
http://iie.fing.edu.uy/

Hay una fuerza motriz más poderosa que el vapor,
la electricidad y la energía atómica: la voluntad.

Albert Einstein

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

En primer lugar a nuestras familias y amigos por el apoyo y la comprensión durante el proceso.

A nuestros tutores, Pablo, Germán y Federico por la guía técnica y sobretodo por su disposición en todo momento, apoyando y facilitando el trabajo del grupo.

En un sentido más amplio a nuestros compañeros y profesores que en el transcurso de la carrera han contribuido a nuestra formación académica y humana.

Al Instituo de Ingeniería Eléctrica (IIE) por facilitarnos los materiales necesarios para la realización del proyecto.

A todos MUCHAS GRACIAS!

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Con el avance tecnológico se está sufriendo una insuficiencia en el espectro radioeléctrico utilizado para las telecomunicaciones. Esta insuficiencia es consecuencia en parte de un uso ineficiente del espectro. Con el propósito de mejorar este problema han surgido varias vetas de estudio, algunas de ellas se tratan en este documento: las Redes Cognitivas y la Movilidad Espectral.

Las Redes de Radio Cognitivas se basan en el hecho de compartir el espectro. Cuando algún usuario deja libre alguna porción de espectro otro usuario lo podría utilizar bajo ciertas restricciones. Esta tesis se enfoca en el estudio del acceso a los canales libres y cómo minimizar el tiempo de este proceso. Para realizar el estudio se plantea el escenario de una comunicación punto a punto. Bajo esta situación se estudian métodos para coordinar el pasaje hacia un nuevo canal y el restablecimiento de la comunicación. Básicamente los métodos se centran en los llamados **algoritmos de encuentro**, los cuales se encargan de generar secuencias numéricas (basadas en distintas propiedades matemáticas) que representan los canales a los que se intentará acceder para reanudar la comunicación.

Luego de una extensa revisión bibliográfica del tema se hallan un gran numero de algoritmos de encuentro, de éstos se seleccionan algunos para estudiar en profundidad (*Jump Stay*, *MCA*, *MMCA*, *DRSEQ* y *CRSEQ*). Luego se simulan en *MatLab* y por último se estudia el desempeño de los mismos en una comunicación punto a punto con las radios cognitivas enteramente desarrolladas para este proyecto.

Como consecuencia, este documento contiene, por un lado el estudio de los algoritmos de encuentro seleccionados, y por otro el diseño, desarrollo y estudio de desempeño de la radio cognitiva implementada en *hardware*.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Descripción del Proyecto

Con el avance de las telecomunicaciones se ha comenzado a sufrir una insuficiencia de espectro útil, principalmente debido a un uso ineficiente del mismo, tanto temporal como geográfico. Las actuales redes inalámbricas tienen como característica habitual la utilización de una banda de frecuencia fija, y una gran porción de ese espectro asignado es usado esporádicamente, dejando así "huecos" que de ocuparse mejorarían la eficiencia en el uso. A partir de esto se ha comenzado a estudiar posibles formas de mejorar ese uso basándose en ocupar anchos de banda cuando los licenciatarios de los mismos, llamados usuarios primarios (PU), no los están usando, y devolverlos cuando comiencen a transmitir nuevamente. En vista de lo anterior se están desarrollando técnicas de acceso dinámico al espectro, y la clave para estas técnicas son las Redes Cognitivas. Para que las mismas funcionen como se desea hay cuatro tareas que deben realizar satisfactoriamente:

- Determinar qué porciones del espectro están libres.
- Seleccionar el mejor canal entre los disponibles.
- Coordinar el acceso al canal seleccionado con los demás usuarios.
- Dejar libre el canal cuando se detecta a un usuario primario.

Dentro de las dos últimas tareas descriptas se centra el proyecto. Para el mismo se toma como hecho que la red puede detectar que un usuario primario ocupó el canal, y se centra en estudiar y evaluar algoritmos con el fin de coordinar y realizar el cambio de canal sin perder la comunicación. Para hacer estos estudios se plantea el escenario de una comunicación punto a punto entre dos usuarios no licenciatarios, también llamados usuarios secundarios (SU), y se analiza la movilidad de éstos en el espectro frente a la necesidad de un usuario primario en utilizar el canal.

Los pasos a seguir para cumplir con este objetivo se detallan a continuación. Se comienza con una extensa revisión bibliográfica en el tema y un estudio general de los algoritmos de encuentro. Luego se seleccionan los algoritmos de encuentro

más populares para realizar un estudio profundo. Los algoritmos son, ***Jump Stay*** [16], ***MCA*** y ***MMCA*** [14], ***DRSEQ*** [8] y ***CRSEQ*** [11]. Se comienza con una comprensión de los mismos y estudio de su forma de funcionamiento, basándose en modelos teóricos propuestos por sus respectivos autores. A partir de esto se ensaya su desempeño modificando las hipótesis de funcionamiento (esto se realiza mediante una serie de simulaciones en ***MatLab***).

Luego de la comprensión de los diferentes algoritmos se pasa al **desarrollo de la radio cognitiva en *hardware***, este es el paso siguiente para estudiar el desempeño de los mismos fuera de un ambiente simulado y enfocado al prototipo final. Para esto se requiere definir las tareas que serán necesarias cumplir para continuar la comunicación. Se debe destacar que en este punto se optó por añadir un objetivo particular al proyecto. El mismo es que la plataforma debería ser diseñada con una estructura modular, entendiendo que las diferentes tareas pueden llegar a ser muy complejas y en el futuro se pueden lograr implementaciones más eficientes de alguno de los bloques. Solamente es necesario modificar alguno de los módulos y la plataforma seguirá funcionando correctamente.

Como paso final se prueban los diferentes algoritmos en la **plataforma de control** diseñada y se compara su desempeño con el esperado en las simulaciones.

1.2. Estado del arte

Desde los comienzos en la utilización de la tecnología de radio para la transmisión de información inalámbrica se han destinado dispositivos (radios) con *hardware* dedicado para realizar distintas funciones según el tipo de comunicación y los requerimientos del medio. Frente a esta realidad surge el desafío de obtener una radio capaz de utilizar el *hardware* en una forma genérica y de reconfigurarse mediante *software* en distintos momentos. El concepto de *Software Radio* (*SR*) recoge esta idea. Mitola la define en [13] como: *"A Software Radio is a radio whose channel modulations waveforms are defined in software. That is, waveforms are generated as sampled digital signals, converted from digital to analog via wideband DAC (Digital to Analog Converter) and the possibly unconverted from IF (Intermediate Frequency) to RF (Radio Frequency). The receiver, similarly, employs a wideband ADC (Analog to Digital Converter) that captures all the channels of the software radio node. The receiver then extracts, down converts and demodulates the channel waveform using software on a general purpose processor."*

Esta flexibilidad implica tener el *hardware* lo suficientemente potente como para implementar la *SR* en un dispositivos de comunicación inalámbrica, lo cual no es posible actualmente en un sentido ideal. La plataforma bajo la cual se implementa *SR* es *Software Defined Radio* (*SDR*). En el marco de una *SDR* se tienen radios que realizan el procesamiento de la señal en *software* sin requerir la modificación de la misma en *hardware* dedicado. La digitalización de la señal se realiza en el punto más próximo a la antena pasando a tener las muestras disponibleas para manejarse en un procesador en tiempo real.

Los primeros desarrollos de *SDR* se fundamentaron en proyectos para aplicaciones militares. Implementaciones como el proyecto *SpeakEasy* [6] o el programa

Joint Tactical Radio Systems [4] buscaron la realización de radios que seguían las líneas de *SDR*. En el presente la herramienta de *SDR* más popular es *GNU radio* [1], la cual se describe en 2.1 ya que se utiliza de forma central en este proyecto. En conjunto con ésta se utilizan dispositivos de *hardware* compatibles con el objetivo de implementar la interfaz física en la comunicación. Un ejemplo de éstos es *USRP* (*Universal Software Radio Peripheral*), desarrollado por Matt Ettus [3].

La aplicación de *SR*, en particular *SDR*, se presenta como una solución extremadamente adecuada para la situación actual en el ámbito de las telecomunicaciones, especialmente en redes inalámbricas. Las políticas de los distintos organismos que regulan las redes y la asignación fija del espectro genera una utilización poco adecuada del recurso. Así, bandas de frecuencia por debajo de los 3 Ghz, valiosas por sus características de propagación son cada vez más escasas cuando se busca desplegar una nueva red. En vistas de la necesidad de mejorar la utilización del espectro surge el estudio de las llamadas Radios Cognitivas (*Cognitive Radios -CR-*), las cuales se definen como radios capaces de cambiar sus parámetros de transmisión basadas en la interacción con el ambiente en el que operan [9]. De esta forma, parámetros como la frecuencia de portadora, modulación, acceso al medio, etc. son controlados por la radio misma a partir de decisiones basadas, por ejemplo, en el estado de los canales de comunicación o la situación de otras radios en una red. Si bien este concepto es abierto, y como lo remarca en [9] no requiere de una implementación basada en software o más específicamente con *SDR*, se desprende que la utilización de este último es el camino natural. De la definición presentada se obtienen dos características principales que debe tener una radio cognitiva [12]:

- **Capacidad cognitiva**: Tecnología necesaria para capturar la información de su entorno de radiofrecuencia e identifica las partes del espectro que no estén siendo utilizadas.

- **Auto-reconfiguración**: Tecnología necesaria para que el dispositivo pueda variar, de manera dinámica, distintos parámetros relacionados con la transmisión o recepción (frecuencia, potencia, modulación, etc.), de acuerdo con su entorno.

Dentro de las aplicaciones y campos en los cuales actualmente se estudia la implementación de las *CR* se destaca en primer lugar el análisis con señales *ATSC DTV* (*Advanced Television System Committee for Digital Television* [2]). En este sentido ha ido el esfuerzo de IEEE en generar el estándar 802.22 [7], que precisamente explora y define las capas de control de acceso al medio (MAC) y física (PHY) para el despliegue de una red haciendo uso de los huecos en el espectro en las bandas de TV. Por otro lado se encuentra el trabajo pionero de *DARPA* (*Defense Advanced Research Projects Agency*) en el desarrollo de sistemas que hacen uso compartido del espectro por medio de técnicas adaptativas como *DSA* (*Dynamic Spectrum Access*).

Se puede decir que los esfuerzos y el estado actual persigue el objetivo paradigmático de las radios cognitivas, esto es, la correcta detección de bandas inutilizadas en el espectro y la coordinación en la movilidad entre distintos canales.

1.3. Objetivo general y alcance del proyecto

En este proyecto se tiene como objetivo el estudio e implementación en *USRP*, de algoritmos de manejo del espectro en *SDR*. Además, se buscará que la plataforma a entregar al cliente sea reutilizable para futuros estudios en el área.

El alcance del proyecto es estudiar y simular al menos cuatro de los algoritmos de encuentro propuestos en publicaciones. Luego se plantea la implementación en una comunicación punto a punto en una radio cognitiva en *USRP*. Se incluye además el estudio del desempeño de los algoritmos implementados.

1.3.1. Restricciones

- La implementación se realiza solamente con el *software GNU Radio*, sobre *Python* y *C++*.
- La implementación se realiza en los *USRP* que posee el cliente.
- Sólo se trabaja sobre comunicaciones punto a punto.

1.3.2. Criterios de éxito

Resultados satisfactorios en la prueba experimental de alguno de los algoritmos implementados en *USRP*. Los resultados serán satisfactorios si luego de establecida una comunicación punto a punto, se da el cambio de canal de ambas radios y se logra continuar la comunicación en el nuevo canal seleccionado.

1.4. Descripción de los capítulos

En el segundo capítulo se describen las herramientas utilizadas, tanto de *software* como de *hardware*.

El tercer capítulo, "Algoritmos de Encuentro", analiza el proceso por el cual las dos radios cognitivas se encuentran en un canal común. El mismo se encuentra dividido según los algoritmos que se estudian y posteriormente se implementan. En cada caso se realiza una descripción general del algoritmo, luego se definen los detalles del mismo y por último se presentan resultados del rendimiento del mismo a partir de las simulaciones realizadas.

En el cuarto capítulo se presenta el diseño y desarrollo de la radio cognitiva implementada. La misma es la que brinda soporte para que los algoritmos bajo estudio se ejecuten y arrojen los resultados que se desean analizar. En esta sección se realiza una presentación progresiva partiendo desde la idea conceptual hasta los detalles de la implementación obtenida.

El quinto capítulo presenta un análisis del desempeño de la radio cognitiva desarrollada, así como también se justifica el rendimiento de la misma y propone hacia dónde apuntar el trabajo a futuro para mejorarlo.

En el sexto capítulo se muestran los resultados obtenidos a partir de la ejecución de los algoritmos en la radio que se implementó. Inicialmente se realiza una

descripción de la plataforma de prueba para obtener las mediciones y se definen los parámetros que se decide observar. Seguidamente se presentan los resultados numéricos para cada algoritmo. Finalmente se realiza para cada uno una comparación con los resultados "teóricos" a partir de las propuestas originales y los resultados de las simulaciones según se registró en el capítulo dos.

En el capítulo siete se presentan las conclusiones del proyecto de forma global y además se mencionan las posibilidades de trabajo a futuro.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

Descripción de herramientas utilizadas

2.1. Plataforma de desarrollo GNU-Radio

La herramienta de *software* que se utilizará para la implementación de la radio cognitiva, es una herramienta de desarrollo abierta y libre que se utiliza para implementar sistemas de *SDR* llamada *GNU Radio*. Puede utilizarse con *hardware* de bajo costo, o sin el mismo para realizar simulaciones. Como se explica en [1] es muy utilizada en ambientes académicos, aficionados y comerciales, en estos últimos como soporte a la investigación en comunicaciones inalámbricas y en sistemas de radio en el mundo real.

Con *GNU Radio* se puede trabajar con datos que se hayan generado o relevado previamente, con lo que se obtiene un marco para el desarrollo de algoritmos y estudio de procesamiento de señales sin necesidad de tener que estar obteniendo las señales en tiempo real. Todo el código fuente tiene los derechos de autor de *Free Software Foundation*, lo que implica que el trabajo con el espectro electromagnético se torne más accesible.

En *GNU Radio* se utilizan dos lenguajes de programación, ***Python*** para escribir las aplicaciones y ***C++*** para las partes en que se requiere procesamiento de señal. Por lo tanto los bloques que procesan señales están desarrollados en *C++*, mientras que *Python* es utilizado para generar flujos conectando los bloques anteriores. De esta manera, se pueden desarrollar sofisticados sistemas de radio de forma simple y accesible.

Una clave en los sistemas *SDR* es la reconfigurabilidad. Esto implica que se debe tener una radio genérica que implemente el procesamiento de la señal en software. El *hardware* de bajo costo utilizado para la implementación de *SDR* por *GNU Radio* es el llamado *Universal Software Radio Peripheral* (***USRP***), el cual se encargará de digitalizar los datos del aire y entregárselos a *GNU Radio* a través de alguna interfaz, por ejemplo *USB*. Esta segmentación se puede apreciar en la figura 2.1.

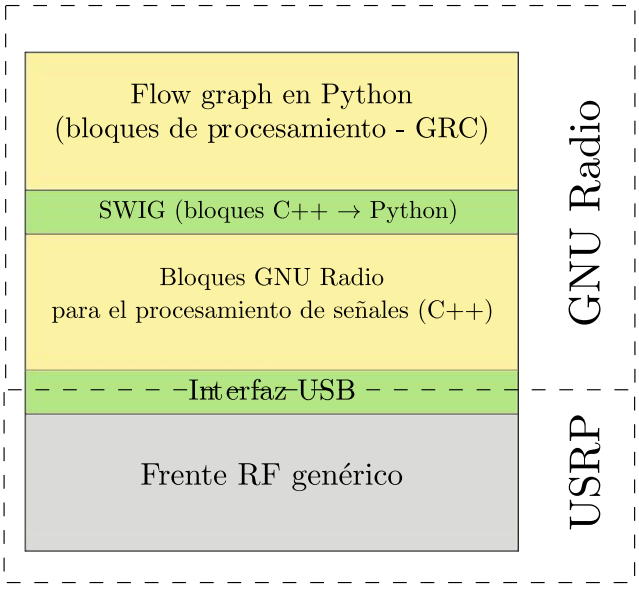


Figura 2.1: GNU Radio

GNU Radio además cuenta con una herramienta gráfica para la creación de flujos de comunicación y a partir de estos generar el código fuente en *Python*. Esta herramienta se llama **GNU Radio Companion** o **GRC**. A través de una interfaz gráfica e interactiva se pueden conectar los diversos bloques de procesamiento de señal y configurar sus parámetros. De esta forma queda determinado el flujo de señal formando lo que se denomina *flowgraph*.

2.2. Descripción de hardware

A continuación se describe el *hardware* que se utiliza para la implementación de la radio cognitiva.

2.2.1. USRP

Los *Universal Software Radio Peripheral* (*USRP*) son una plataforma de *hardware* flexible y de bajo costo pensada para el desarrollo e implementación de *SDR*. Esta plataforma es desarrollada por *Ettus Research* [3] y consiste principalmente en dos placas, la *motherboard* y la *daughterboard*, o también llamadas placa primaria y placa secundaria respectivamente. En la placa primaria se encuentra el ***FPGA*** (*Field Programmable Gate Array*), los convertidores ***ADC*** y ***DAC***, la alimentación y la conexión *USB*. Mientras que las placas secundarias son las que se encargan de traer la señal a banda base cuando se recibe o llevar la señal a pasabanda cuando se transmite. El *FPGA* antes mencionado es el procesador del *USRP*.

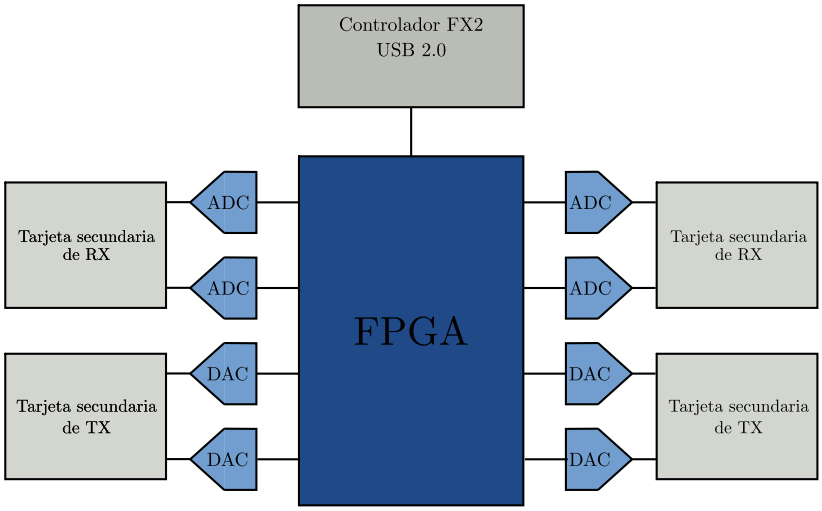


Figura 2.2: Diagrama de bloques de un USRP

En la figura 2.2 se puede apreciar un diagrama de la arquitectura antes descripta.

Particularmente en el desarrollo de este proyecto se utiliza el ***USRP B100*** (figuras 2.3 y 2.5), algunas de las especificaciones del mismo son:

- Interfaz USB 2.0
- FPGA Xilinx Spartan 3A-1400
- ADCs Dual 64 MS/s 12-bit
- DACs Dual 128 MS/s 14-bit
- Reloj flexible desde 10 MHz a 64 MHz
- 8 MHz de ancho de banda con muestras de 16 bit
- 16 MHz de ancho de banda con muestras de 8 bit

Figura 2.3: USRP utlizados.

Las placas secundarias que se utilizaron con los *USRP* son el modelo *WBX* y como ya se explicó las mismas se encargan de implementar el *transceiver* en radio frecuencia.

A partir de esto queda definida la arquitectura de las *SDR* que actuarán en la radio a implementar, donde cada usuario contará con un transmisor y un receptor de la forma que se muestra en la figura 2.4.

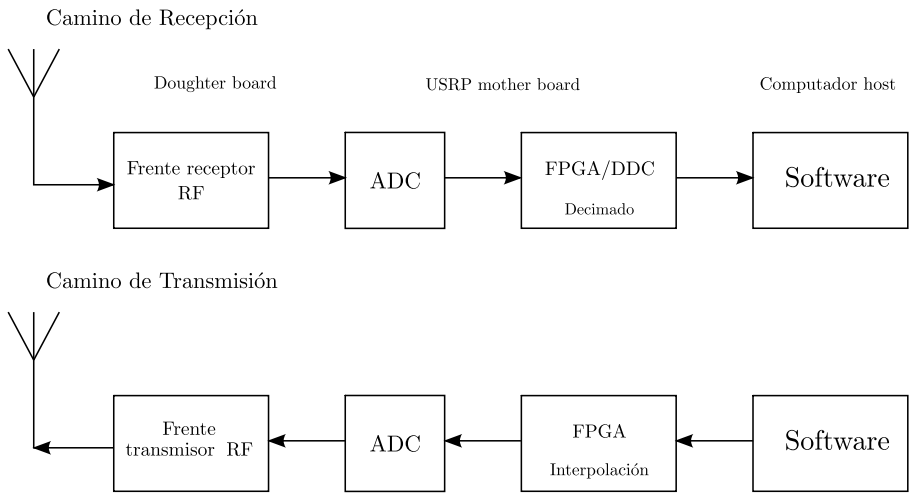


Figura 2.4: Arquitectura de SDR

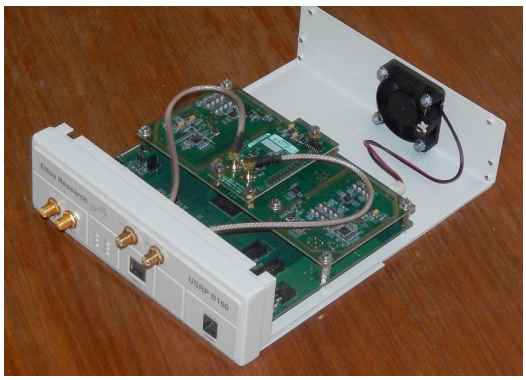


Figura 2.5: Interior de un USRP, con las *daughterboards* a la vista.

2.2.2. Antenas

Las antenas utilizadas son del tipo log periódica y se encuentran implementadas en *PCB* (*printed circuit boards*)2.6. El rango de frecuencia con performance óptima se encuentra en entre los 850 - 6500 MHz.

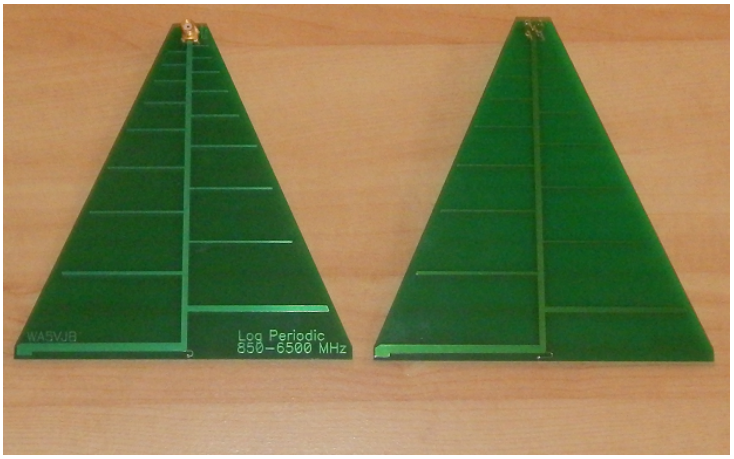


Figura 2.6: Antenas utilizadas en los USRP

2.2.3. Computadoras

Se utilizan dos *notebook*.
Una modelo *Emachines D728* con las siguientes características:

- Procesador *Intel Pentium T4500 2.3 GHz*
- Memoria 2 GB DDR3

Y una *Hewlett-Packard hp 550* con las siguientes características:

- Procesador *Intel Core 2 Duo E4300 1.8 GHz*
- Memoria 2 GB DDR3

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Algoritmos de Encuentro

3.1. Introducción

Para introducir la idea por detrás de estos algoritmos se presenta el siguiente ejemplo. Dos usuarios, A y B, se encuentran con una comunicación establecida haciendo uso exclusivo de un canal en el espectro. En cierto momento se ven en la situación de tener que realizar un cambio de canal manteniendo la comunicación que ya tienen establecida, ¿qué opciones tienen?. Una opción es realizar una consulta central de los canales libres y disponibles, para finalmente acordar entre ambos a cuál de estos canales se moverán. Otra opción, en caso de no tener un sistema centralizado, podría ser moverse independientemente y de forma aleatoria hacia un canal. Realizando este procedimiento repetidamente llegaría el punto en que finalmente se encontrarían en un canal.

Teniendo en cuenta lo anterior, se llama ***Rendezvous*** al proceso por el cual los SU se encuentran. De esta manera se dice que se logró el *Rendezvous* cuando se produce el encuentro. En el afán de lograr procedimientos que garanticen el encuentro de una manera óptima es que se ha trabajado en el campo de las redes cognitivas. A grandes rasgos se puede decir que el que garantiza *Rendezvous* (en condiciones razonables) en el menor tiempo es el mejor.

Para atacar este problema se han propuesto **algoritmos que garantizan el *Rendezvous*** o al menos minimizan la cantidad de casos donde no se da el encuentro. También se han dedicado estudios a comparar la performance de los mismos. En secciones posteriores se presentan, analizan y comparan algunos de estos algoritmos. Para realizar esta tarea, por un lado, se fijan las condiciones de trabajo (elección de sistema donde se realizará el rendezvous), y por otro lado se seleccionan los tiempos a medir como parámetros de la performance. Estos últimos son:

- ***TTR*** (*Time To Rendezvous*): cantidad de *time slots* (*TS*) hasta que se da el *Rendezvous*
- ***MTTR*** (*Maximum Time To Rendezvous*): cantidad máxima de *TS* que podria llegar a demorar alcanzar el encuentro.

En esta línea los autores de [17] dan una clasificación de los distintos tipos de algoritmos según el tipo de sistema en el que puede trabajar. Un diagrama de esta taxonomía se puede apreciar en la figura 3.1

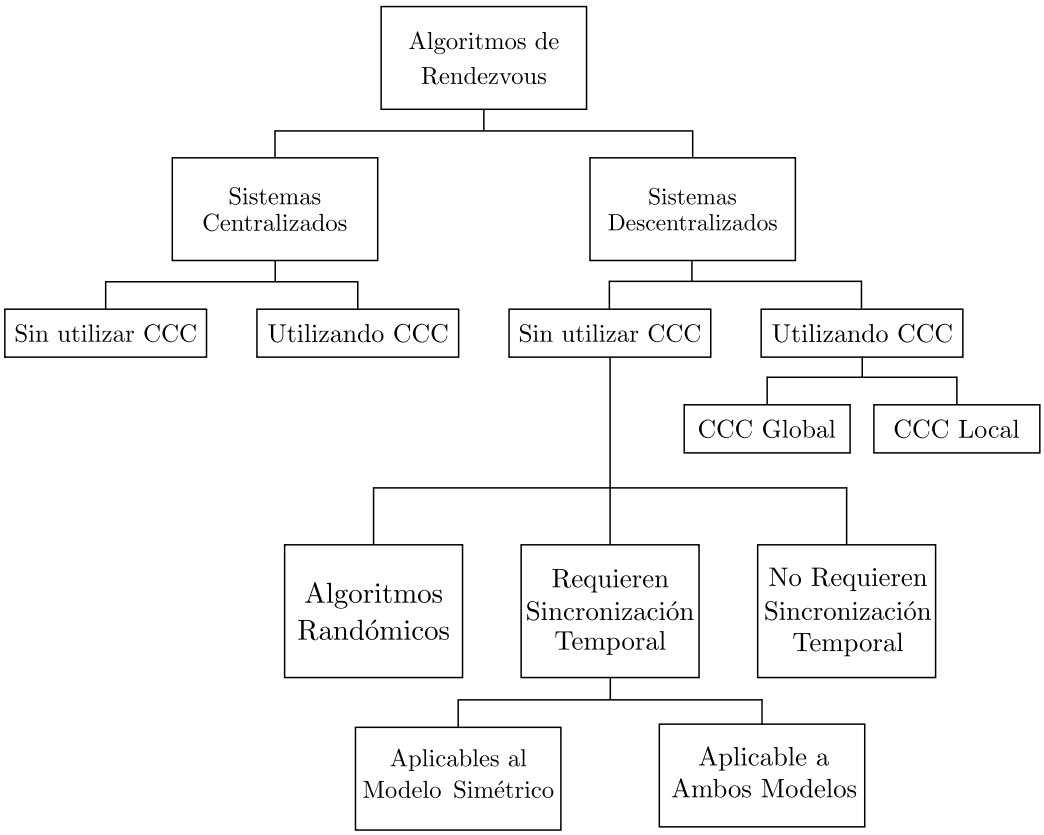


Figura 3.1: Taxonomia de Algoritmos de Rendezvous

Para entender la figura 3.1 se deben explicar los siguientes conceptos:

- **Sistemas Centralizados**: Sistemas que requieren un servidor central para realizar el encuentro.
- **Sistemas Descentralizados**: Sistemas que no necesitan un servidor, aunque si podrían requerir un **CCC**.
- **CCC** (*Common Control Channel*): canal siempre disponible entre los distintos usuarios por el cual se intercambia información de control, por ejemplo el próximo canal en el que seguirá la transmisión.
- **Algoritmos Randómicos**: Algoritmos que generan la secuencia de canales en forma aleatoria.
- **Sincronización Temporal**: refiere a si existe o no sincronización a nivel de *time slot*.

- **Modelo Asimétrico**: En este modelo se considera que todos los usuarios tienen distintas listas de posibles canales disponibles pero con al menos un canal común.
- **Modelo Simétrico**: En este caso se considera que todos los usuarios tienen la misma lista de posibles canales disponibles.

Los algoritmos para sistemas descentralizados y que no utilizan canal de control, se les llama algoritmos ciegos o ***Blind Rendezvous*** ya que los usuarios no hacen uso de una comunicación exclusiva para acordar en qué canal encontrarse. Básicamente estos algoritmos se encargan de establecer los movimientos entre los canales, a este proceso de saltar de un canal a otro se le llama ***Channel Hopping*** (CH). Sobre este tipo de algoritmos se centra el estudio.

3.1.1. Modelo de sistema a utilizar

Para el estudio de los algoritmos se considera que la comunicación es **punto a punto**, o sea, solo entre dos usuarios. Se toma el **modelo simétrico** teniendo en cuenta que alguno de los canales de la lista podrían estar no disponibles, por ejemplo porque está ocupado por un PU. Por lo tanto se le da el mismo tratamiento a los casos en los que no se produzca el encuentro por distintos resultados en el CH a cuando el canal no esté disponible.

Para la descripción de los algoritmos se utiliza la siguiente notación:

- **M**: cantidad de canales posibles.
- $\mathbf{c_i}$: canal i.
- **P**: menor número primo mayor o igual que M.
- **t**: número de *time slot*.

Seguidamente se presenta un estudio general de algunos algoritmos seleccionados. Luego, para medir el desempeño se realizan simulaciones en ***MatLab*** restringiéndose al modelo de sistema descripto. Dichas simulaciones se realizan con los siguientes parámetros:

1. $\mathbf{M = 30}$.
2. **Cantidad de ejecuciones** $\mathbf{= 1000}$.
3. **Cantidad de canales no disponibles** $\mathbf{= 1, 10, 20, 29}$.

Las simulaciones tomando distinta cantidad de canales no disponibles representa un acercamiento a un ambiente real. Es razonable suponer la presencia de PU ocupando parte del ancho de banda. Este estudio cuantifica la robustez de los algoritmos frente a distintos porcentajes de canales no disponible para alguno de los usuarios.

3.2. DRSEQ y CRSEQ

Los algoritmos presentados en esta sección, **DRSEQ** y **CRSEQ**, se describen en [8] y [11] respectivamente.

3.2.1. Descripción

El acercamiento para la realización del *Rendezvous* en estos dos métodos consiste en que ambos usuarios secundarios realicen la búsqueda de un canal común por medio de **secuencias predefinidas**. El algoritmo en si no presenta complejidad mayor que esa, esto es, ambos SU visitan canales potencialmente ocupables para establecer la comunicación según un orden preestablecido.

Lo que requiere un diseño y análisis más delicado es el método por el cual se construyen las secuencias. Dicha construcción busca **minimizar el *MTTR***, inclusive cuando los radios no están sincronizados. Para esto último se presentan los métodos a continuación.

3.2.2. Consideraciones preliminares

Además de los parámetros definidos en la introducción se tienen en cuenta las siguientes definiciones:

- El tiempo se encuentra dividido en TS de igual duración T. Los slots se numeran desde 0 en adelante.
- La secuencia de *Rendezvous* es $SEQ = \{c_0, c_1, ..., c_{M-1}\}$.
  - La relación entre el numero de slot (I) y el número de canal visitado se puede expresar como $a_{i=I\,mod M}$
  - Los elementos de las secuencias para los nodos A y B se denotan c_i^A y c_i^B respectivamente.

A continuación se define una propiedad de las secuencias, la cual se utiliza para garantizar el *Rendezvous*.

Definición 1 *La secuencia de Rendezvous* $SEQ = \{c_0, c_2, ..., c_{M-1}\}$ *es Invariante bajo desfasaje* k $(k = 0, 1, ..., M-1)$*, si existe un slot* $I \in k, k+1, ..., k+(M+1)$ *tal que* $c^A_{i=I mod M} = c^B_{i=(I-k) mod M}$

El problema de *Rendezvous* utilizando estos algoritmos se puede presentar frente a dos posibles escenarios, con sincronización de tiempo (a nivel de TS) o de forma asíncrona.

Adicionalmente para cualquiera de los dos casos se contempla la situación en que ambos radios no comienzan a realizar la búsqueda de forma sincronizada, o sea si el radio A comienza a realizar la búsqueda en un tiempo x_1, el radio B comienza en un tiempo x_2, con $x_1 \neq x_2$.

3.2.3. Secuencia determinística de encuentro

En este algoritmo (*DRSEQ*) se construye la secuencia según la siguiente fórmula:

$$a_i = \begin{cases} i+1 & \text{si } 0 \leq i \leq M-1 \\ e & \text{si } i = M \\ 2M-i+1 & \text{si } M+1 \leq i \leq 2M \end{cases}$$

Donde e denota el slot vacío.

La secuencia que se obtiene es de la forma que se muestra en la figura 3.2.

número de slot	0	1	2	3	4	5	6	7	8	9	10	11	12
secuencia	1	2	3	4	5	6	e	6	5	4	3	2	1

Figura 3.2: DRSEQ para $M = 6$

El teorema 1 demuestra que este tipo de secuencias cumplen la propiedad definida en 1 y por lo tanto se asegura el *Rendezvous*.

A partir del siguiente teorema se asegura teóricamente el *Rendezvous* bajo las condiciones que se mencionan. La demostración del mismo se encuentra en [8].

Teorema 1 *Para M canales disponibles, DRSEQ es invariante bajo un desfasaje k para $k = 0, 1, ..., 2N$ tal que dos nodos A y B realizan el Rendezvous dentro de $2M + 1$ slots.*

Un esquema gráfico de cómo se da el rendezvous se muestra en la figura 3.3. En este caso se elige $M = 6$ y se muestra para $k = 0, 1, 2, ...$

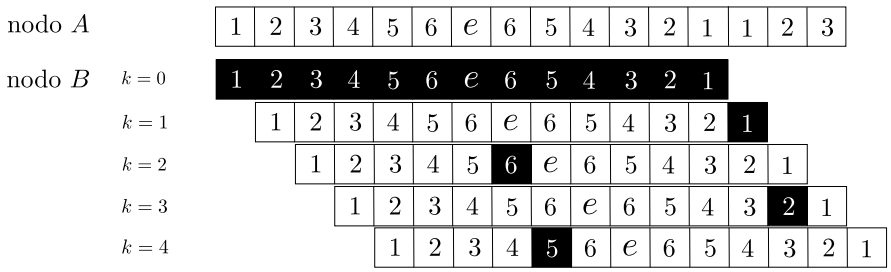


Figura 3.3: Ilustración de DRSEQ con $N = 6$

El caso asíncrono

Para el caso en que no se tenga sincronización a nivel de *TS* el análisis y los resultados son análogos que para el caso estudiado. Para estudiar el algoritmo se supone un tiempo de slot del doble que en el caso en que hay sincronización. De esta manera si $T_{sinc} = T$ siendo T_{sinc} el tiempo de slot con sincronización y T un tiempo dentro del cual se puede establecer el enlace entre dos nodos, se toma $T_{no\ sinc} = 2T_{sinc} = 2T$, con $T_{no\ sinc}$ el tiempo de slot para el caso asíncrono. Bajo esta consideración también se prueba que por el teorema 1 se alcanza *Rendezvous* dentro de $2N + 1$.

3.2.4. Secuencia de canales para el encuentro

En este algoritmo (*CRSEQ*) la secuencia se construye basada en propiedades de los números triangulares y operaciones modulares. La secuencia consiste de P subsecuencias (siendo P el ya definido) y cada subsecuencia se compone de $3P-1$ elementos, obteniendo un largo de $P(3P-1)$.

La j-esima subsecuencia comienza con el número triangular $T_j = \frac{j(j+1)}{2}$. Dentro de esta subsecuencia los primeros $2P-1$ se computan de tal manera que el l-ésimo elemento es igual a $(T_j+l) mod\ M+1$, y los restantes elementos son iguales a $j+1$.

A partir de lo anterior la secuencia se calcula a partir de la siguiente fórmula.

$$a_i = \begin{cases} z\ mod\ M+1 & \text{para } 0 \leq y \leq 2P-1 \\ x\ mod\ M+1 & \text{para } 2P-1 \leq y < 3P-1 \end{cases}$$

donde

- $z = (\frac{x(x+1)}{2} + y)\ mod\ P$
- $x = \lfloor \frac{i}{3P-1} \rfloor$
- $y = i\ mod\ 3P-1$
- $0 \leq i < P(3P-1)$
- P es el número primo mayor o igual a M

A partir del siguiente teorema y haciendo uso de la propiedad definida anteriormente se asegura el *Rendezvous* para el caso en que se tiene sincronización. El *Rendezvous* se asegura dentro de $P(3P-1)$ slots.

Teorema 2 *Para $M(\geq 2)$ canales, CRSEQ es invariante bajo un desfasaje k para todo $k(=0,1,...,(M-1))$ tal que un nodo A y un nodo B realizan el Rendezvous en un canal común $c \in S^A \cap S^B$ dentro de $P(3P-1)$. Donde S^x es el conjunto de canales disponibles observado por el nodo x.*

El caso asíncrono

Para este caso el análisis y resultados son análogos que lo que se indica para *DRSEQ*.

3.2.5. Análisis de desempeño del Algoritmo

Comparación entre DRSEQ y CRSEQ

Para realizar esta evaluación se corre 250 veces el algoritmo para cada M, y se toma el promedio. En ambos casos se toma en cuenta el modelo simétrico y el desfasaje entre ambos usuarios es aleatorio (dentro de lo establecido en cada algoritmo). Los resultados de la simulación se muestran en las figuras 3.4 y 3.5.

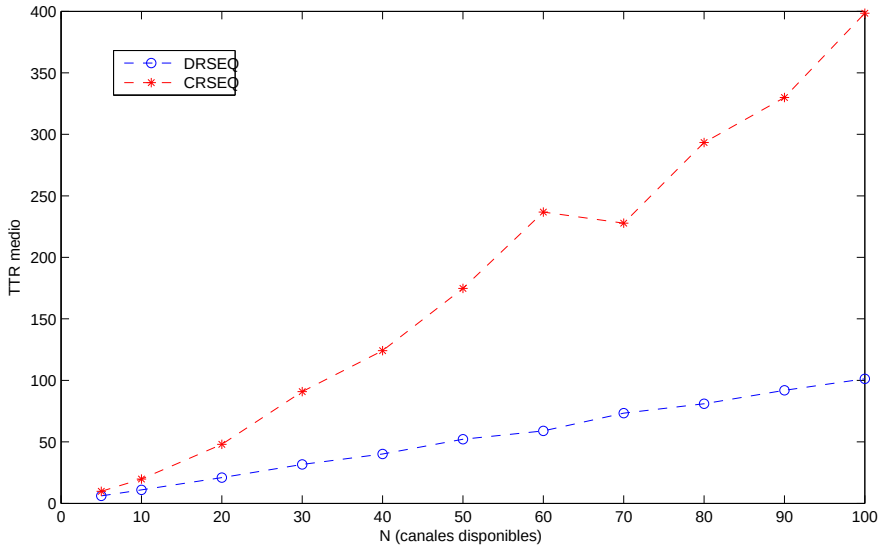


Figura 3.4: TTR promedio Vs M

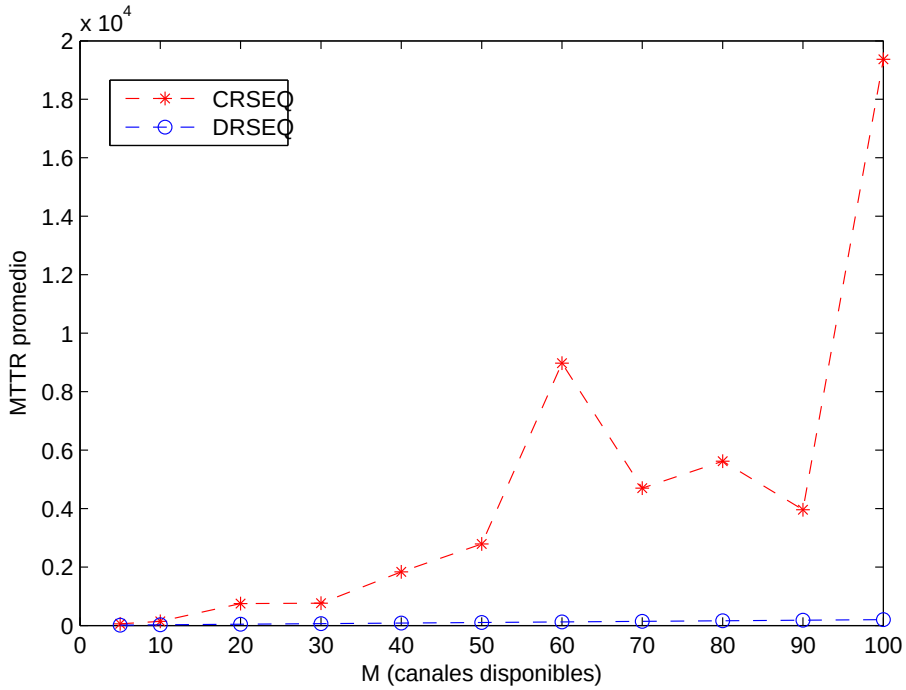


Figura 3.5: MTTR promedio Vs M

A partir de estas simulaciones se observa que el desempeño de CRSEQ en términos de TTR y MTTR es inferior a DRSEQ. Sin embargo, se debe tener en

cuenta que estos comportamientos son bajo el modelo simétrico, modelo sobre el cual no está asegurado el *Rendezvous* para DRSEQ.

3.2.6. Análisis de desempeño bajo el modelo definido

Seguidamente se estudia el desempeño de CRSEQ cuando se retiran canales disponibles para realizar el *Rendezvous*.

En figuras 3.6 hasta 3.9 se presentan histogramas para el caso particular mencionado en la introducción.

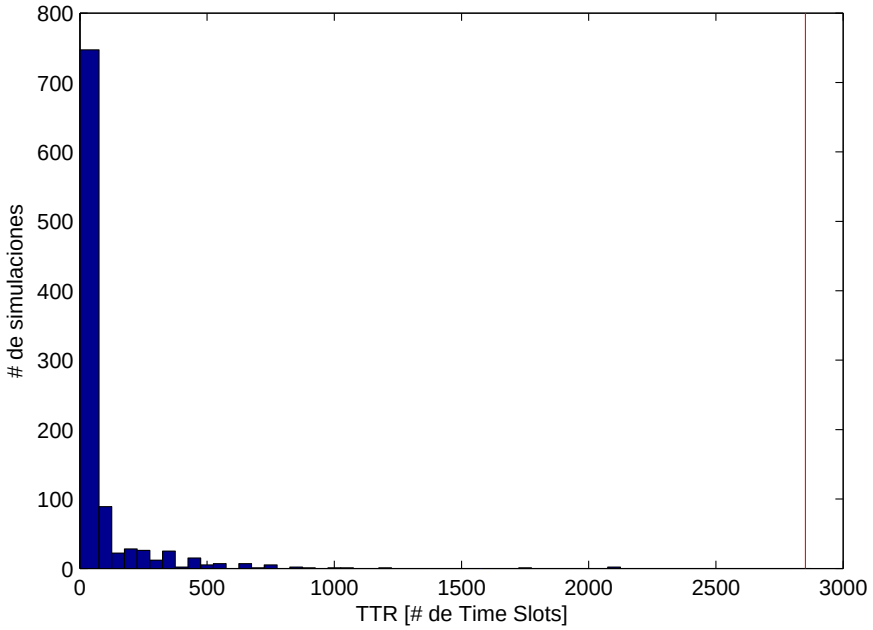


Figura 3.6: Histograma tomando que 1 de 30 canales se encuentra no disponible

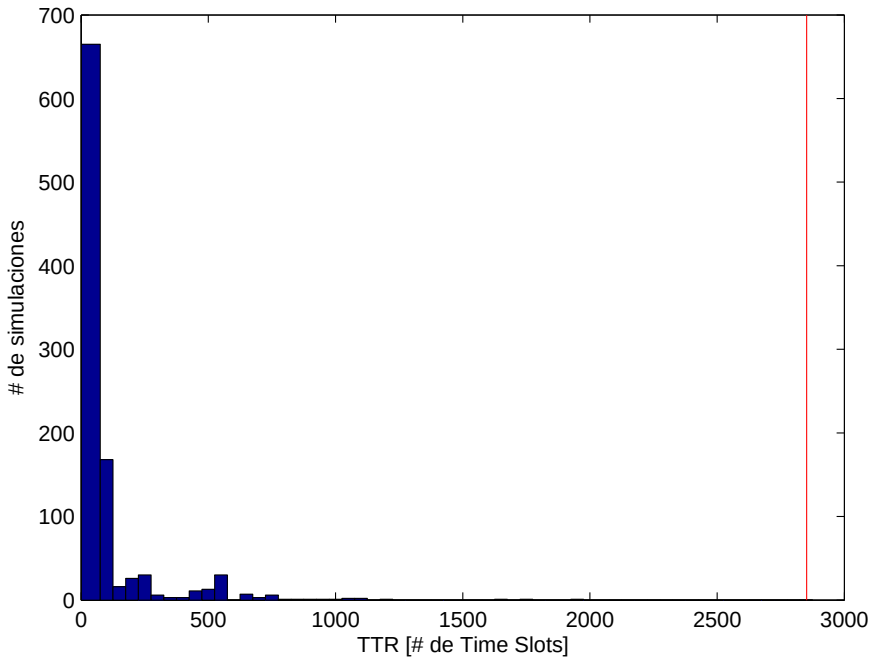


Figura 3.7: Histograma tomando que 10 de 30 canales se encuentran no disponibles

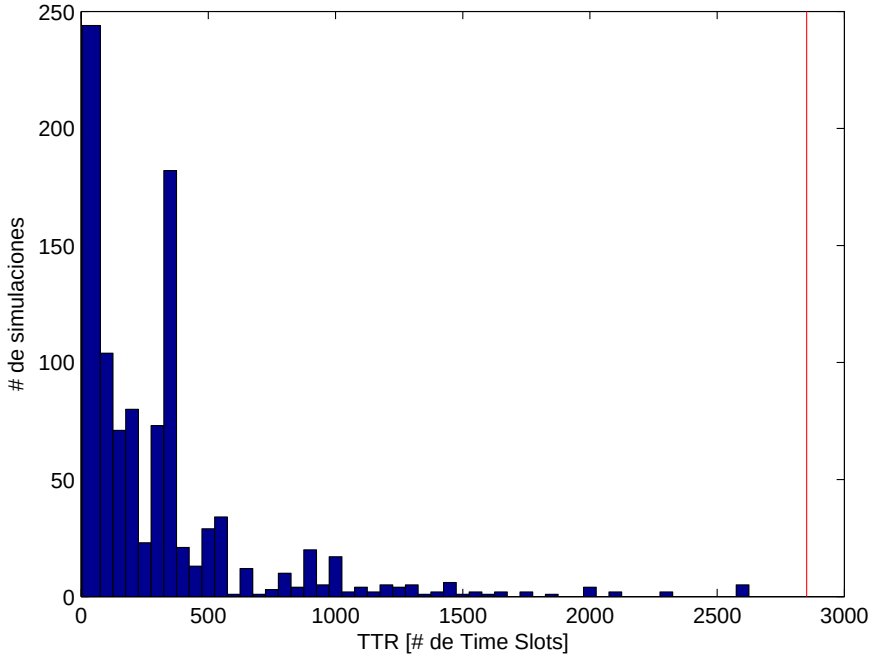


Figura 3.8: Histograma tomando que 20 de 30 canales se encuentran no disponibles

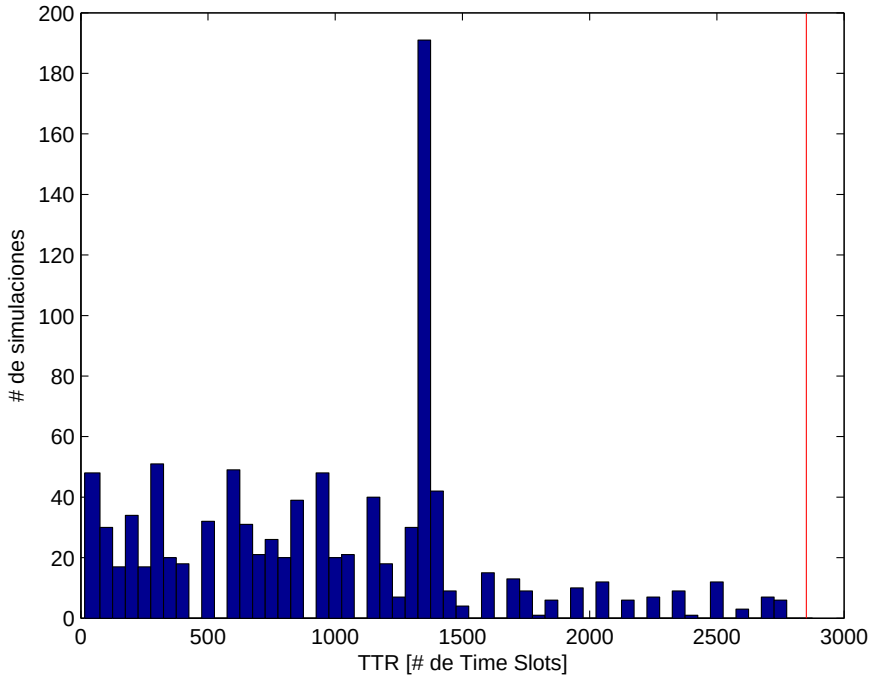


Figura 3.9: Histograma tomando que 29 de 30 canales se encuentran no disponibles

La línea vertical roja se encuentra en el límite teórico para el cual se garantiza que se dará el *Rendezvous* bajo un modelo asimétrico.

En función de estas simulaciones se muestra en la tabla 3.1 los resultados numéricos, correspondientes a cada caso, para el TTR promedio y MTTR.

Canales no disponibles para RV	TTR promedio	MTTR
1	95,4570	2120
10	113,8960	1949
20	330,5460	2579
29	979,8120	2775

Tabla 3.1: Desempeño de CRSEQ bajo modelo a utilizar

3.3. MCA

3.3.1. Descripción

Como se describe en [14] y [15], ***MCA*** (*Modular Clock Algorithm*) es un algoritmo que utiliza aritmética modular basada en números primos para **garantizar el *Rendezvous*** bajo ciertos modelos.

Primero que nada se añade un notación adicional a la presentada en la sección 3.1.1

- $\mathbf{r_i}$: tasa en la cual la radio cognitiva i salta de canal en canal. Cada TS la radio i salta r_i canales hacia adelante.

A continuación se presenta el algoritmo propuesto por los autores:

Algoritmo 1: Modular Clock Algorithm

```
1  Encontrar M_i, el número de canales disponibles
2  Calcular P_i, el primer primo mayor a M_i
3  j_i^0 = rand[0, M_i)
4  while No se de el rendezvous do
5      Elegir r_i en [0, P_i) de forma aleatoria
6      for t = 0 to 2P_i do
7          j_i^{t+1} = (j_i^t + r_i) mod P_i
8          if j_i^{t+1} < M_i then
9              c = c_{i, j_i^{t+1}}
10         else
11             c = c_{i, (j_i^{t+1} mod M_i)}
12         Probar rendezvous en el canal c
```

Desarrollando el algoritmo se obtiene lo siguiente. En primer lugar se obtiene el valor de M_i, se calcula P_i, j_i^0 (primer índice de canal) y r_i. Seguidamente, en cada *TS* del sistema (t), se modifica el índice j según la igualdad $j_i^{t+1} = \left(j_i^t + r_i\right) mod(P_i)$, y si luego de $2P_i$ *TS* no se llega al *Rendezvous* se toma otro valor de r_i. Luego de apreciar el Teoriema 3, la Proposición 1 y el análisis de tiempos que se desarrolla en el capítulo 3.3.2 es que se logra entender el hecho que se tome un nuevo valor para r_i luego de $2P_i$ *TS*. Para finalizar se elige el canal $c = c_j$ (en el cual se va a intentar el *Rendezvous*), de tal manera que si $j_i^{t+1} < M_i \Rightarrow j = j_i^{t+1}$ y de lo contrario $j = j_i^{t+1} \text{mod}(M_i)$ (esta elección particular es para asegurar que $j \in [0, M_i)$).

Se puede observar que entre la línea 6 y la línea 12, el índice de canal del radio i que se encuentra en un *TS* t, se puede expresar como:

$$\boxed{j_i^t = t \cdot r_i + j_i^0 \bmod P_i} \tag{3.1}$$

Es necesario tener presente la ecuación 3.1 de aquí en más, ya es que fundamental en el algoritmo.

3.3.2. Análisis de desempeño del Algoritmo

Debido a que el estudio de este algoritmo se basa fuertemente en lo propuesto por los autores de [14], se deben aclarar un par de conceptos. En el trabajo anteriormente mencionado, se definen una serie de modelos, de los cuales se utilizan dos, "*Shared Model*" e "*Individual Model*". Si bien están definidos con mayor profundidad son muy similares a los que se especifican en este trabajo como *Modelo Simétrico* y *Modelo Asimétrico* respectivamente.

Sin más preámbulos se comienza con una serie de teoremas y proposiciones que garantizan o no la convergencia de la ecuación 3.1.

Teorema 3 *Bajo el "Modelo Simétrico" cuando dos radios realizan el Algoritmo 1 con $r_1 \neq r_2$, se llegará al Rendezvous, y el mismo ocurrirá en menos de P_i TS.*[1]

Proposición 1 *Bajo el "Modelo Simétrico" cuando dos radios realizan el MCA simplemente utilizando la ecuación 3.1 con $r_1 = r_2$ y $j_1^0 \neq j_2^0$, nunca se dará el rendezvous.*[2]

Luego de haber sido presentados el Teorema 3 y la Proposición 1, y volviendo al Algoritmo 1 se procede a explicar la condición de "*time-out* " ubicada entre las líneas 6 y 12 del mismo. Se supone que una radio comienza a correr el algoritmo en $t = 0$ y la otra en $t = P - \epsilon$. Como ambas radios modifican sus respectivos valores de r_i cada $2P$ *TS*, durante $P + \epsilon$ *TS* los mismos se van a mantener constantes (en caso de ser distintos por el Teorema 3 se va a lograr el *Rendezvous*). Otra posibilidad es que una radio comience a correr el Algoritmo 1 en $t = 0$ y la otra en $t = P + \epsilon$. En la figura 3.10 se puede apreciar este ejemplo. Si se toma $\hat{t} = 0$ el momento en que la radio 2 comienza a correr el agoritmo, tenemos que en $\hat{t} = P - \epsilon$ la radio 1 resetea r_1, con lo que se obtiene un caso similar al anterior.

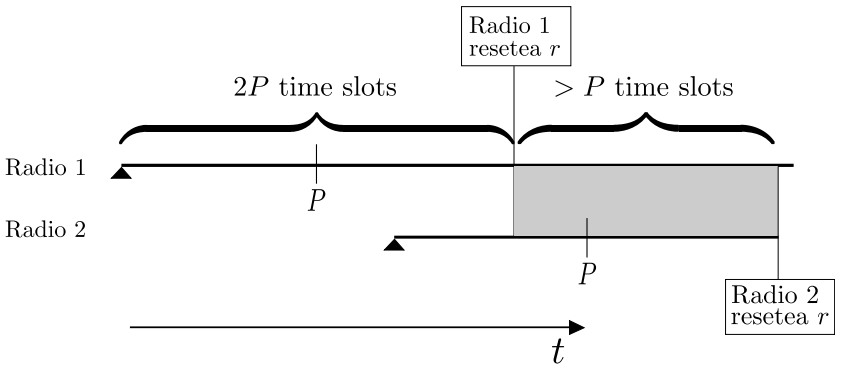


Figura 3.10: Utilizando el reset en $2P$ time slots para garantizar el rendezvous

A continuación se quita la hipótesis de que ambas radios tienen la misma cantidad de canales disponibles, encontrándose entonces frente al Modelo Asimétrico (o Individual Model para los autores de [14]). En primer instancia lo único que suponemos bajo este modelo es que $M_i > 0$

Teorema 4 *Bajo el "Modelo Asimétrico" cuando dos radios realizan el MCA simplemente siguiendo la ecuacón 3.1, con $P_1 \neq P_2$, el rendezvous ocurrirá dentro de $P_1 P_2$ TS.*[3]

[1] La demostración completa de este teorema se encuentra en [14]
[2] La demostración completa de esta proposición se encuentra en [14]
[3] La demostración completa de este teorema se encuentra en [14]

Proposición 2 *Bajo el "Modelo Asimétrico" cuando dos radios realizan el MCA simplemente siguiendo la ecuacón 3.1, con* $C_1 \neq C_2$ *y* $P_1 = P_2$*, no es posible garantizar el rendezvous.*

3.3.3. Análisis de desempeño bajo el modelo definido

Hasta aquí, basados en una serie de teoremas y proposiciones, se han presentado ciertos escenarios en los cuales el algoritmo converge y en cuales no. También se describe el efecto de que una radio comience a correr el algoritmo un número de *TS* posterior a la otra, lo que de aquí en más se ha de llamar desfasaje. Para estudiar este efecto se simulan en *MatLab* 4 posibles escenarios en los cuales se desfasa el comienzo del algoritmo de la radio 2 respecto a la 1. Estos casos son, desfasaje de 4 *TS*, 8 *TS*, 16 *TS* y 25 *TS*. A partir del análisis de los casos se realiza una comparación con el escenario donde no hay desfasaje. Para realizar las simulaciones se utilizan los parámetros descriptos en 3.1.1 pero se añade la hipótesis que todos los canales están libres. Con esto último se puede enfocar en la influencia del desfasaje solamente, ya que el desempeño bajo la influencia que algunos canales no se encuentren disponibles al momento de intentar el *Rendezvous* será estudiado más adelante.

Se obtuvieron los siguientes resultados.

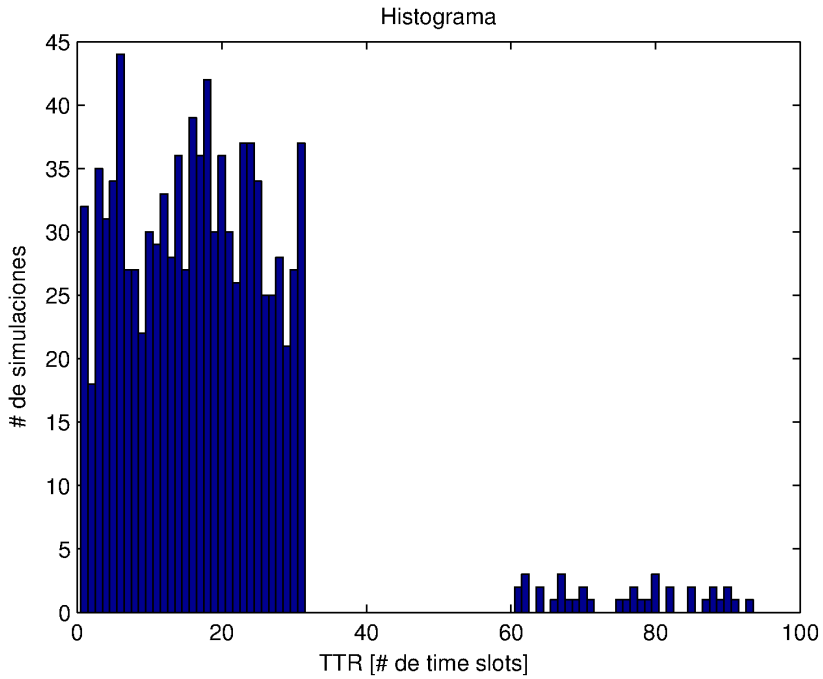


Figura 3.11: Histograma con un retardo de 4 time slot

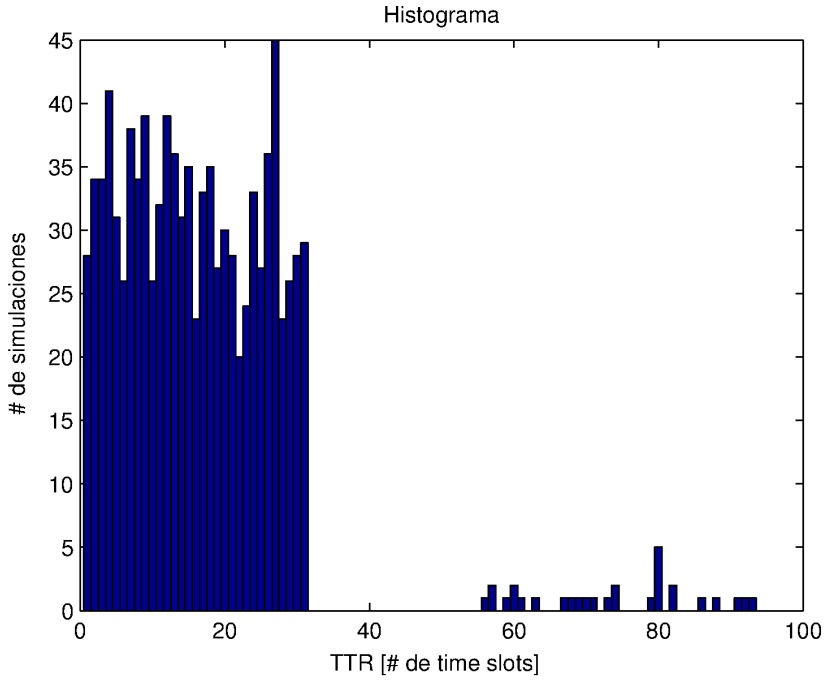


Figura 3.12: Histograma con un retardo de 8 time slot

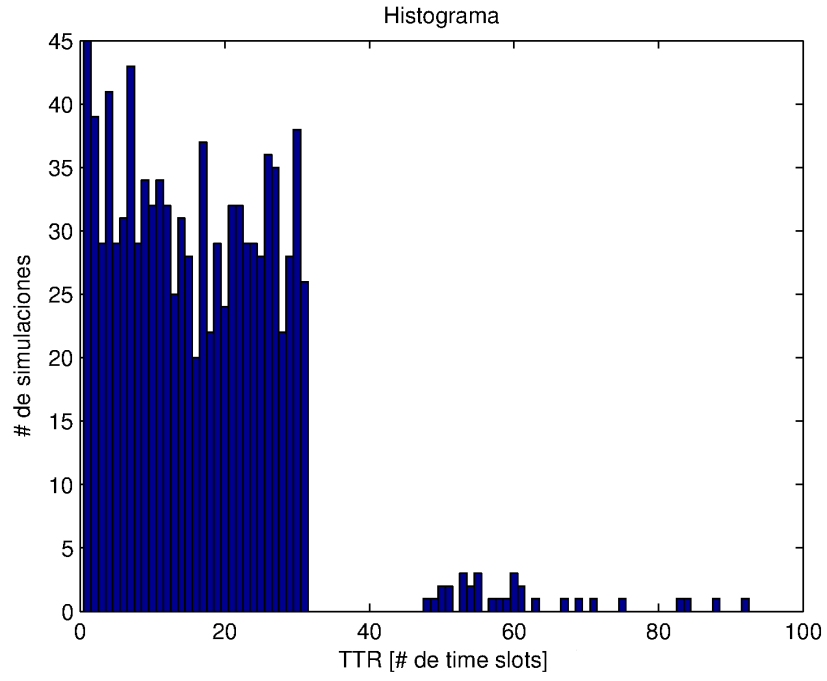


Figura 3.13: Histograma con un retardo de 16 time slot

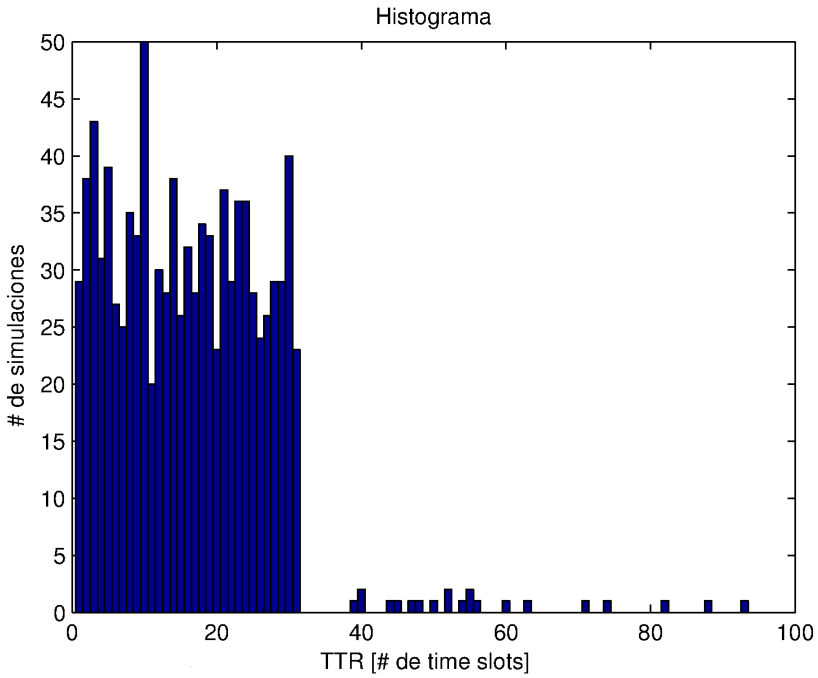


Figura 3.14: Histograma con un retardo de 25 time slot

Como se puede apreciar en las simulaciones anteriores, la mayoría de las veces se logra el *Rendezvous* en menos de 40 *TS*. Realizando un estudio más a exahustivo se encuentra que en el caso que no hay retardo y todos los canales están libres, el 95 % de las veces se logra el *Rendezvous* en 31 *TS* o menos, por lo cual parece un buen umbral para realizar el estudio. Más especificamente, al calcular el porcentaje de éxito para los diferentes retardos en hasta 31 *TS*, se obtiene la tabla 3.2.

Retardo	Porcentaje de éxito
0	97,1 %
4	96,3 %
8	97,7 %
16	96,9 %
25	97,9 %

Tabla 3.2: MCA: porcentaje de éxito para diferentes retardos

A partir de la tabla 3.2 se puede concluir que el retraso de una radio respecto a la otra no parece ser un factor determinante en el desempeño del algoritmo. Continuando con el estudio y basándose en el modelo propuesto en la sección 3.1.1 se obtienen los siguientes resultados.

Canales no disponibles	TTR promedio	TTR max	TTR$< P$
0	17,4090	89	97,1 %
1	19,4900	148	93,8 %
10	44,1140	322	68,7 %
20	110,3910	764	39,8 %
29	1401,1	8501	3,2 %

Tabla 3.3: Desempeño de MCA bajo modelo a utilizar

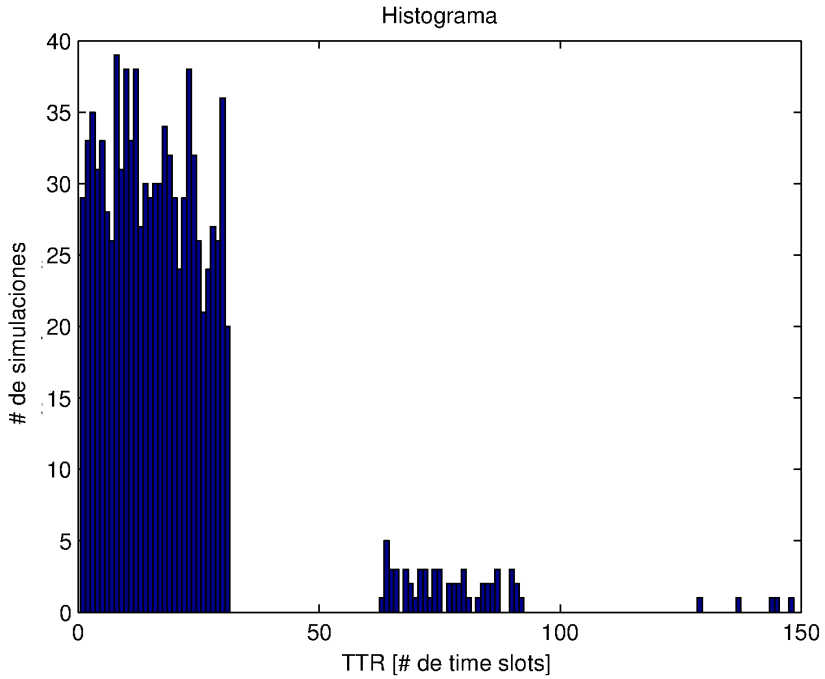


Figura 3.15: Histograma tomando como que 1 de 30 canales se encuentra no disponible

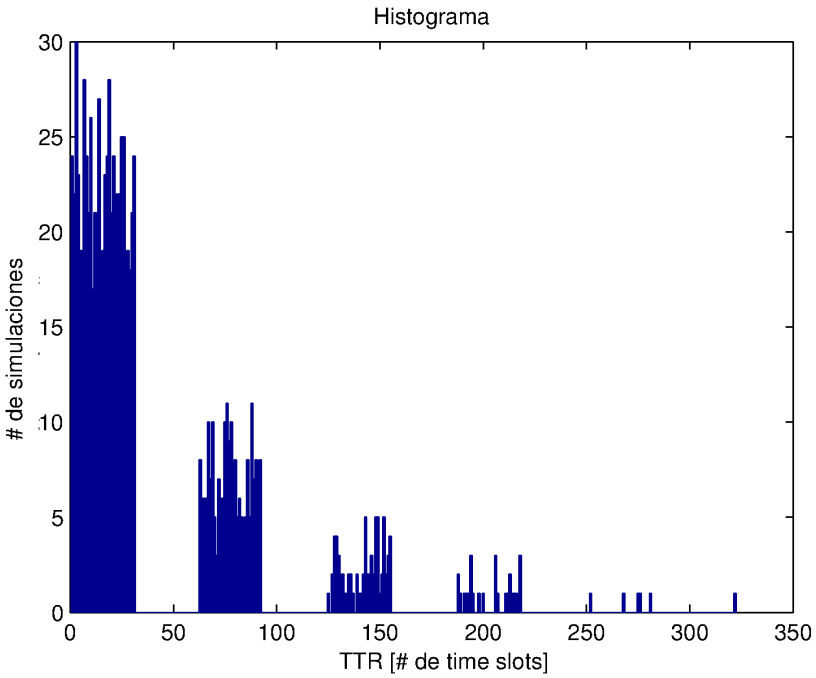


Figura 3.16: Histograma tomando que 10 de 30 canales se encuentran no disponibles

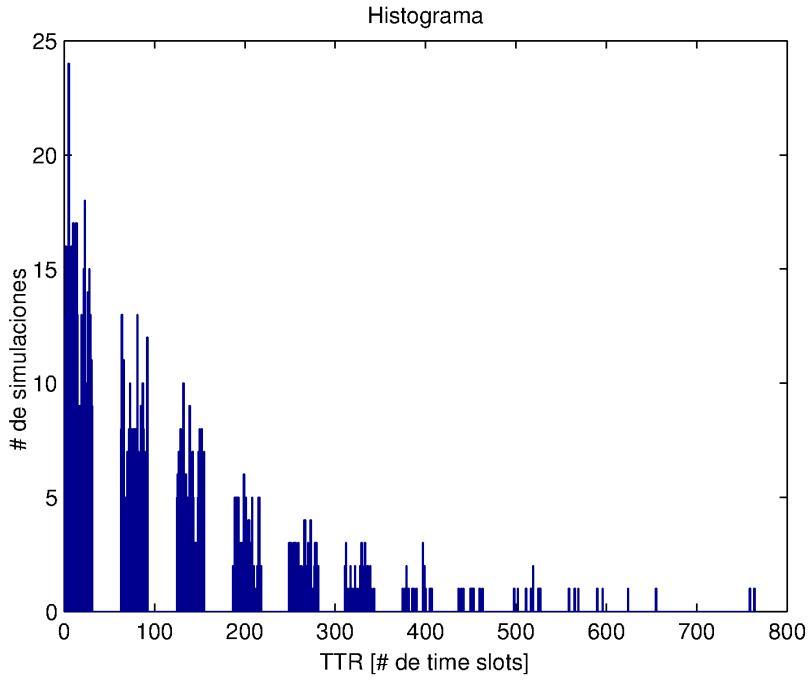


Figura 3.17: Histograma tomando que 20 de 30 canales se encuentran no disponibles

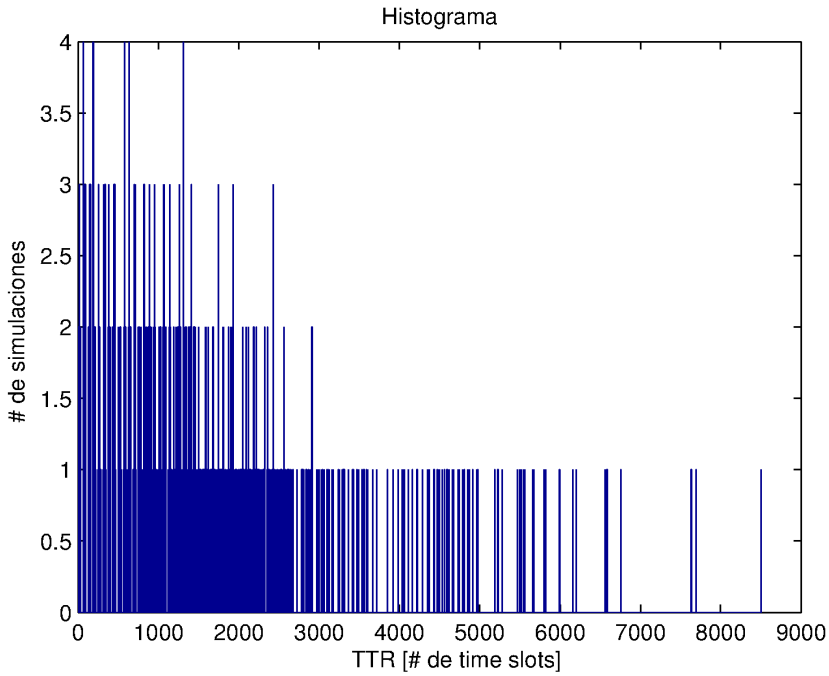


Figura 3.18: Histograma tomando que 29 de 30 canales se encuentran no disponibles

3.4. MMCA

3.4.1. Descripción

Para evitar el problema presentado en la Proposición 2, los autores en [14] proponen una modificación al algoritmo anterior, llegando así al algoritmo ***MMCA***(*Modified Modular Clock Algorithm*). El nuevo algoritmo se presenta a continuación.

Algoritmo 2: Modified Modular Clock Algorithm

```
1  Encontrar M_i, el número de canales disponibles
2  j_i^0 = rand[0, M_i)
3  while No se de el rendezvous do
4      Elegir r_i en [0, M_i) de forma aleatoria
5      Elegir el número primo p_i en [M_i, 2M_i] de forma aleatoria
6      for t = 0 to 2P_i^2 do
7          j_i^{t+1} = (j_i^t + r_i) mod P_i
8          if j_i^{t+1<M_i} then
9              c = c_{i,j_i^{t+1}}
10         else
11             c = c_{i,(j_i^{t+1} mod M_i)}
12         Probar rendezvous en el canal c
```

En el Algoritmo 2 el número primo P_i es elegido aleatoriamente en $[M_i, 2M_i]$ cada vez que pasen $2{P_i}^2$ *TS* sin llegar al *Rendezvous*.

3.4.2. Análisis de desempeño bajo el modelo definido

Al igual que para el *MCA* se ve el efecto de retraso de una radio frente a la otra. A primera instancia se esperaría que no fuera un factor determinante en el desempeño del algoritmo ya que no lo era en el algoritmo 1. En los histogramas a continuación se muestra el fragmento mas representativo de los mismos, ya que el MTTR se da en valores muy alejados del cero.

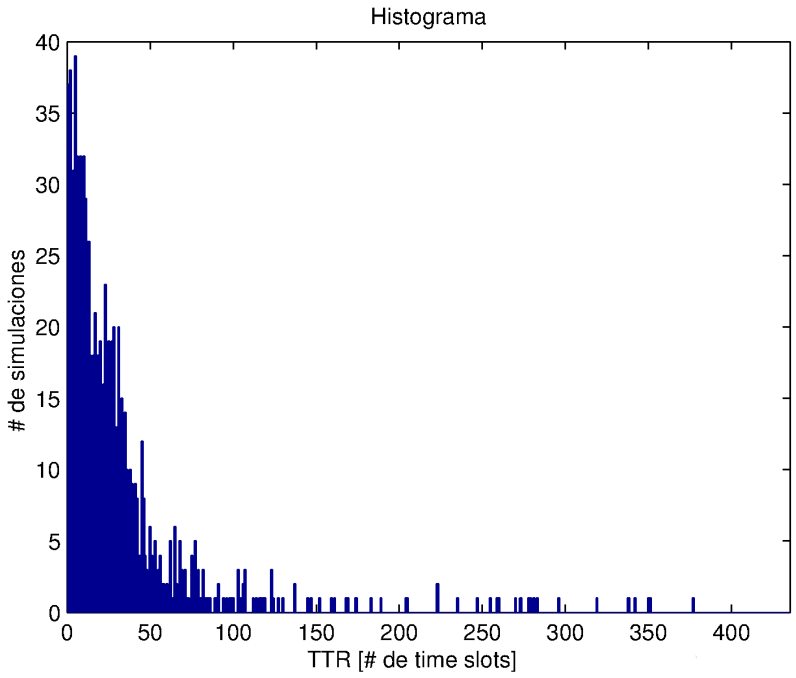


Figura 3.19: Histograma para 4 time slots de retardo

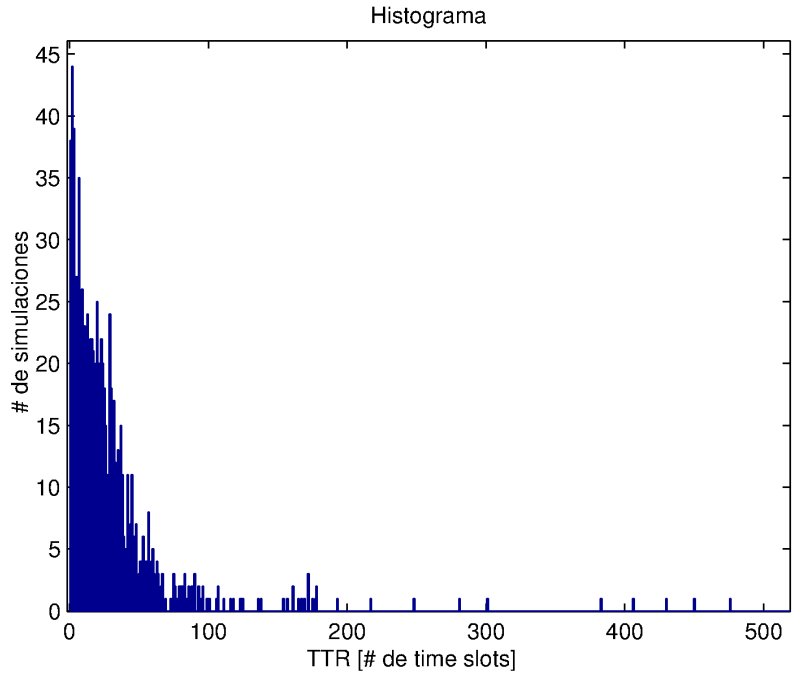


Figura 3.20: Histogramas para 16 time slot de retardo

Igual que para el caso de *MCA*, se fija un umbral (en este caso 200). Para el

caso en que no hay retardo y todos los canales están libres se logra el *Rendezvous* en un número mayor al 95 % de las veces. Tomando esto en cuenta se obtiene la tabla 3.4

Retardo	Porcentaje de éxito
0	97,3 %
4	96,7 %
8	97,6 %
16	98,1 %
25	96,7 %

Tabla 3.4: MMCA: porcentaje de éxito para diferentes retardos

Se puede concluir que el retraso de una radio respecto a la otra no parece ser un factor determinante en el desempeño del algoritmo.

Descartando eso, se simula en *MatLab* basándose en las hipótesis presentadas en 3.1.1.

Se obtiene lo siguiente.

Canales no disponibles	TTR promedio	MTTR	TTR< 200
0	56,5530	4419	97,3 %
1	61,5590	7068	96,2 %
10	234,8680	10663	92,9 %
20	641,9970	21161	79,8 %
29	1112,4	13674	30,8 %

Tabla 3.5: Desempeño de MCA bajo modelo a utilizar

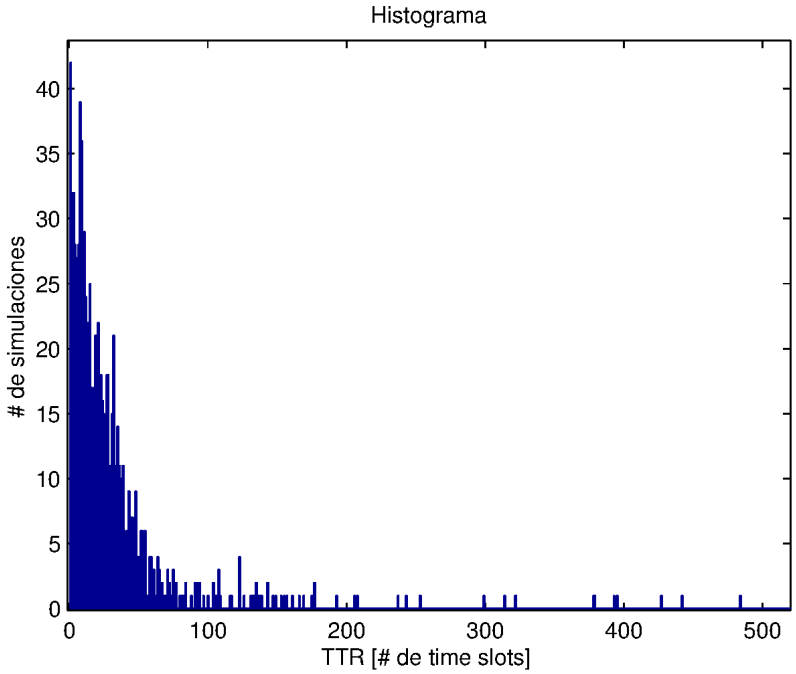


Figura 3.21: Histograma tomando que 1 de 30 canales se encuentra no disponible

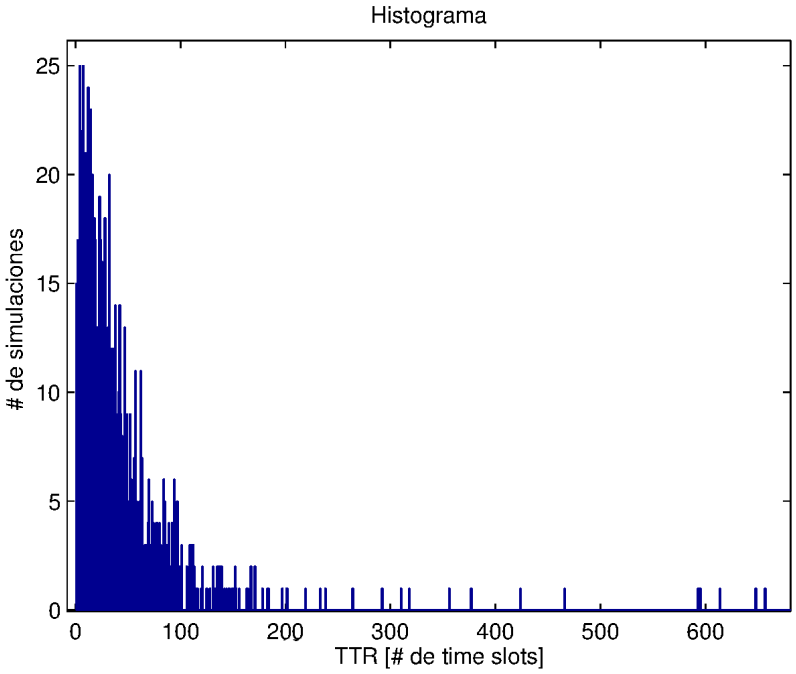


Figura 3.22: Histograma tomando que 10 de 30 canales se encuentran no disponibles

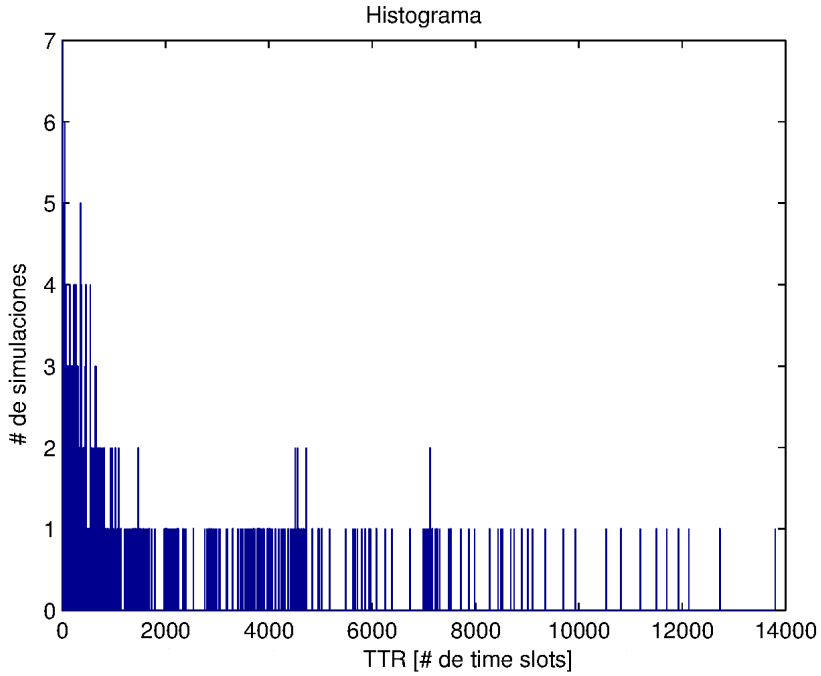


Figura 3.23: Histograma tomando que 29 de 30 canales se encuentran no disponibles

3.5. Jump Stay

El algoritmo tratado en esta sección es ***Jump-Stay***, propuesto en [16] y [10] por algunos de los autores más respetados en el área de las Redes Cognitivas. Dicho algoritmo garantiza el encuentro en redes con dos usuarios o múltiples usuarios, así como también bajo el modelo simétrico como bajo el modelo asimétrico.

En este estudio, se considera una red cognitiva constituida por dos usuarios secundarios ($K = 2$), con uno o más PU en la misma área geográfica. Los PU son los licenciatarios del espectro, el cual se divide en M ($M > 1$) canales. Estos canales están indexados como $1, 2, \ldots, M$ y estos índices son conocidos por todos los usuarios. También se considera el conjunto de canales potencialmente disponibles como $C = c_1, \ldots, c_M$. Al conjunto de canales disponibles para el usuario i se le llama C_i, entonces C_i está incluido en C.

Por lo tanto si se está trabajando bajo el modelo simétrico, ambos usuarios tienen el mismo conjunto de canales disponibles, entonces $C_1 = C_2 = C$. Si se trabajara bajo el modelo asimétrico se debe calcular el número de canales disponibles en común, esto es $G = |\cap_{(i=1)}^{K} C_i|$.

Se asume como hipótesis que se cuenta con un sistema de tiempo ranurado (*time-slotted system*). Nótese que la coordinación entre usuarios es difícil de lograr antes del encuentro y cada usuario puede comenzar su secuencia de CH en diferente momento, por lo tanto en [16] se propone que este algoritmo no requiere sincronización de tiempos.

3.5.1. Descripción

La secuencia de *CH* es generada en rondas, consitiendo cada una en un **período de salto** ($jump - pattern$) y en un **período de permanencia** ($stay - pattern$), en ese orden. Intuitivamente se entiende que durante el período de salto los usuarios cambian de canal continuamente, mientras que en el período de permanencia se mantienen esperando en un mismo canal.

En cada ronda el período de salto consta de **2P** *TS* mientras que el período de permanencia es de **P** *TS*, por lo tanto cada ronda llevara 3P *TS* en total. En el período de salto los usuarios inician la secuencia con índice i_0 y se mantiene cambiando de canal con una longitud de paso de r_0 (r_0 tasa en la cual la radio cognitiva salta de canal en canal) mediante el uso de las operaciones de módulo en P. Por otro lado en el período de permanencia el usuario se mantiene en el canal i_0. De esta manera el algoritmo compara el número de *TS* con $2P$, de ser menor se está en período de salto, y entonces se ejecuta un cambio de canal.

3.5.2. Algoritmo

La implementación de este algoritmo se basa en una función. Esta se muestra a continuación y es la que se encarga de decidir hacia dónde es el próximo salto.

Algoritmo 3: Función JSHopping

```
1  Input: M, P, r, i, t
2  Output: canal c
3  t = t mod(3P)
4  if (t < 2P) then
5      j = ((i + tr − 1)modP) + 1
6  else
7      j = r
8  if j > M then
9      j = ((j − 1)modM) + 1
10 Return c = c_j
```

A modo de explicación, la función *JS Hopping* (3) verifica si se encuentra en un periodo de salto o en un periodo de permanencia. Y a partir de esta verificación, se adjudica el canal al que se saltará o no. Y la misma se ejecuta dentro del algoritmo para obtener el canal al que se debe saltar, como se muestra en 4

Algoritmo 4: Algoritmo JS_2

```
1  Input: M, C_k
2  P = nextprime(M)[a]
3  r_0 = rand[1, M]
4  i_0 = rand[1, M]
5  t = 0
6  while no se de el Rendezvous do
7      n = floor(t/3P)
8      r = ((r_0 + n − 1) mod M) + 1
9      c = JSHopping(M, P, r, i_0, t)
10     t = t + 1
11     Intentar Rendezvous en el canal c
```

[a] la función $nextprime(x)$ devuelve el menor número primo mayor que x.

3.5.3. Análisis de desempeño del Algoritmo

Un punto a favor de este algoritmo es que no necesita sincronización de tiempo, esto quiere decir que la ronda de *CH* puede comenzar en distinto momento para cada usuario. En el peor de los casos los autores garantizan un **TTR** $\leq$ **3P**, el cual se da cuando un usuario está en el período de permanencia y el otro comienza su ronda de *CH*. De esta manera el solapamiento entre el período de permanencia y el período de salto es menor que P y $r_1 = r_2$.

En este análisis se contempla principalmente los casos en los que el usuario 2 comienza su ronda de *CH* cuando el usuario 1 se encuentra en el período de salto. Se considera como hipótesis razonable el hecho que el usuario 2 comience su ronda de *CH* como máximo 2P *TS* más tarde que el usuario 1, aunque verdaderamente va a depender del sistema de sensado y cuánto demora el usuario 2 en determinar que el usuario 1 ya no se encuentra presente en el canal.

Ahora, este caso se puede dividir en 3 clases. Para realizar esta división se debe definir l como el solapamiento entre los períodos de salto de ambos usuarios y r_i la tasa en la cual la radio cognitiva i salta de canal en canal. Por lo tanto los posibles casos con sus respectivos ***MTTR*** son:

a) $l \geq P, r_1 \neq r_2 \Rightarrow TTR \leq P$ (3.24)

b) $l \geq P, r_1 = r_2 \Rightarrow TTR \leq 2P + 1$ (3.25)

c) $l < P \qquad \Rightarrow TTR \leq 2P$ (3.26

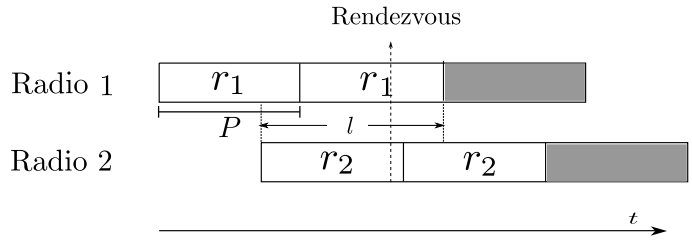


Figura 3.24: Clase A del Caso 1

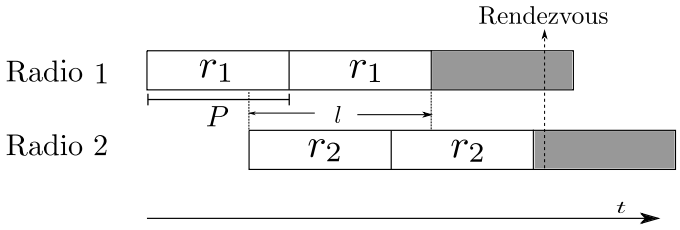


Figura 3.25: Clase B del Caso 1

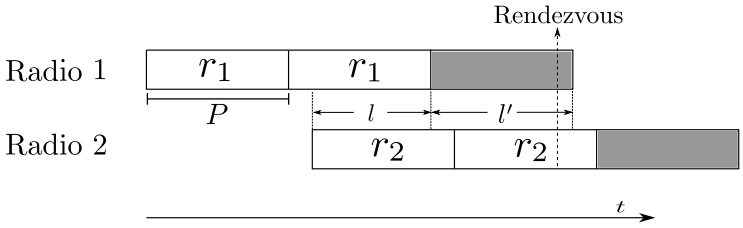


Figura 3.26: Clase C del Caso 1

Como se puede apreciar, en cada caso los autores especifican la cota para el TTR, lo cual se comprueba con simulaciones en *MatLab* para el primer caso de los tres presentados anteriormente. Para realizar esto se considera $M = 10$ y se comienza la ronda de CH desde el mismo canal (esto es $i_{01} = i_{02}$), esto permite simular el caso en que la comunicación está en curso cuando es necesario el cambio de canal. Para verificar la cota de TTR ($MTTR$) se corre el algoritmo 2000 veces y se construye un histograma:

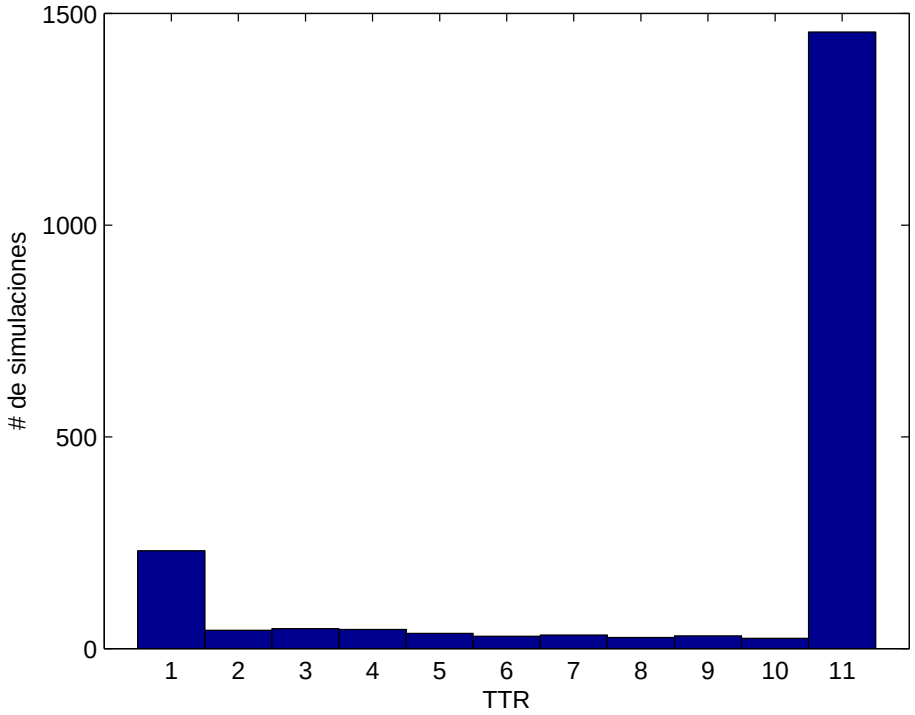


Figura 3.27: Histograma JS

En la figura 3.27 se aprecia claramente que la gran mayoría de las simulaciones se dan para un TTR= 11, lo cual es coherente con el valor propuesto por los autores. En este caso aseguran $TTR \leq P = 11$.

El otro caso se da cuando el usuario 2 comienza la ronda de CH mientras el usuario 1 se encuentra en el período de permanencia. Este caso, de forma similar al anterior, también se puede dividir en 3 clases. Para estas 3 clases los autores de [16] también calculan su $MTTR$, pero en este caso se define l como el solapamiento entre el período de permanencia del usuario 1 y el período de salto del usuario 2; y también se define l' como el solapamiento entre los períodos de salto de ambos usuarios.

a) $l = P,\ r_1 \neq r_2 \qquad \Rightarrow TTR \leq P$ (3.28)

b) $l > P,\ l' > P, r_1 \neq r_2 \Rightarrow TTR \leq 2P$ (3.29)

c) $l < P,\ r_1 \neq r_2 \qquad \Rightarrow TTR \leq 3P$ (3.30)

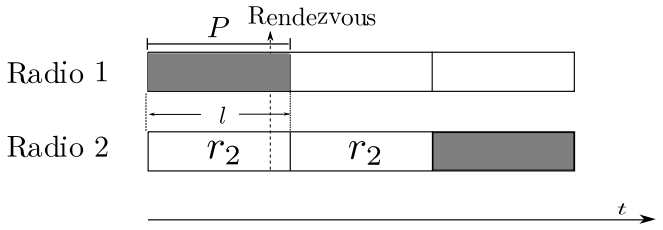


Figura 3.28: Clase A del Caso 2

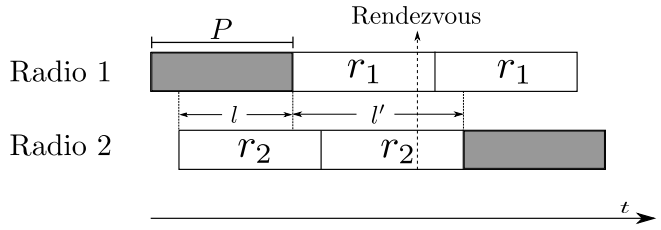


Figura 3.29: Clase B del Caso 2

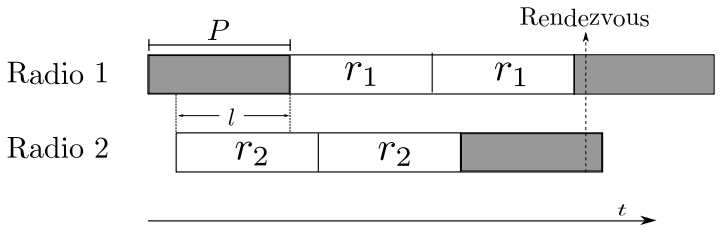


Figura 3.30: Clase C del Caso 2

Como se puede apreciar para estas tres clases se da la mayor demora para que se produzca el encuentro. Por lo tanto para terminar de probar el desempeño del algoritmo se simuló el caso 2.c en *MatLab*.

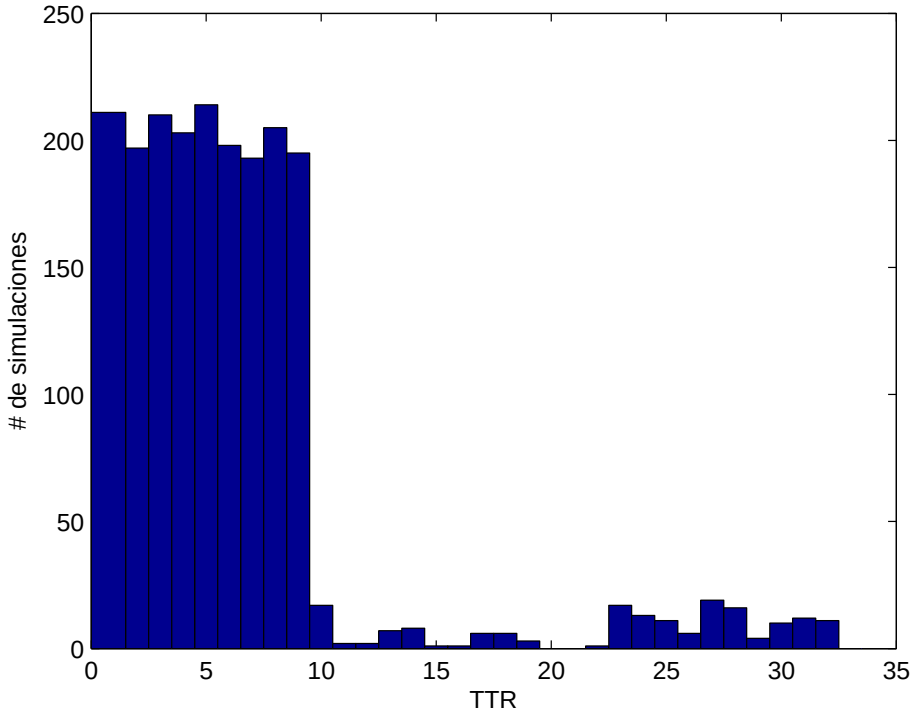


Figura 3.31: Histogrma JS Stay-Jump

En el histograma 3.31 se puede observar el desempeño del algoritmo *Jump-Stay* en las condiciones donde peor responde, comprobando así los límites de tiempo dados por los autores del algoritmo y verificando que para la gran mayoría de los casos el *Rendezvous* se resuelve en un tiempo sensiblemente menor a la cota dada. Para verificar que la mayoría de los casos tienen una cota de TTR menor a 3P, se ejecutan las simulaciones varias veces, obteniéndose que el 90 % de los casos tiene como cota P.

3.5.4. Análisis de desempeño bajo el modelo definido

Como se explica en el comienzo del capítulo, por el alcance del proyecto el estudio se centra principalmente en el modelo simétrico, pero los desarrolladores del algoritmo presentan resultados bajo el modelo asimétrico también. Los resultados no son tan buenos como se esperaría pero son suficientemente buenos para garantizar el *Rendezvous*, lo cual es importante bajo un sistema con modelo asimétrico, ya que la gran mayoría de los algoritmos existentes no puede garantizar el encuentro bajo este contexto.

En la sección 3.1.1 se describe el modelo en el que se implementarán los algoritmos, y aunque el modelo es simétrico, el desempeño no será tan bueno como los resultados exhibidos ya que se simularán canales ocupados para darle un marco de realidad al proyecto.

Para medir el desempeño del algoritmo bajo las hipótesis del modelo definido se simula en *MatLab* el *Rendezvous* pero de forma gradual se quitan canales de la lista de los 30 posibles canales disponibles.

Luego de realizar la simulación en dicho modelo con el algoritmo para el modelo simétrico, se observa que para algunos casos el desempeño es muy malo, incluso en algunos casos no llegando al encuentro. Se encuentra que el algoritmo genera una secuencia periódica de período $3P$ lo cual no se había apreciado en el modelo sin quitar canales, esto debido a que el encuentro siempre se da dentro del primer período. Se modifica el algoritmo para que cada vez que se cumpla una ronda ($t = 3P$) se calcule nuevamente el r_0, logrando que la secuencia deje de ser periódica y consiguiendo un mejor desempeño. Con este cambio el algoritmo queda de la siguiente manera:

Algoritmo 5: Algoritmo JS_2

```
1  Input: M, C_k
2  P = nextprime(M)
3  r_0 = rand[1, M]
4  i_0 = rand[1, M]
5  t = 0
6  while no se de el Rendezvous do
7      if mod(t, 3P) = 0 then
8          r_0 = rand[1, M]
9      n = floor(t/3P)
10     r = ((r_0 + n − 1) mod P) + 1
11     c = JSHopping(M, P, r, i_0, t)
12     t = t + 1
13     Probar rendezvous en el canal c
```

Es válido aclarar que con las modificaciones realizadas el algoritmo es muy similar al propuesto por los autores para el modelo asimétrico, pero tiene mejor desempeño en el modelo presentado en la sección 3.1.1

Seguidamente se muestra el desempeño del algoritmo modificado en el modelo definido, para esto se presentan histogramas en las condiciones explicadas en el comienzo de este documento.

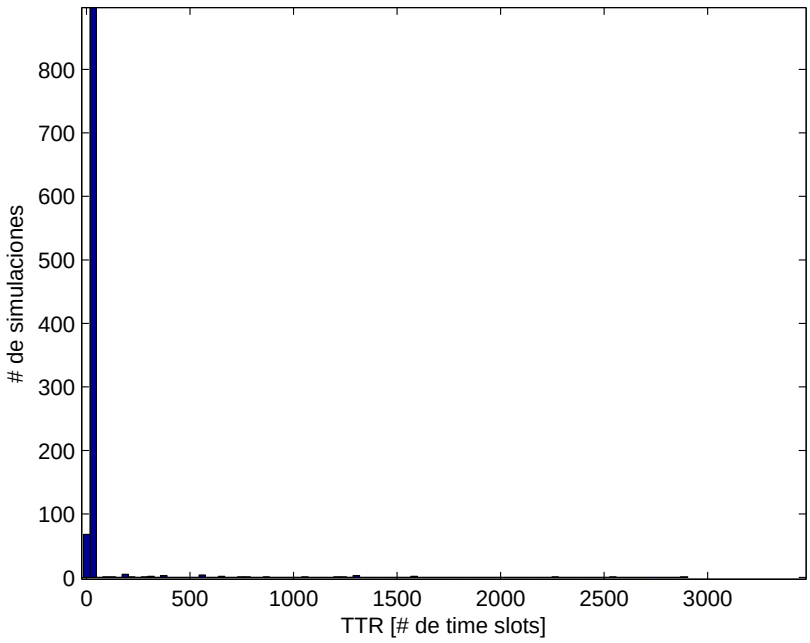


Figura 3.32: Histograma tomando que 1 de 30 canales se encuentra no disponible

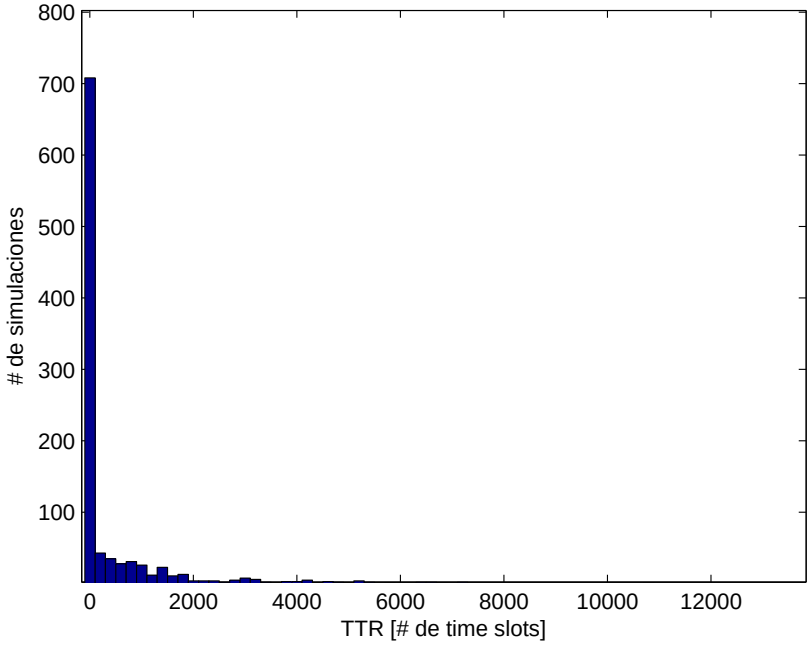


Figura 3.33: Histograma tomando que 10 de 30 canales se encuentran no disponibles

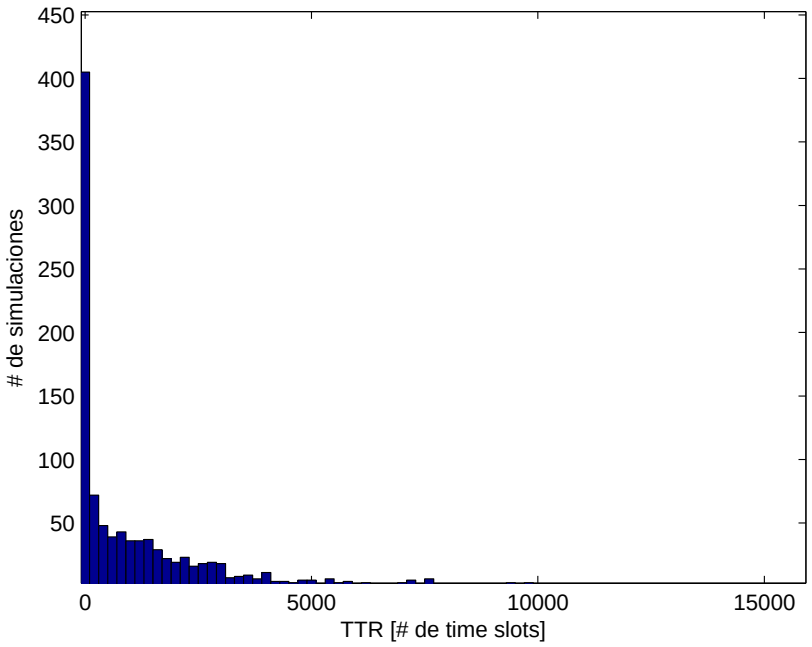


Figura 3.34: Histograma tomando que 20 de 30 canales se encuentran no disponibles

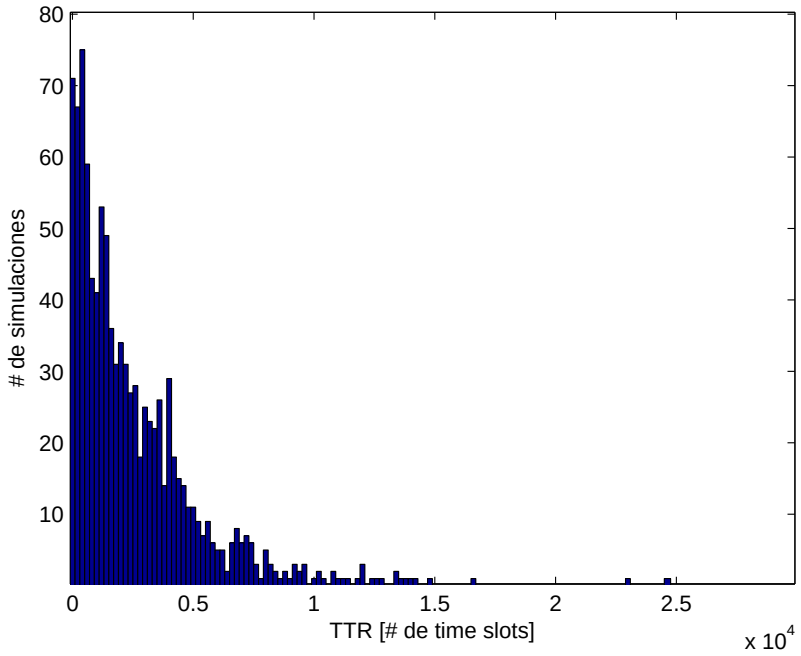


Figura 3.35: Histograma tomando que 29 de 30 canales se encuentran no disponibles

Según los autores el algoritmo tiene como cota superior para el modelo asimétri-

co $3MP(P-G)+3P$ donde G es la cantidad de canales en común, por lo tanto esta cota seria de $MTTR$= 83793 para el caso en que solo se deja un canal libre y de $MTTR$= 2883 para el caso en que solo se quita un canal de la lista. Aunque se sabe que este modelo no es exactamente asimétrico se supone que el desempeño debería ser mejor, por lo que estos valores sirven como cota para evaluar el desempeño.

Otro valor importante es la cantidad de ejecuciones en las que se da el Rendezvous dentro del primer período del algoritmo, esto es, qué porcentaje de las 1000 ejecuciones tuvieron un TTR menor a $3P$ (TTR< 93, para $M = 30$).

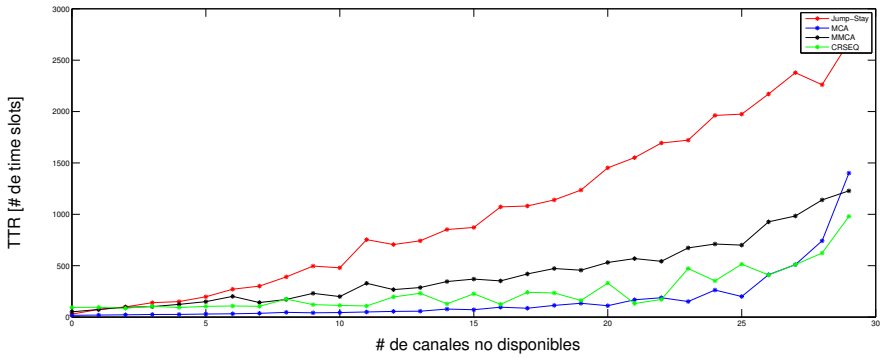
Teniendo en cuenta lo anterior se observa que para los cuatro casos se tiene un TTR menor a la cota. La tabla 3.6 muestra los resultados.

Canales no disponibles	Cota teórica	TTR promedio	MTTR	TTR< $3P$
0	− − −	29,2693	31	100 %
1	2883	52,6847	2803	97,0 %
10	30703	474,3784	10606	71,9 %
20	58683	1272,5	13116	38,2 %
29	83793	2571,4	24763	6,7 %

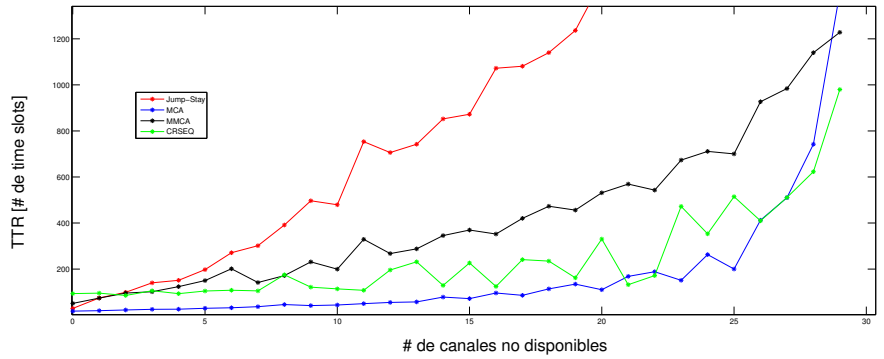
Tabla 3.6: Desempeño de JS bajo modelo definido

3.6. Comparación de Algoritmos

En secciones anteriores se presentaron diferentes parámetros para medir el desempeño de los algoritmos. Uno de ellos es el valor medio del TTR.



(a) número canales no disponibles Vs TTR



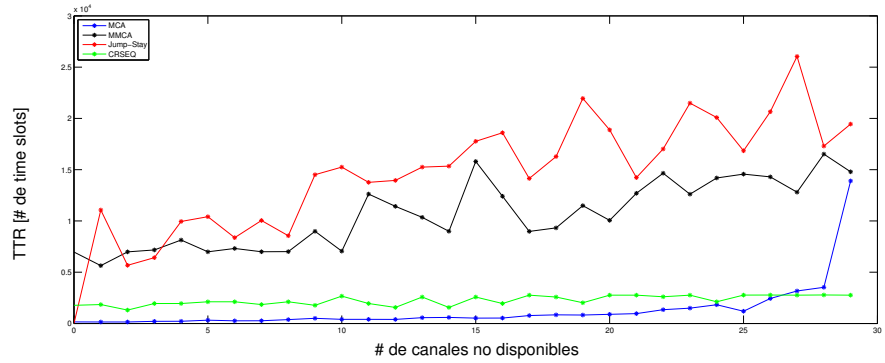
(b) zoom gráfica (a)

Figura 3.36: Valor medio de TTR para distintos canales no disponibles

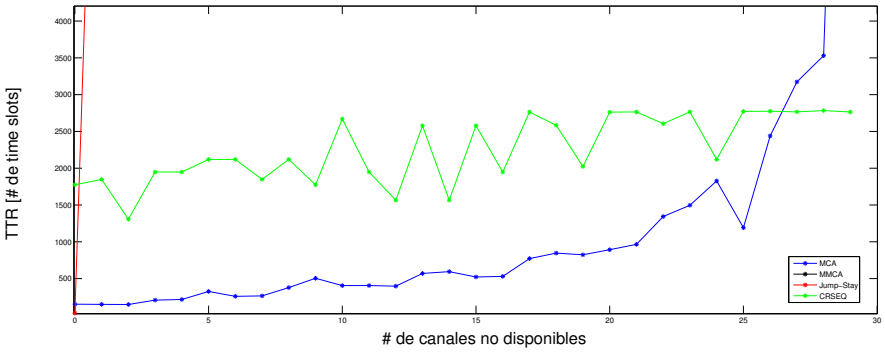
Para obtener los valores medios de TTR mostrados en la figura 3.36 se simularon en *MatLab* 30 mil ejecuciones para cada algoritmo. Todas estas ejecuciones se reparten en 30 casos, donde progresivamente se van dejando menos canales disponibles para realizar el rendezvous. Así, se realizan mil ejecuciones con 30 canales disponibles, mil con 29, mil con 28 y así sucesivamente hasta tener un único canal disponible.

En las gráficas de la figura 3.36 se puede apreciar cómo el algoritmo MCA tiene un muy buen desempeño (desde el punto de vista de valor medio de TTR), superando en casi todos los casos al resto de los algoritmos simulados. Vale la pena destacar que el hecho que supere a MMCA a primera instancia no era de esperarse. Según los autores de [14] la modificación a MCA se basa en obtener un mejor rendimiento para el *Individual Model* (entre otras cosas), y a pesar que el modelo simulado no es exactamente el propuesto sino un caso particular de este no se encuentran mejores resultados.

También es muy buena la performance de CRSEQ. Sin embargo en este caso se observan variaciones más abruptas dependiendo de cuántos canales están ocupados. En el caso de MCA se tiene un comportamiento más constante, donde no se presentan los "picos" que se ven en CRSEQ. Esto puede impactar en una red dinámica dónde se manifiestan muchos cambios a nivel de cantidad canales ocupados. En este caso la experiencia de los usuarios es la de una fuerte variación en el tiempo que lleva cada vez que se debe restablecer la comunicación.



(a) número de canales no disponibles Vs MTTR



(b) zoom gráfica (a)

Figura 3.37: MTTR para distintos canales no disponibles

El algoritmo JS queda sorprendentemente mal posicionado. Previamente se presentaba con uno de los que tenía mejor rendimiento. Se deduce que la simulación del modelo utilizado para realizar las simulaciones afectó notoriamente el desempeño. En la gráfica (a) de la figura 3.36 se nota que la diferencia sustancial sucede cuando la cantidad de canales no disponibles aumenta, traduciéndose esto el rápido creciemiento de la curva. Sin embargo para para valores bajos de número de canales no disponible la diferencia (al igual que sucede con los otros algorítmos) en TTR no es tán grande, inclusive siendo menor que CRSEQ (por ejemplo). Esto último abre el estudio de probabilidad de ocupación de canales en un ambiente real con lo cual se podría minimizar subjetivamente el impacto del rendimiento, pero esto se encuentra fuera del alcance del proyecto.

El otro parámetro importante para evaluar completamente el desempeño de los algoritmos es el MTTR (definido anteriormente)

En las figuras 3.36 y 3.37 se aprecia que el posicionamiento de los algoritmos a nivel de MTTR se repite con respecto a lo analizado para TTR. CRSEQ y MCA tienen un MTTR considerable menor que los otros dos algoritmos en todos los casos, salvo cuando no se quitan canales ya que en este caso se está bajo el modelo simétrico.

Los valores extremadamente elevados de JS y MMCA se pueden catalogar como la no convergencia del algoritmo, sobretodo si se piensa en una aplicación donde los valores de MTTR a nivel de un usuario serían inaceptables. Estos casos requerirían, por ejemplo, la modificación de los algoritmos mediante la inclusión de un umbral de *time out*, de modo de reiniciar la búsqueda en caso de sobrepasar dicho umbral.

A partir de este estudio comparativo se concluye que CRSEQ y MCA presentan un mejor desempeño, y se espera que esto se refleje en la implementación de las radios cognitivas en *GNU radio* y *USRP*.

Capítulo 4

Desarrollo de la Radio Cognitiva en USRP

4.1. Introducción

Antes de comenzar con el diseño y desarrollo de las radios cognitivas es necesario tomar decisiones importantes en cuanto al manejo del espectro, comenzando con la lista de canales que se ha de utilizar, así como también la técnica de **transmisión** y **recepción** en dichos canales de forma de obtener una comunicación duplex.

Se decide utilizar los canales de televisión del 20 al 39 y técnicas de *FDD* (*Frequency Division Duplex*), ésto último consiste en separar el ancho de banda del canal en dos bandas, una para recepción y otra para transmisión. Tomando en cuenta esto y el hecho de que los canales de televisión tienen un ancho de banda de 6 MHz se dividen los mismos en dos, teniendo así 3 MHz en cada banda, incluyendo la guarda necesaria. De esta manera, se elige la codificación de tal forma de utilizar un ancho de banda de 1 MHz para la transmisión y entonces se tiene 2 MHz como intervalo de guarda.

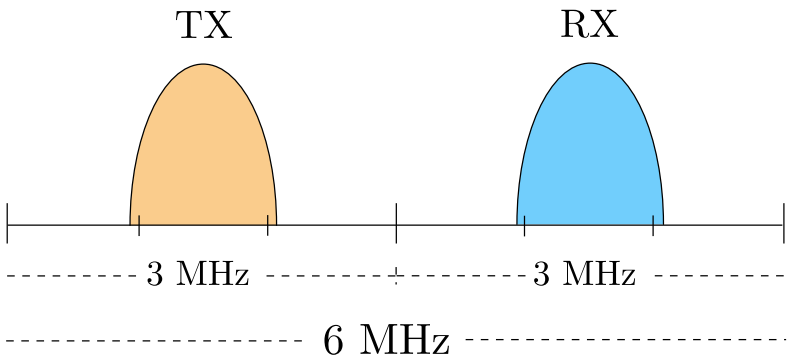


Figura 4.1: FDD

En la tabla 4.1 se detalla la lista de canales que se utiliza en la implementación. En la misma se especifica el número de canal para la URSEC (Unidad Reguladora de Servicios de Comunicación [5]), el número con el cual se llamará de ahora en más a los mismos, la frecuencia de inicio del canal y las dos frecuencias centrales que se utilizarán para el *FDD*.

Canal (URSEC)	Canal	f de inicio (MHz)	f_c1	f_c2
20	1	506	$507,5$	$510,5$
21	2	512	$513,5$	$516,5$
22	3	518	$519,5$	$522,5$
23	4	524	$525,5$	$528,5$
24	5	530	$531,5$	$534,5$
25	6	536	$537,5$	$540,5$
26	7	542	$543,5$	$546,5$
27	8	548	$549,5$	$552,5$
28	9	554	$555,5$	$558,5$
29	10	560	$561,5$	$564,5$
30	11	566	$567,5$	$570,5$
31	12	572	$573,5$	$576,5$
32	13	578	$579,5$	$582,5$
33	14	584	$585,5$	$588,5$
34	15	590	$591,5$	$594,5$
35	16	596	$597,5$	$600,5$
36	17	602	$603,5$	$606,5$
37	18	608	$609,5$	$612,5$
38	19	614	$615,5$	$618,5$
39	20	620	$621,5$	$624,5$

Tabla 4.1: Lista de canales

Ambas radios cognitivas basan su funcionamiento en ***SDR*** por lo cual se desarrolla el sotware para realizar la tareas y el procesamiento necesario en la comunicación. La programación de dicho software se puede separar en dos partes, por un lado el desarrollo de los **flujos de comunicación de datos** y por otro el desarrollo de la **plataforma de control de la radio cognitiva**. En ambos conceptos se profundiza en las secciones de este capítulo dejando claro a qué se refiere con cada uno. De forma básica, en la programación de los flujos de comunicación se parte del código obtenido a partir del diseño de *flowgraphs* en GRC; mientras que el desarrollo de la plataforma de control se realiza a partir de la programación de módulos y funciones específicas que buscan cumplir distintos requeriminetos funcionales de la radio cognitiva.

4.2. Diseño

Principalmente se requiere diseñar la lógica de funcionamiento de la radio cognitiva. Para esto se debe contar con un esquema que describa cómo la radio releva información del medio (capacidad cognitiva) y qué decisiones toma en función de esos datos (auto reconfiguración).

Se presentan dos resultados, un **diseño inicial** y un **diseño final**. En el primer caso se trata de una propuesta teórica, en el sentido que no se utiliza para la implementación de la radio. Los motivos de no utilizar este diseño, y por lo tanto terminar utilizando el diseño final, se basan en impedimentos prácticos. Estas dificultades se describen en la sección correspondiente. Sin embargo, el registro de esta propuesta se basa en el valor de un diseño más completo, que de resolver ciertos problemas abiertos y fuera del alcance del proyecto, se obtendrían mejores resultados en el desempeño de la radio. Así, el diseño final es una simplificación del primero. Es fundamental comprender ambos para interiorizarse con el resultado de este proyecto y querer profundizar en posibles mejoras.

4.2.1. Diseño inicial de plataforma de control

Como primera aproximación al desarrollo de *software* necesario para el funcionamiento de la radio cognitiva se decide separar las tareas en cuatro bloques funcionales. Estas tareas son fundamentales en la comunicación e implementación de movilidad en el espectro, por esto constituyen la base para realizar un diagrama de flujo que muestra el funcionamiento de la radio cognitiva. Los bloques son:

- **Comunicación principal** – Se encarga de la transmisión y recepción de datos en la comunicación principal.
- **Sensado** – Se encarga de sensar un canal de comunicación, es decir, decidir si un canal se encuentra libre o no.
- **Reconocimiento** – Se encarga de reconocer un usuario secundario particular.
- **Algoritmo** – Se encarga de aplicar algún algoritmo de *Rendezvous*.

El control de las funciones de cada uno de estos bloques es manejado por un **programa principal** o plataforma de control. De esta manera la interacción entre los bloques es gestionada tomando decisiones de acuerdo al diagrama de flujo que se muestra en la figura 4.2.

Para realizar el diagrama de flujo básico se deben de tomar algunas hipótesis en el funcionamiento. La más importante es que se considera muy poco probable el caso en que dos usuarios lleguen a un mismo canal en el mismo momento. Además no se toman en cuenta posibles retardos en la comunicación real y se decide que ambos usuarios tengan distintos roles (implicando distintas tareas) según el orden con que arriben al nuevo canal. Con estas consideraciones se procede, primeramente, a realizar diagramas de casos de uso, estos son, diagramas que

muestran cómo procedería el programa principal en distintas situaciones (se omite la presentación de estos diagramas ya que se muestran los definitivos en la sección 4.2.2).

El diagrama de la figura 4.2 se basa en que la transmisión proseguirá ininterrumpidamente mientras no llegue un *PU* al canal donde se lleva a cabo la comunicación y ambos *SU* se mantengan en el mismo. Si esto no sucediera se procede a pausar la comunicación y a realizar el cambio de canal de acuerdo a lo que indique el algoritmo de *Rendezvous*. Luego, cuando un usuario llega a un nuevo canal y en el mismo no hay un *PU*, se distinguen los usuarios. Si un usuario llega al canal y lo sensa vacío este usuario pasa a **transmitir una señal particular** durante un tiempo (su rol es de usuario B) mientras que si llega y sensa un *SU* (su rol es de usuario A) se dedica a intentar reconocer al SU presente en el canal. En este punto se pueden dar dos situaciones, si lo reconoce debe **notificar el encuentro** al usuario B y proceder a restablecer la comunicación principal, en caso contrario se vuelve a cambiar de canal para repetir el procedimiento. Por otro lado el usuario B luego de transmitir la señal particular debe esperar la confirmación de encuentro por parte del usuario A para restablecer la comunicación, si esto no sucediera vuelve a cambiar de canal.

En la implementación de este diseño se detectan grandes problemas asociados a las hipótesis tomadas. El principal problema surge en las primeras pruebas de desempeño, cuando las radios arriban a un nuevo canal y se debe de decidir quién es el usuario A y el usuario B a partir del orden de llegada, en este punto se concluye que la hipótesis de que era improbable que ambos usuarios llegaran al mismo tiempo era errónea. No solo es probable que esto suceda, si no que sucede con mucha frecuencia. Esto desemboca en que ambos usuarios quedan como usuario A o usuario B, generando una situación que no está contemplada en flujo. En consecuencia se hace imposible el reconocimiento y por lo tanto el reestablecimiento de la comunicación no se logra.

El inconveniente se atribuye a que los tiempos que necesitan las *USRP* y la plataforma de control para realizar algún cambio en la transmisión o recepción es apreciable, y por lo tanto la hipótesis de que ambos usuarios no arriban en un mismo instante no se cumple porque no es un instante, sino que es un periodo de tiempo.

Dado este problema, se realiza un cambio radical en el diseño de la plataforma de control manteniendo solamente algunos fragmentos básicos del diagrama de la figura 4.2. Este nuevo diseño se explica en la sección 4.2.2 y para llegar a él ser realizaron varias simplificaciones. Como se explicó se mantiene la descripción del diseño inicial en este documento considerando el trabajo a futuro, ya que si se profundizara sobre los métodos de reconocimiento de *SU* y de sensado, este diseño sería el más completo para comenzar el estudio.

Adicionalmente, los inconvenientes detectados dan lugar a la necesidad de realizar un estudio detallado de los tiempos que se requieren para el buen funcionamiento de la radio cognitiva, ya que estos no son tan bajos como se esperaba.

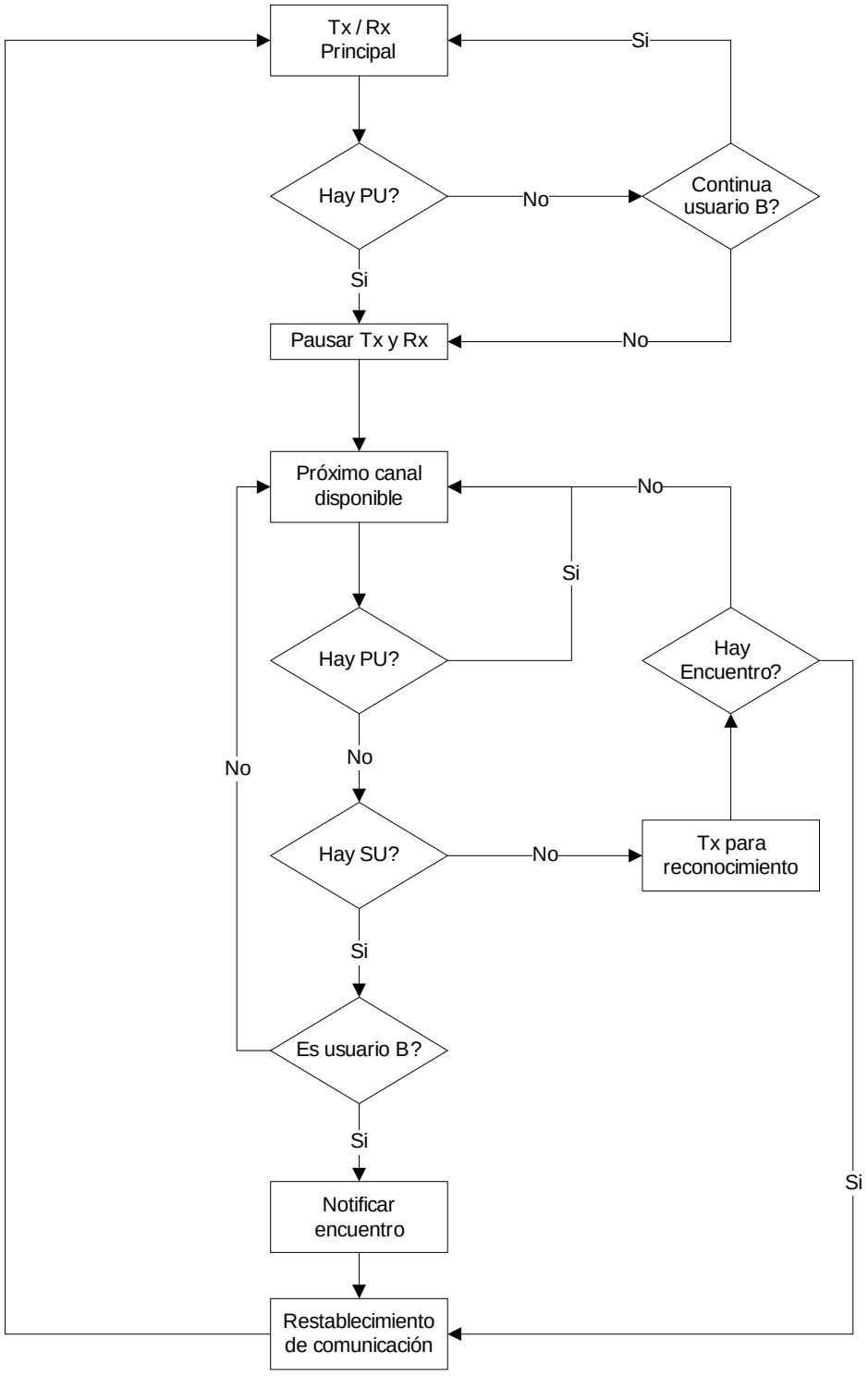


Figura 4.2: Diagrama de flujo inicial

4.2.2. Diseño final de plataforma de control

A partir de los inconvenientes encontrados en el funcionamiento del desarrollo presentado en la sección 4.2.1 se ve una clara necesidad de realizar cambios drásticos. Sin embargo, se utiliza el mismo camino metodológico que en el diseño inicial. El primer cambio implica reducir la cantidad de bloques funcionales a tres. Los mismos son:

- **Comunicación principal** – Se encarga de la transmisión y recepción de datos en la comunicación principal. Dentro de la recepción se implementa el sensado de usuario primario y secundario.
- **Reconocimiento** – Se encarga de reconocer un *SU* particular. También sensa la presencia de *PU* en el nuevo canal. Dentro de este bloque se implementa la transmisión y recepción particular que se utilizan para realizar el reconocimiento del *SU*.
- **Algoritmo** – Se encarga de aplicar algún algoritmo de *Rendezvous*. Inicializa los parámetros del algoritmo a utilizar y calcula las secuencias de canales.

Como se puede apreciar, el bloque funcional que se quita con respecto al diseño inicial es el de **Sensado**. Las tareas de dicho bloque se reparten entre el bloque **Comunicación Principal** y **Reconocimiento** ya que dependiendo de la etapa en que se encuentre la radio alguno de los anteriores realiza esta función.

El control de las funciones de cada uno de estos bloques es manejado por un programa principal. De esta manera la interacción entres los bloques es gestionada tomando decisiones de acuerdo al siguiente diagrama de flujo:

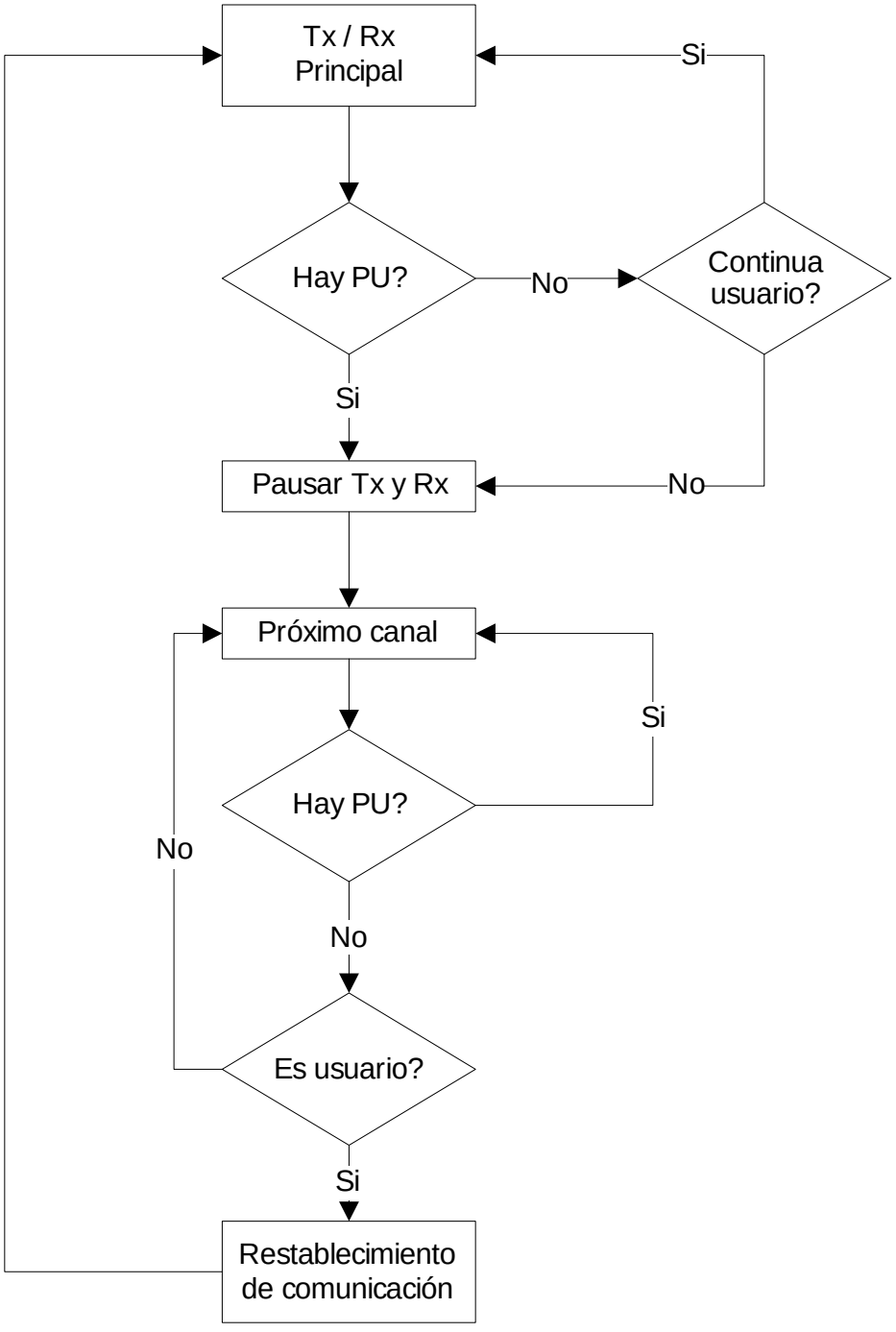


Figura 4.3: Diagrama de flujo final a implementar

El diagrama de la figura 4.3 se basa en que la **transmisión proseguirá ininterrumpidamente** mientras no llegue un PU al canal donde se lleva a cabo la comunicación y ambos SU se mantengan en el mismo. Si esto no sucediera se **pausa** la comunicación y se realiza el **cambio de canal**. Luego, cuando un usuario llega a un nuevo canal se procede a **sensar** el mismo para detectar la presencia o

ausencia de algún PU. Hasta aquí el funcionamiento es similar al diseñado inicialmente, sin embargo el gran cambio radica en lo siguiente. Luego de no detectar PU se comienza a transmitir una señal particular y paralelamente a intentar detectar una señal particular de otro SU. Si este reconocimiento es satisfactorio se prosigue con la comunicación principal. En caso contrario, si luego de pasado un tiempo de espera no se reconoce al otro SU, se precederá a cambiar de canal nuevamente.

Seguidamente se expondrán las interacciones entre los bloques funcionales y el programa principal en algunos de los casos de uso estudiados para una mejor comprensión del diagrama de flujo. Se toma como caso de uso cada una de las distintas situaciones que se pueden dar en la comunicación.

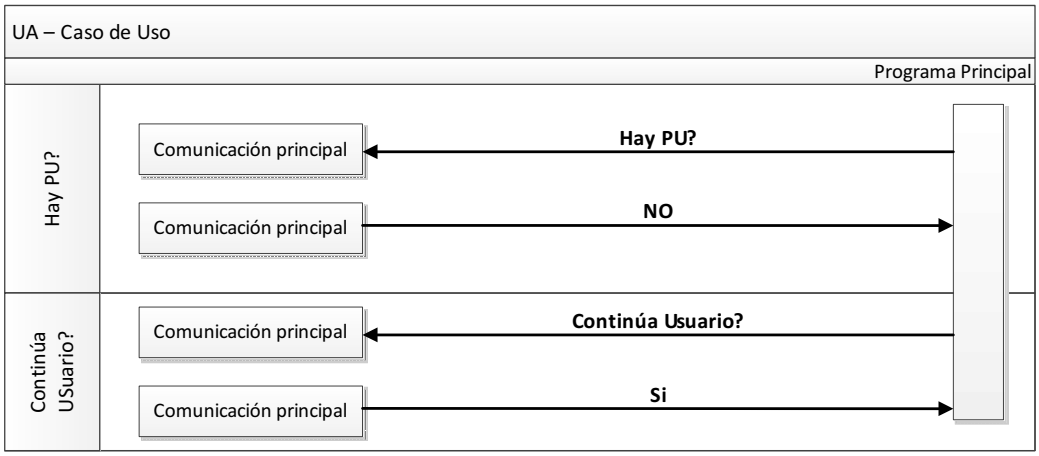


Figura 4.4: Comunicación principal ininterrumpida

En el ejemplo de la figura 4.4 se estudia el caso de uso en que la comunicación no se interrumpe porque no aparece un PU y el SU con quien se lleva a cabo la comunicación se mantiene en la misma.

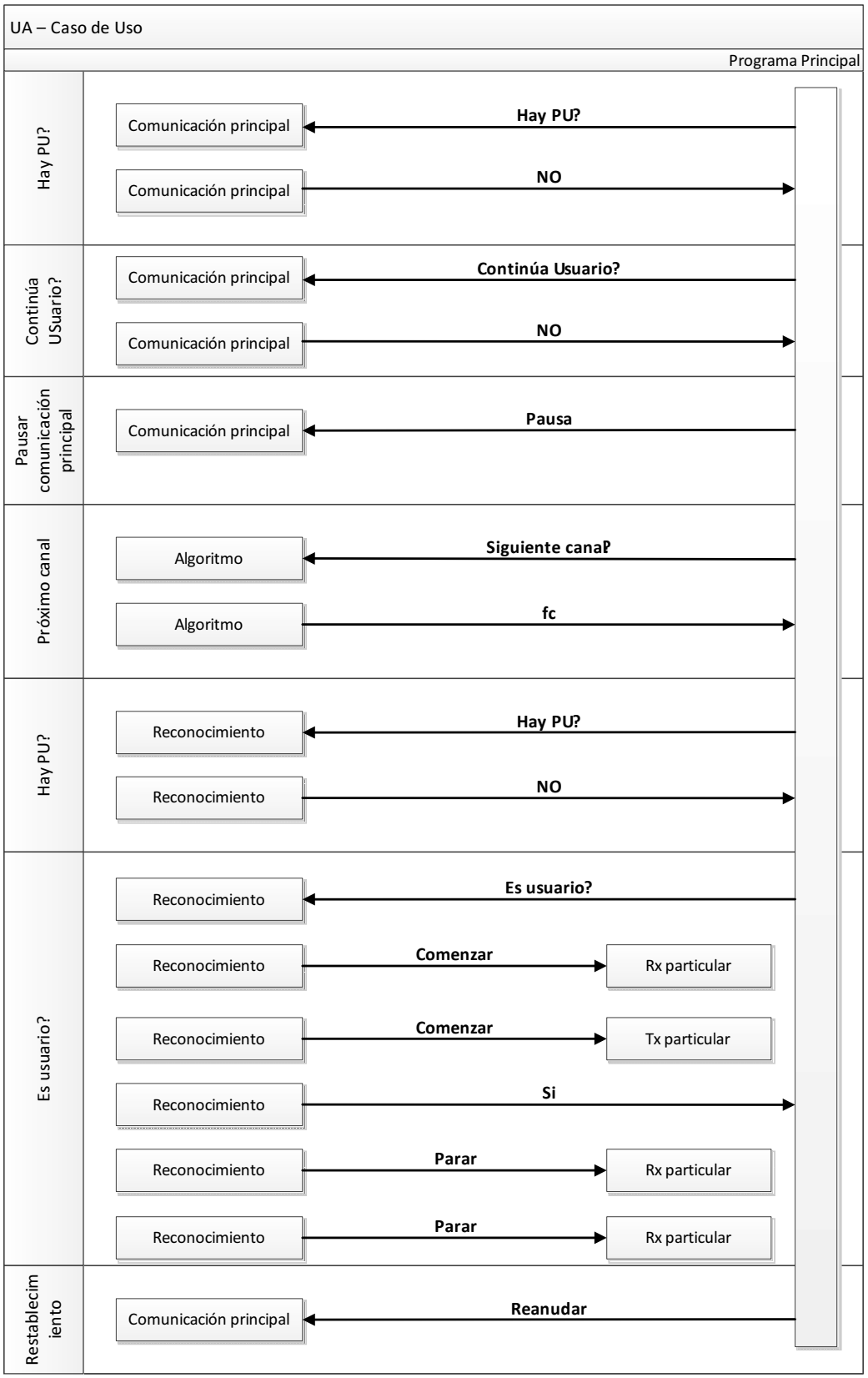


Figura 4.5: Usuario secundario abandona comunicación

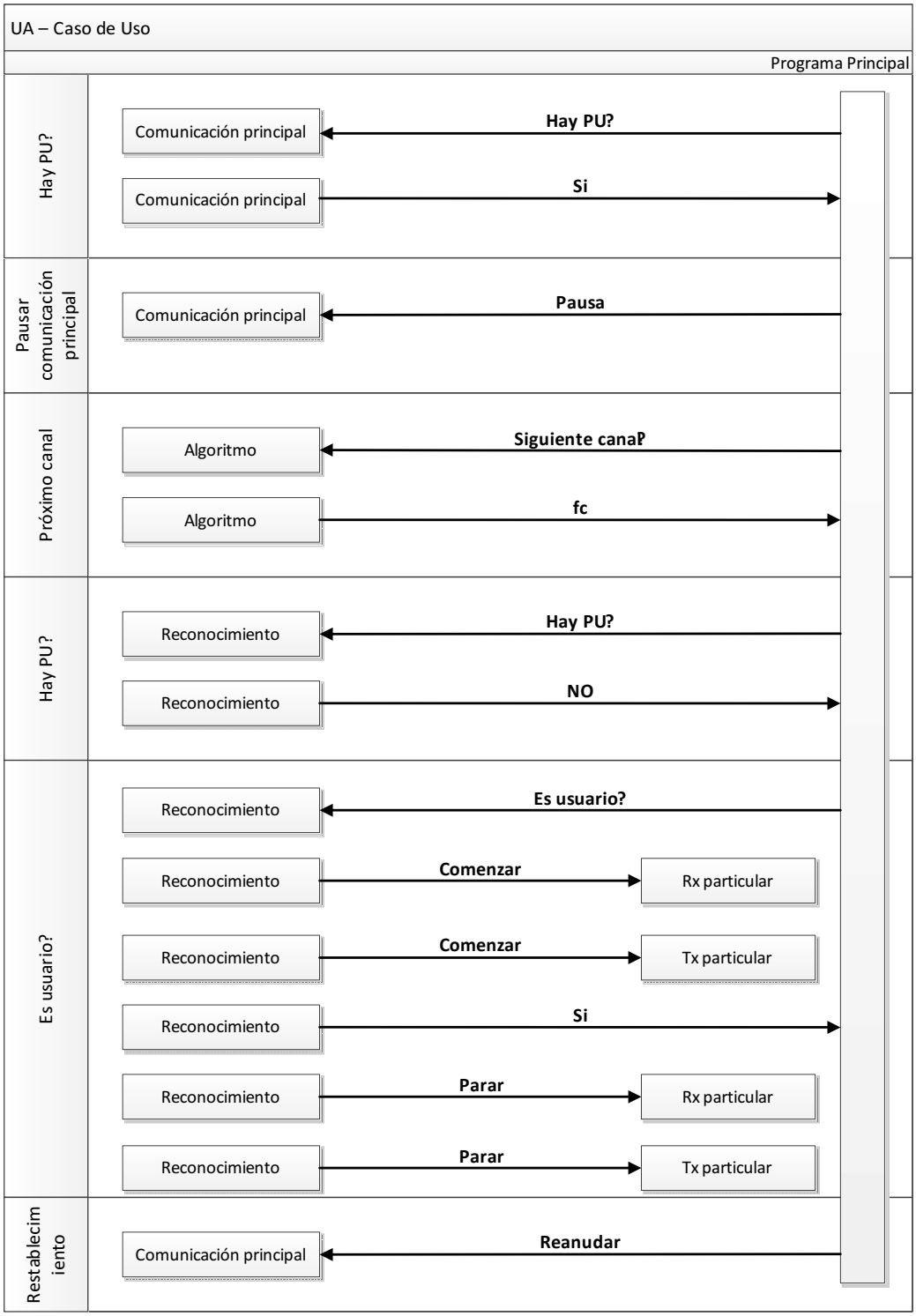


Figura 4.6: Aparición de usuario primario

En el ejemplo de la figura 4.5 se puede apreciar el caso en el que por alguna razón uno de los usuarios abandona el canal, entonces el restante debe abando-

narlo también. A continuación prueba en un posible canal (obtenido mediante el algoritmo correspondiente) que casualmente es en el que se encuentra el usuario con el que se mantiene la comunicación, si esto no sucediera y se encontrara un *PU* o el *SU* no estuviera en el mismo, simplemente se sigue cambiando de canal hasta encontrarse.

En el ejemplo de la figura 4.6 tenemos el caso en que la comunicación es interrumpida por la aparición de un *PU* en el canal donde se lleva a cabo la comunicación principal. Para ilustrar mejor esta idea se considera el caso en que luego del primer salto se encuentra inmediatamente con el *SU* con el que se mantenía la comunicación.

4.3. Implementación

El diseño estudiado en la sección 4.2.2 es la base para la implementación de la radio cognitiva. En cada una de las siguientes secciones se describen las partes más representativas en el desarrollo de la radio y el resultado obtenido.

4.3.1. Sensado

Se le llama sensado a la función de **medir potencia** en un canal del espectro. Luego, a partir de esta medición, se toman decisiones en el control de la radio cognitiva. Para esto se debe tener algún valor de referencia para comparar con el valor de potencia medido, éste valor de comparación se le llama **umbral**.

Para simplificar y focalizarse en obtener un sensado funcional se implementa de esta manera solamente el **sensado de *SU*** y la aparición de un *PU* se simula aleatoriamente.

Tomando como base lo anterior el sensado se implementa en dos diferentes etapas. Por un lado en la comunicación principal, para detectar la presencia del otro usuario con el que se lleva a cabo la comunicación, y por ende detectar cuando el mismo abandona el canal. Por otro lado se implementa al final del reconocimiento cuando se llega a un nuevo canal para confirmar que el otro usuario haya logrado el reconocimiento también. Los detalles de este proceso se profundizan en la sección 4.3.3.

Figura 4.7: Bloque de GRC utilizado para el sensado

Para la implementación del sensado se utiliza el bloque de *GRC* llamado ***Probe Avg Mag ^ 2*** (figura 4.7), el cual toma muestras de la señal que se está transmitiendo en el canal y mide potencia en el mismo. Los parámetros del bloque, *Threshold* y *Alpha*, se mantienen con sus valores por defecto. El primero fija un

umbral en dB tal que si la señal en la entrada se encuentra por debajo la medida es 0. El segundo representa el valor del polo del filtro IIR implementado en el bloque, y el cual utiliza antes de comparar la señal con *Threshold*.

Para obtener un valor concreto se implementan una serie de mediciones y se realiza el promedio de las mismas. Con esto se evita un comportamiento variable en la salida del bloque, como se explica en 5.2. Además se aumenta la "ventana" de medición, lo cual implica que para detectar otro usuario no bastará con que éste pase por el canal, si no que deberá permanecer en el mismo todo el tiempo que dure la "ventana". De esta manera se evita, por ejemplo, sensar un usuario que en realidad está abandonando el canal. Concretamente, para realizar este promedio se toman diez muestras, esperando 1 ms entre cada una.

4.3.2. Flujos de comunicación principal

El objetivo principal es obtener una plataforma de transmisión y recepción para realizar la prueba de los algoritmos. Esta implementación no busca un estudio en si misma por lo que se utilizaron la mayor cantidad de herramientas predefinidas.

La comunicación principal se basa en un flujo de *GNU radio* generado basicamente en *GRC*. La misma consiste en una transmisión de datos utilizando una modulación digital ($GMSK$) con frecuencia de portadora en la banda de $509Mhz$ a $629Mhz$.

Transmision (TX) principal

El flujo de datos desde su origen hasta el frente de RF se esquematiza en la figura 4.8

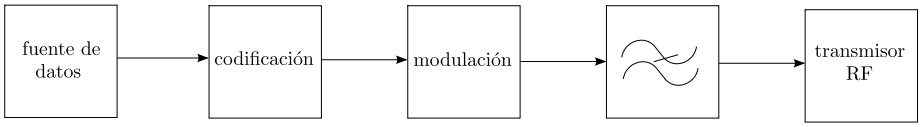


Figura 4.8: Diagrama de bloques de TX principal

Para implementar esta transmisión se utilizan los bloques de procesamiento provistos por *GNU radio*. En la captura 4.9 se aprecia especificamente los bloques utilizados y las conexiones entre los mismos.

Por razones prácticas se trabaja con un archivo de texto como fuente de datos.

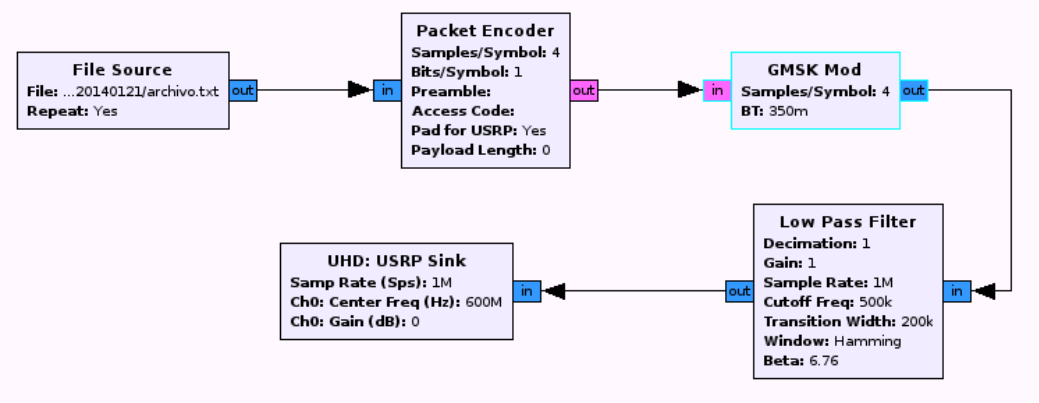


Figura 4.9: Diagrama de bloques de TX principal

La clase de *python* generada con este *flowgraph* es la que se utiliza en la implementación final a menos de alguna modificación realizada en el código cambiando ciertos parámetros.

Recepción (RX) principal

El flujo de datos desde la recepción hasta la recolección de los mismos se esquematiza en la figura 4.10

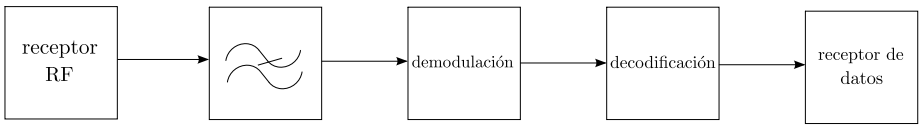


Figura 4.10: Diagrama de bloques de RX principal

De igual manera que la transmisión principal, en la recepción se utilizan bloques de procesamiento provistos en *GNU radio*. Los bloques y sus correspondientes conexiones se muestran en la captura 4.11.

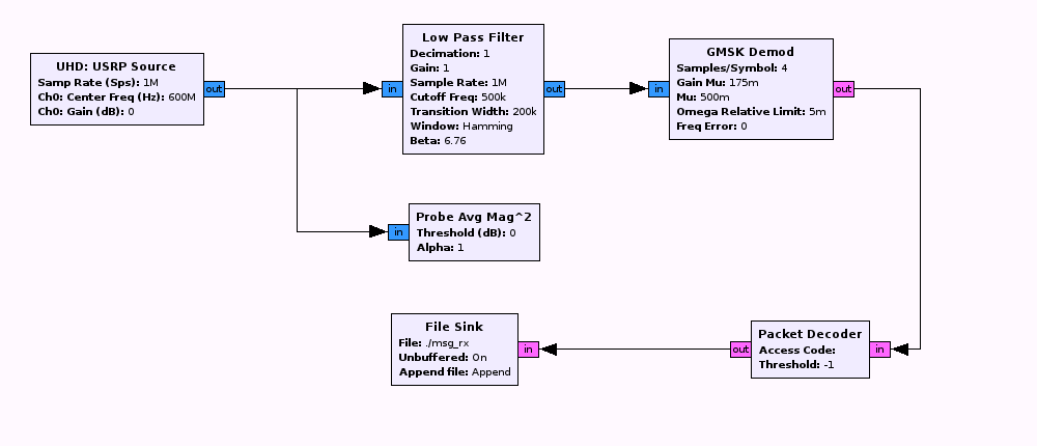


Figura 4.11: Diagrama de bloques de RX principal

En el *flowgraph* se aprecia que la señal recibida y filtrada pasa a través de un bloque *probe Avg Magˆ 2*. Como se explica en la sección 4.3.1 este bloque facilita la detección de *SU* a partir de la comparación con un nivel de potencia.

Observaciones

- Como se mencionó, los datos en la transmisión principal provienen de un archivo de texto. La utilización de este archivo es arbitraria. Los bloques encargados de realizar el manejo de los archivos son el *file source* y *file sink* esto se puede apreciar en las figuras 4.9 y 4.11.
- La modulación escogida para la trasnsmisión es ***GMSK*** (*Gaussian minimum shift keying*) la cual deriva de *MSK*. Estas modulaciones implementan un esquema de fase continua. Como el nombre lo indica *GMSK* difiere de *MSK* en que los pulsos binarios en banda base son suavizados mediante un filtro Gaussiano.
- la frecuencia de portadora es seteada en los bloques ***UHD: USRP sink*** y ***UHD: USRP source***, por lo cual queda bajo su manejo la conversión de la señal de banda base a pasabanda y viceversa.

4.3.3. Reconocimiento de Usuario Secundario

Luego de que un usuario llegue a un nuevo canal y verifique que en el mismo no se encuentre un *PU*, es necesario confirmar la presencia del usuario con el que se mantenía la comunicación principal.

Es importante destacar que este concepto no se encuentra presente en las publicaciones consultadas. Sin embargo el reconocimiento es clave para el correcto funcionamiento de la radio cognitiva en si. Ya que, si bien en el capítulo 3 se presentaron diferentes algoritmos de encuentro, los cuales proponen distintas técnicas para que se cumpla el *Rendezvous*, el hecho de verificar que en un canal puntual

está presente el otro usuario es fundamental para **garantizar el encuentro**. Los detalles que se presentan sobre esta tarea se deben, precisamente, a la centralidad de la misma y la falta de fuentes precedentes sobre su estudio.

Frente a diferentes formas para realizar esta tarea, en este proyecto se opta por una implementación que se basa en enviar y recibir un **mensaje particular** durante un tiempo determinado. Para esto se crea una función (*reconocimiento_SU*), cuyo pseudo-código se puede apreciar en las líneas siguientes:

```
1 Inicializar parámetros()
2 while (No se de el encuentro) do
3 |   Enviar mensaje()
4 |   Recibir mensaje()
5 |   if ((Se recibe mensaje particular) or (pasa un tiempo determinado))
  |   then
6 |   |   Abandonar el While
7 return (Se produce o no el encuentro)
```

Dejando de lado las complejidades de la implementación, la idea detrás se puede resumir en unos pocos pasos. Esto es, durante un tiempo determinado enviar un mensaje particular y si ambos reciben dicho mensaje se puede decir que se da el encuentro.

Antes de profundizar en el tema se definen dos tiempos que serán de gran utilidad para comprender todo el proceso de reconocimiento implementado.

- **Tiempo de espera**: De no recibirse el string característico por parte del otro usuario, el tiempo de espera es el tiempo máximo que cada usuario se mantiene en un canal durante el reconocimiento.
- **Tiempo de confirmación**: Es el tiempo que cada usuario se mantiene en el canal (durante el reconocimiento), luego de recibir el mensaje particular enviado por el otro usuario secundario.

Abordando más específicamente la implementación se identifican una serie de tareas a cumplir:

1. Transmitir un mensaje particular.
2. Recepción y procesamiento de los datos recibidos.
3. Elección del tiempo de espera.
4. Elección del tiempo de confirmacón.

1. En este punto se separó el envío del mensaje particular de la transmisión principal, esto tiene como finalidad generar independencia entre la transmisión principal y la transmisión particular. Si bien en la implementación se

utiliza la misma modulación que en la transmisión principal (*GMSK*), la separación se justifica en dejar la libertad para que en modificaciones futuras se pueda optar por otro esquema para la transmisión.

En la figura 4.12 se puede apreciar el flujo de datos de la transmisión particular. A grandes rasgos cuenta con un modulador de *GMSK* y un bloque *Message Source*.

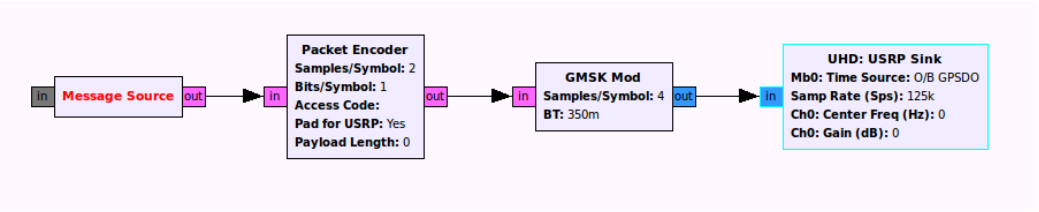


Figura 4.12: Flujo de datos de transmision particular

La utilización del bloque *Message Source* tiene la siguiente justificación. En *GNU radio* se tienen principalmente dos formas de procesar y transmitir un mensaje particular. Se puede utilizar un manejo mediante vectores de datos o mensajes. En caso de que se quiera realizar un procesamiento de los datos en tiempo real se recomienda casi exclusivamente el uso de mensajes. Por otro lado, junto con los mensajes aparece el objeto *message queue* (colas de mensajes) con una variedad de funciones que simplifican el procesamiento y de las cuales se hace uso en esta implementación. Este objeto es una instancia de la clase *gr.msg_queue*.

Se elige como mensaje particular a transmitir un *string* ("87654321"), el mismo se envía de manera repetida durante un tiempo determinado. Para cumplir con dicho objetivo se utiliza la función del objeto creado a partir de la clase *gr.msg_queue*, esta es: *insert_tail(gr.message_from_string(*87654321*))*. Este mensaje es el que va a ser reconocido en la recepción de datos por el otro usuario.

2. Como se mencionó en el punto anterior, para la recepción y procesamiento del mensaje particular se utiliza el objeto *gr.msg_queue()*. El flujo de datos de la recepción particular se puede apreciar en la figura 4.13.

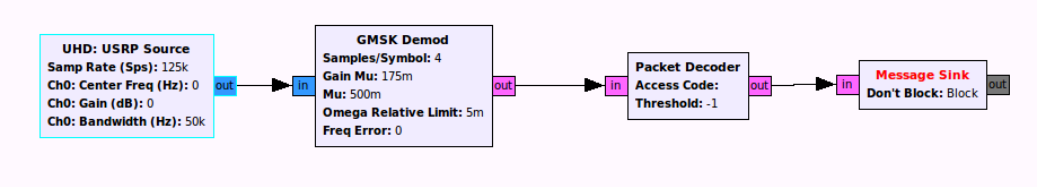


Figura 4.13: Flujo de datos de recepción particular

El mensaje es recibido en el bloque *Message Sink* y almacenado en el objeto "cola" (*cola = gr.msg_queue()*). En el caso en que se esté efectivamente

recibiendo el mensaje 87654321, es de esperar que en el objeto "cola" se encuentre el mismo repetido varias veces. Como criterio de diseño se decide por aceptar que se recibe el mensaje cuando se encuentra el mismo más de 3 veces en "cola". Para esto se utiliza la función de "cola" *delete_head().to_string()* (que convierte el mensaje en la cola a un *string*), y luego la función *count(*87654321*)* de *string*.

En resumen el proceso se puede describir como:

- Suponiendo que se recibe el mensaje particular, el mismo se guarda en "cola" (objeto de la clase *gr.msg_queue*).
- Dicho objeto tiene una función que transforma los mensajes a *string*. Con lo cual se crea *msg = cola.delete_head().to_string()*.
- Se utiliza una función auxiliar de *string*, la cual sirve para contar la cantidad de veces que aparece el mensaje, y si el mismo aparece más de 3 veces se acepta que se recibe correctamente el mensaje particular.

Esto es:

```
1 msg = cola.delete_head().to_string()
2 if (msg.count("87654321") > 3) then
3 |  Se recibe correctamente el mensaje particular
```

En cuanto a los puntos 3 y 4, los mismos serán detallados en la sección 5.3 ya que el estudio de tiempos necesita mayor profundidad con lo que se le dedica una sección entera.

4.3.4. Programa principal

El programa principal consta de un *script* en código *python* el cual se encarga de generar objetos de clases particulares y así, utilizar las funciones y variables de estos objetos para manejar el proceso en la radio cognitiva.

El mismo implementa el flujo descrito en la sección 4.2.2 gestionando diferentes funciones provistas por otros módulos. De esta manera funciona como plataforma de control y se encarga, a modo global, del correcto funcionamiento de la radio cognitiva en cada usuario.

Una función del programa principal (*main.py*) es asignarle una identidad a cada usuario. Recordando lo mencionado en la sección 4.1, para la transmisión y recepción se utilizan técnicas de *FDD*, separando así el espectro asignado a cada canal en dos partes, una para la transmisión y otra para la recepción. Teniendo esto en cuenta se elige como técnica de diseño asignarle un identificador a cada usuario, los cuales son usuario **A** y **B**. Con esto se elige que el usuario **A** reciba en la mitad "baja" del canal y transmita en la mitad "alta" del canal, y el usuario **B** transmita en la mitad "baja" y reciba en la mitad "alta". Vale la pena aclarar que si bien la elección de **A** y **B** se impone al inicio del programa principal de

forma arbitraria, se podría implementar la decisión por medio de alguna técnica de *handshake*.

Otro punto a destacar es que se opta por un diseño del programa en una estructura modular, teniendo así independencia entre la diferentes tareas, pudiendo modificar cada uno de los bloques mencionados en la sección 4.1 (**comunicación principal**, **reconocimiento** y **algoritmo**), ya sea por medio de mejoras como cambiando drasticamente el enfoque. De esta forma el programa principal seguirá funcionando correctamente, siempre y cuando se respeten las interfaces y llamadas que se realizan desde *main.py* a los distintos módulos.

4.3.5. Algoritmos

Para implementar los algoritmos se crea un módulo en *Python* para que maneje los mismos. Además se encarga del mapeo entre frecuencia y número de canal, ya que los algoritmos devuelven el número del canal al que hay que saltar. Este módulo se compone básicamente por una clase y cuenta con la función principal de ejecutar el algoritmo que se elija utilizar. Las opciones son los cuatro algoritmos implementados.

- **CRSEQ**
- **Jump-Stay**
- **MCA**
- **MMCA**

Cada uno de los algoritmos se implementa como clases individuales, con la función básica de generar la secuencia de *Channel hopping*. Luego son importadas desde el módulo **algoritmos.py** y las secuencias pasan a formar un atributo de la clase **algoritmo**, cualquiera sea el que se haya escogido. Este atributo es al cual el programa principal puede acceder (llamando a la función correspondiente) de modo de identificar a qué frecuencias se debe ir moviendo.

4.3.6. Resultado

Finalmente la radio cognitiva corre a partir de un programa en *python* llamado *main.py*, este es el que llamamos programa principal. Dicho programa implementa la plataforma de control de la manera que se explicó en 4.3.4. Adicionalmente se cuenta con una serie de módulos de *python* los cuales son importados desde el programa principal. Cada uno de estos módulos consta básicamente de la definición de una clase con sus correspondientes variables y funciones.
Los módulos que se tienen son:

- TX_ppal.py - *Transmisión principal*
- RX_ppal.py - *Recepción principal*

- algoritmos.py - *Manejo de algoritmos*
- reconocimiento.py - *Manejo del reconocimiento*
- TX_particular.py - *Transmisión particular (reconocimiento)*
- RX_particular.py - *Recepción de señal particular (reconocimiento)*
- JS.py - *Algoritmo JS*
- MCA_MMCA.py - *Algoritmo MCA y MMCA*
- CRSEQ.py - *Algoritmo CRSEQ*

En la sección 4.2.2 se definen los bloques funcionales en los cuales se desarrollan las tareas de la radio cognitiva. El esquema 4.14 muestra cómo se vinculan los módulos de *python* programados con los bloques funcionales, y cómo los diferentes archivos listados anteriormente terminan implementando dichos bloques.

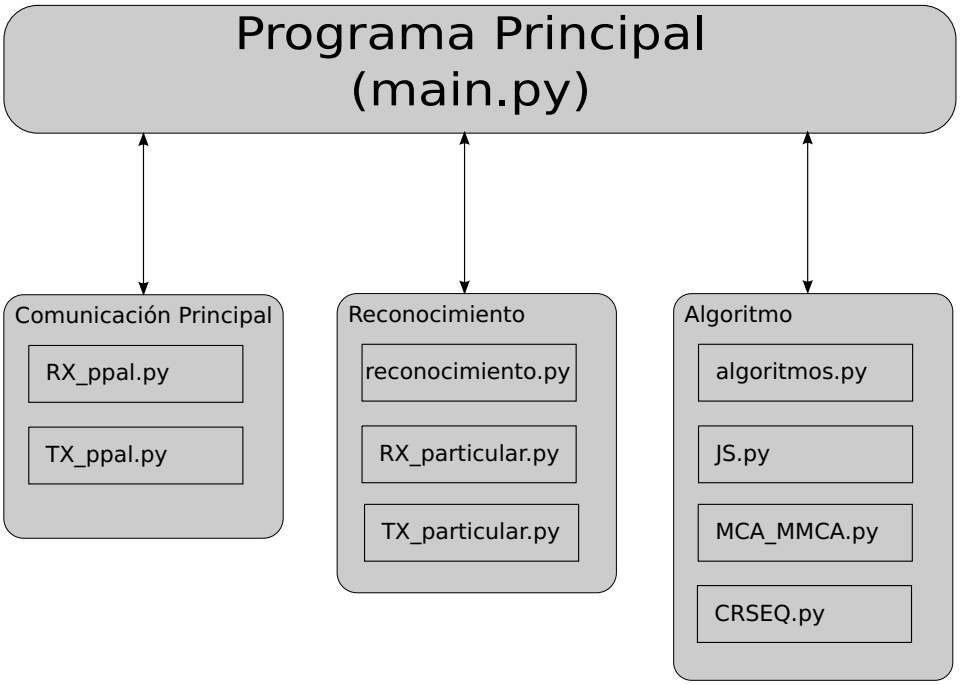


Figura 4.14: modulos de python y bloques funcionales

Las clases TX_principal y RX_principal, definidas en los módulos TX_ppal y RX_ppal respectivamente, presentan independencia con respecto al resto de las clases. La creación de un objeto de cualquiera de éstas, a partir de la definición de

variables como el archivo a enviar y la frecuencia de *TX/RX*, genera el camino para realizar la transmisión y recepción de la comunicación principal. Ambos corren en paralelo haciendo uso la capacidad del *USRP* para transmitir y recibir utilizando una sola *daughterboard*. Las funciones definidas en las clases, como se muestra en el diagrma 4.15, son llamadas por el programa principal, y sus resultados se usan para definir acciones, básicamente decidir si **se continua la transmisión principal o se abandona**.

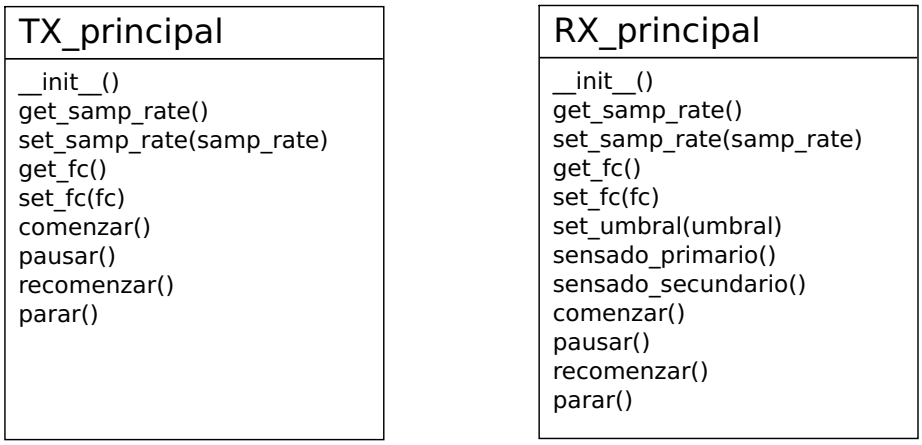


Figura 4.15: diagrama de clases TX_principal y RX_principal

Para las clases involucradas en el **reconocimiento** existe un vínculo que las relaciona. La creación de un objeto a partir de la clase recon (definida en reconocimiento.py) simboliza el marco dentro del cual se lleva a cabo el reconocimiento del *SU*. Las funciones de recon hacen uso de los objetos creados a partir de TX_part y RX_part (figura 4.16). Si bien éstos no son atributos de la clase, la asociación radica en que se pasan como parámetros a las funciones en el reconocimiento. Nuevamente, la creación y el control de estos vínculos se lleva a cabo en el programa principal.

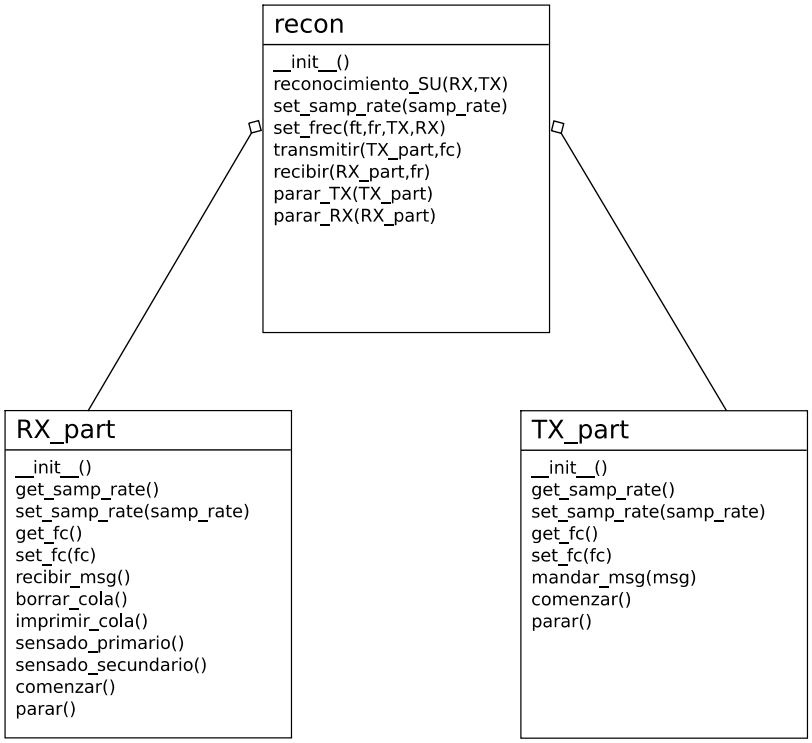


Figura 4.16: diagrama de clases involucradas en el reconocimiento

Finalmente, las clases que se utilizan en el manejo de los algoritmos presentan un vínculo jerárquico más fuerte. Cada uno de los algoritmos particulares termina siendo un atributo de la clase algoritmo, como se explicó en 4.3.5. El diagrama 4.17 esquematiza esto.

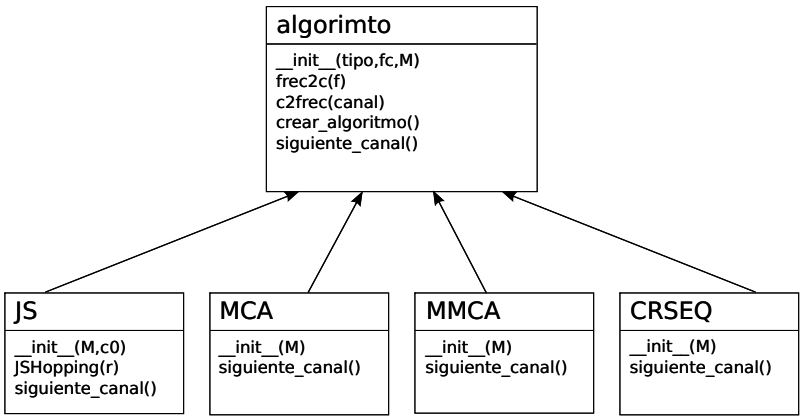


Figura 4.17: Diagrama de clases relacionadas a los algoritmos de Rendezvous

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Desempeño de la Radio Cognitiva

La implementación que se obtuvo a partir del diseño planteado funciona correctamente bajo ciertos límites. A continuación se desarrollan los factores que influyen en el desempeño de la radio cognitiva. A partir de esto se busca una justificación de estos límites y realizar un planteo claro de los temas a mejorar en el futuro.

5.1. Falso encuentro

Llamamos falso encuentro al caso en el que alguna de las radios llega a reconocer al otro *SU* y luego del tiempo de confirmación no se logra concretar el *Rendezvous*. En otras palabras, los *SU* se cruzan en un canal común pero no se logra establecer la comunicación principal nuevamente.

Observando el esquema en la figura 5.1. El usuario B llegó al canal 1 y se encuentra esperando que se realice el encuentro con el usuario A. En un determinado momento, señalado con la primer línea punteada llega al canal el usuario A. Sin embargo B termina su tiempo de espera en el canal 1 y se mueve hacia el canal 2. Como se definió anteriormente, el reconocimiento, y por lo tanto el encuentro, se confirma luego de que se realiza un sensado al finalizar el tiempo de confirmación. Así, el usuario A sensará el canal 1 finalizado T_{conf} (ya que aún permanece en este canal) y no encontrará a B. Sin embargo **ambos usuarios estuvieron presentes en el mismo canal**.

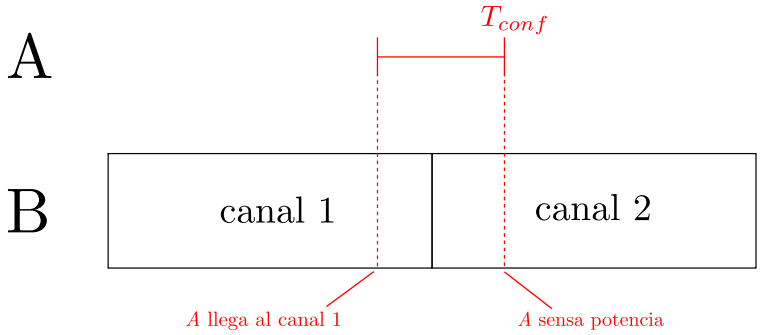


Figura 5.1: esquema de una falso encuentro

Los falsos encuentros se separan en dos tipos, falsos encuentros **detectados** y falsos encuentros **encubiertos**. Esta distinción se basa en el manejo que la radio cognitiva es capaz de realizar cuando se da un falso encuentro. Si la radio cognitiva logra percibir que se está en presencia de un falso encuentro (para esto se realiza un sensado luego del tiempo de confirmación), se dispara un mecanismo por el cual el algoritmo de búsqueda continúa de manera normal. Este caso es similar al que se muestra en la figura 5.1, donde se supone por ahora que esta situación es válida, la explicación y justificación se encuentra en secciones posteriores.

Sin embargo, los falsos encuentros encubiertos, como su nombre lo indica, son aquellos que la radio cognitiva no logra distinguir, y por lo tanto un *SU* asume que se da el *Rendezvous* mientras que el otro no. El caso en que esto sucede, y que también se explica posteriormente, se da en un esquema como en la figura 5.1 cuando $canal\ 1 = canal\ 2$.

5.2. Umbrales y sensado

Para sensar (como se mencionó en la sección 4.3.1), se utiliza el bloque *Probe Avg Mag^2*. El uso de este bloque presenta una serie de inconvenientes:

1. Encontrar un tiempo mínimo entre sensados para que efectivamente se mida potencia.
2. Fijar el umbral para filtrar señales no deseadas.
3. Intepretar el valor numerico que arroja el bloque como la presencia (o ausencia) de un usuario.

Vale la pena aclarar lo importante del punto 1, pues si se sensa constantemente se obtiene como resultado 0. Se observa que dicha función necesita un tiempo entre sensados para funcionar correctamente. Para no entrar en la lógica interna del bloque *Probe Avg Mag^2*, este tiempo fue calculado de manera experimental tras varios ensayos, concluyéndose que un valor de 1 ms es el más apropiado. Otra razón

por la cual no se profundiza en esto es que se entiende que existen otras técnicas para resolver el problema de sensado y reconocimiento mucho más eficientes (y también mucho más complejas), pero las mismas exceden el alcance del proyecto.

Otro inconveniente a tomar en cuenta es que de sensar una sola vez se puede caer en el momento en el que el otro usuario se encuentre transmitiendo con muy baja potencia (o nula) pero de forma transitoria. Por esta razón se realiza un promedio de 10 mediciones. Esta solución implica una mayor duración en el proceso, 10 veces más que realizar una sola medición.

Como se explicó, al bloque en cuestión se le setea un valor de umbral de tal forma que si la señal de entrada es menor al umbral la medición da cero. Para fijar este valor se realizaron varias pruebas optándose por dejarlo en $0dB$ (nivel por el cual no filtraría ninguna señal). En la búsqueda de un valor mínimo para establecer el umbral se encuentran variaciones. Por un lado el valor depende del canal en el que se realice el sensado, por lo que no es independiente de la frecuencia. Por otro lado depende de elementos externos, ya que las medidas tomadas en distintos días difieren. De esta manera se deja de lado esta funcionalidad y se realiza una implementación más simple, donde se toma como nivel de decisión la medida misma de potencia del bloque (esto se puede realizar gracias a la función auxiliar *level()* del bloque).

Al utilizar el valor que devuelve la función *level()* se debe tener en cuenta un problema ya mencionado, la potencia medida depende de la frecuencia. Por lo tanto se decide estudiar este problema realizando mediciones de potencia en los distintos canales, como se puede apreciar en la tabla 5.1 y la figura 5.2. Para realizar dichas mediciones se toman las siguientes consideraciones:

1. Todas las medidas fueron tomadas el mismo día, tomando en cuenta el momento en que no había PU en ninguno de los canales.

2. Para la transmisión se utiliza el bloque de transmisión de la comunicación principal.

3. El *USRP* encargado de la transmisión es la que tiene número de serie EBR15UFB1 con ganancia de 50dB y el encargado de medir potencia es el de número de serie EBR15U1B1 (este punto no es menor pues dependiendo de cual sea el transmisor y cual el receptor, se modifica el valor de la salida de la función *level()*).

f_c (MHz)	Potencia
507,5	0,3707
513,5	0,3828
519,5	0,3036
525,5	0,2247
531,5	0,1623
537,5	0,1346
543,5	0,1464
549,5	0,1583
555,5	0,1961
561,5	0,2765
567,1	0,4256
573,5	0,6061
579,5	0,8488
585,5	0,9390
591,5	0,9741
597,5	0,5827
603,5	0,2361
609,5	0,1053
615,5	0,06675
621,5	0,03336

Tabla 5.1: Medida de potencia para distintos canales

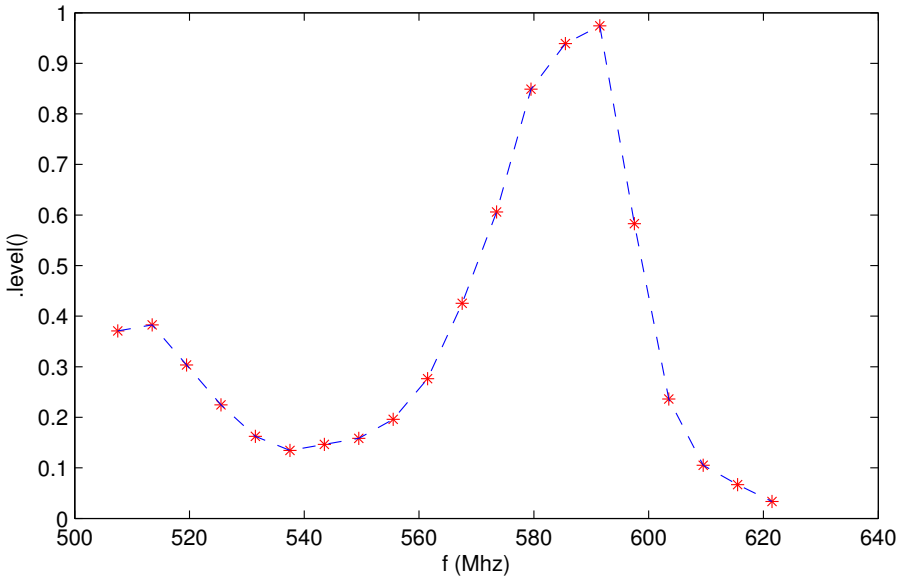


Figura 5.2: Salida de la función level() para distintos canales

Volviendo a la tercer hipótesis. Es importante dejar claro cuál *USRP* funciona como transmisor y cuál como receptor. Ya que el valor sensado varía considerablemente, aproximadamente del orden de una década. Esto se puede apreciar en la tabla 5.2.

f_c (MHz)	Potencia (TX = EBR15UFB1)	Potencia (TX = EBR15U1B1)
555,5	0,1961	0,01671
561,5	0,2765	0,02711
567,1	0,4256	0,04482

Tabla 5.2: Medida de potencia para distintas *USRP*

Se decide como forma de solucionar esta asimetría, asignarle una ganancia en transmisión a una de las dos *USRP*. Este parámetro lo fija el programa principal y se lo pasa a cualquiera de los dos módulos de transmisión.

Como último punto a tomar en cuenta se debe analizar qué valor devuelve el bloque en el momento en que el canal se encuentra vacío. Es de esperarse que no sea 0 y que dependa del canal en que se encuentre. En la figuras 5.3 y 5.4 se muestra el espectro de un canal centrado en $561{,}5MHz$, por un lado se tiene una transmisión modulada en *GMSK* y por el otro el canal vacío. Como se puede apreciar la potencia en el caso en que no hay transmisión en el canal es distinta de cero.

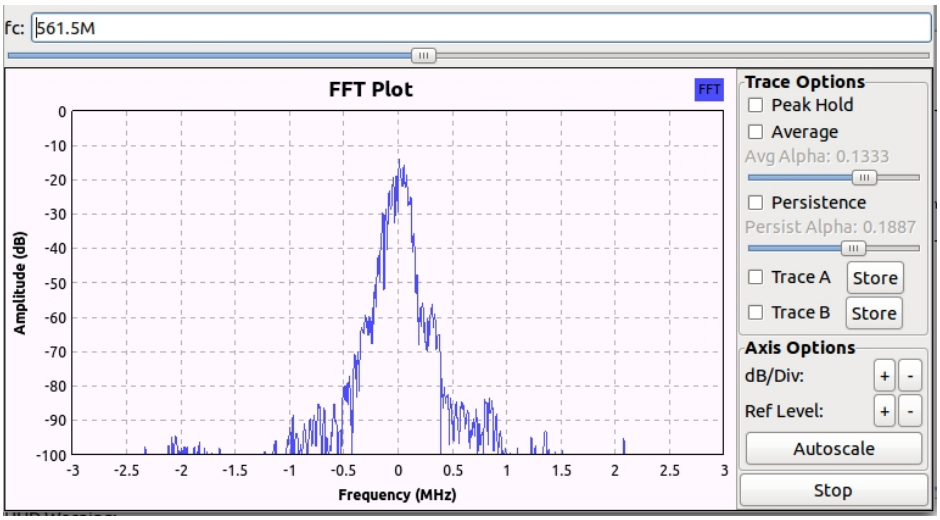


Figura 5.3: Espectro en el canal de $f_c = 561{,}5MHz$ con al transmisión principal presente

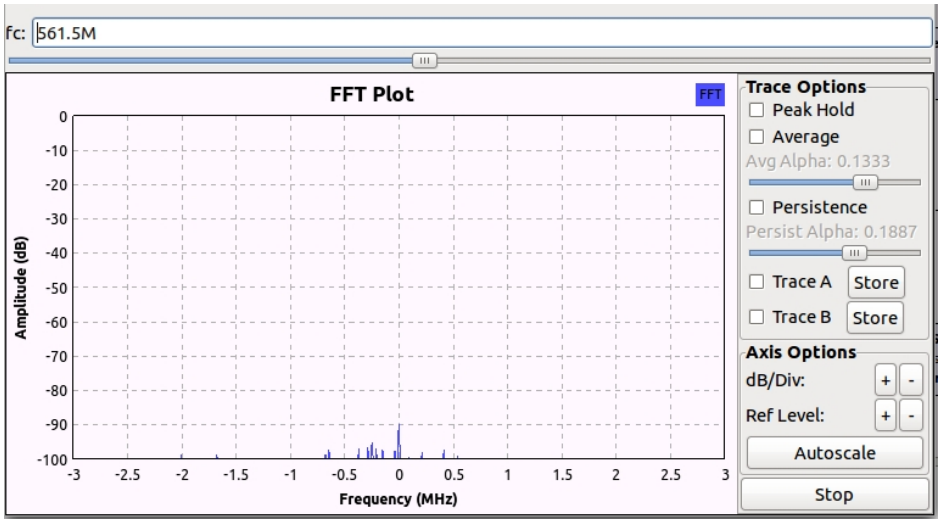


Figura 5.4: Espectro en el canal de $f_c = 561{,}5MHz$ con el canal vacío

Realizando una serie de mediciones (en las mismas hipótesis en la que se midió la salida de la función *level()*), se concluye que el máximo valor es del orden de 1×10^{-8}. Esto se detalla en la tabla 5.3

f_c (MHz)	Potencia
507,5	$1{,}956 \times 10^{-9}$
513,5	$7{,}078 \times 10^{-9}$
519,5	$9{,}779 \times 10^{-9}$
525,5	$1{,}863 \times 10^{-9}$
531,5	$4{,}005 \times 10^{-9}$
537,5	$2{,}142 \times 10^{-9}$
543,5	$7{,}078 \times 10^{-9}$
549,5	$2{,}049 \times 10^{-9}$
555,5	$6{,}585 \times 10^{-8}$
561,5	$1{,}043 \times 10^{-8}$
567,1	$7{,}665 \times 10^{-8}$
573,5	$2{,}822 \times 10^{-8}$
579,5	$7{,}637 \times 10^{-9}$
585,5	$6{,}798 \times 10^{-9}$
591,5	$4{,}843 \times 10^{-9}$
597,5	$1{,}676 \times 10^{-8}$
603,5	$1{,}1269 \times 10^{-9}$
609,5	$4{,}284 \times 10^{-9}$
615,5	$7{,}265 \times 10^{-9}$
621,5	$2{,}887 \times 10^{-9}$

Tabla 5.3: Medida de potencia para el canal vacío

Teniendo en cuenta que el valor medido varia con, la frecuencia, factores externos, en qué momento de la transmisión es medido y, más importante, dependiendo de qué *USRP* es el transmisor y cuál el receptor, se opta por tomar como criterio de diseño un valor de 1×10^{-4} como umbral. Es decir, frente a un valor mayor se considera que se encuentra el SU y la obtención de un valor menor se interpreta como ausencia del SU.

5.3. Análisis de tiempos

Los tiempos que se manejan durante el proceso que realiza la radio cognitiva, repercuten en el desempeño del proceso de encuentro. En la sección 4.3.3 se definen dos tiempos importantes para el reconocimiento (el tiempo de espera T_{espera} y el tiempo de confirmación T_{conf}) que se recogen aquí. En esta sección se procede a estudiar diferentes factores que influyen en la elección del valor de los mismos, y por ende en el desempeño de la radio implementada.

Teniendo como meta calcular el tiempo mínimo necesario para el reconocimiento de SU, se realiza este análisis de tiempos. El mismo comienza definiendo una serie de retardos que son de gran utilidad para la comprensión del problema y los cuales se medien para obtener valores experimentales en una prueba de transimisón/recepción.

Los tiempos definidos son los siguientes:

T_0:
(TX en f_c, RX en f_c)
Dato a enviar $\xrightarrow{\text{⏲}}$ se detecta la señal en recepción.

T_1:
(TX en f_c, RX en f_c)
Dato a enviar $\xrightarrow{\text{⏲}}$ se procesan los datos y se reconoce el usuario.

T_2:
(TX en f_c, RX en f_c')
Rx cambia a $f_c \xrightarrow{\text{⏲}}$ Rx detecta señal.

T_3:
(TX en f_c, RX en f_c)
Tx cambia a $f_c' \xrightarrow{\text{⏲}}$ Rx deja de sensar señal.

Y los resultados obtenidos en promedio son:

Retardo	tiempo (s)
T_0	0,0109
T_1	0,1462
T_2	0,0319
T_3	0,0628

Tabla 5.4: Medida de retardos

A continuación se realiza un análisis de la incidencia que pueden tener los tiempos T_{espera} y T_{conf} en el desempeño de la radio cognitiva. Este análisis es engorroso y puede requerir una lectura detenida, sin embargo su registro es sumamente necesario para el trabajo a futuro. El estudio se realiza comparando una serie de casos de forma progresiva.

1. $T_{espera} < T_1$
   Es claro que en este caso el encuentro no es posible de ninguna manera. Tomando en cuenta el mejor caso, o sea que ambos SU lleguen en el mismo momento al canal, nunca se llega al tiempo mínimo para realizar un reconocimiento.

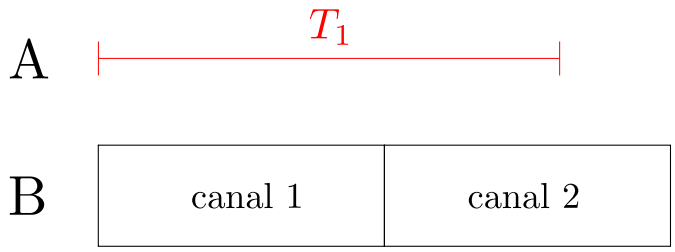


Figura 5.5: $T_{espera} < T_1$

   En la figura 5.5 se ve como cuando el SU A termina el tiempo que requiere para reconocimiento el SU B ya ha realizado el cambio de canal, habiendo este último pasado por un proceso análogo a A. Esto fija una primer cota de modo que $T_{espera} > T_1$

2. En el siguiente caso el SU A llega dentro de una ventana en la cual el SU B está en un tienpo t tal que $T_{espera} - T_1 < t < T_{espera}$.
   Esta situación implica que B dejará el canal ya que terminó su T_{espera}, mientras que el A pasado el T_1 reconocerá a B. Para que el SU A no reanude la transmisión asumiendo que se da el encuentro se implementa una etapa de confirmación, en la cual pasada el tiempo T_{conf} se sensa el canal. Si el resultado del sensado es nulo, se asume que efectivamente el SU B siguió saltando y así A continua en la búsqueda. Esta descripción se muestra en la figura 5.6

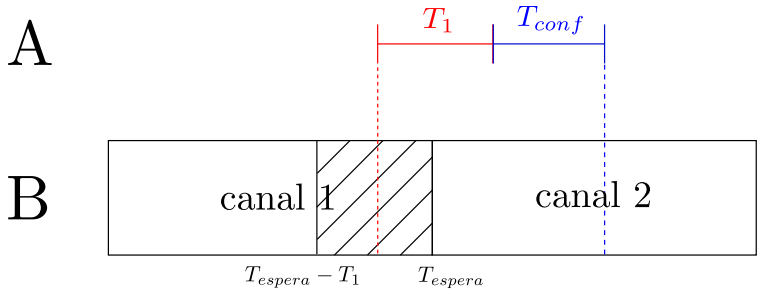


Figura 5.6: El usuario A llega al nuevo canal en una tiempo t, con $T_{espera} - T_1 < t < T_{espera}$

3. Un caso similar al anterior pero donde el *SU* A llega al canal cuando el *SU* B está en un tiempo t tal que $t < T_{espera} - T_1$ se muestra en la figura 5.7. En este caso A reconocerá a B en T_1 y comenzará su etapa de confirmación. Mientras, B pasa por un proceso similar y finalmente se logra el *Rendezvous*.

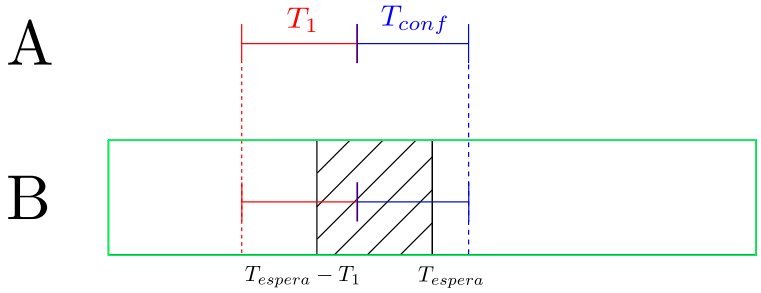


Figura 5.7: El usuario A llega al nuevo canal cuando B se encuentra en t, con $T_{espera} - T_1 < t < T_{espera}$

En esta situación como el reconocimiento se da antes de T_{espera} no se produce el cambio de canal. Esto se esquematiza en la figura 5.7 mediante el trazado verde para el *SU* B.

4. Un caso particular del caso 2) se da cuando el *SU* B toma como próximo canal el mismo que el anterior. Bajo estas circunstancias A sensará a B pasado su T_{conf} asumiendo que se da el encuentro. Sin embargo, A en su nuevo canal no logrará reconocer a B ya que este dejó de transmitir su señal para el reconocimiento al finalizar T_1. En este caso decimos que ocurre un falso encuentro encubierto.

   Hasta ahora se ha tomado en cuenta el retardo T_1 y se analiza su incidencia a partir de los casos presentados anteriormente. En dicho análisis se desprecian los retardos que introducen T_2 y T_3 particularmente. Bajo el mismo

esquema veamos de que manera entran en juego estos retardos a partir de la presentación de las siguientes situaciones.

5. Suponiendo que A llega en un momento en que B se encuentra en t tal que $T_{espera} - T_1 - T_2 < t < T_{espera} - T_1$ se tiene un caso como se muestra en la figura 5.8. Como el SU A viene de un cambio de canal no será sino hasta un tiempo T_2 despues de que se le da la orden de cambio que B sensará efectivamente la señal. Esto implica que el caso real es como el explicado en 2). De esta manera el área rayada la cual expresa un tiempo de inutilidad (a pesar que B estaba en el canal esté no tiene tiempo suficiente para reconocer a A) queda determinada en una ventana tal que $T_{espera} - T_1 - T_2 < t < T_{espera}$

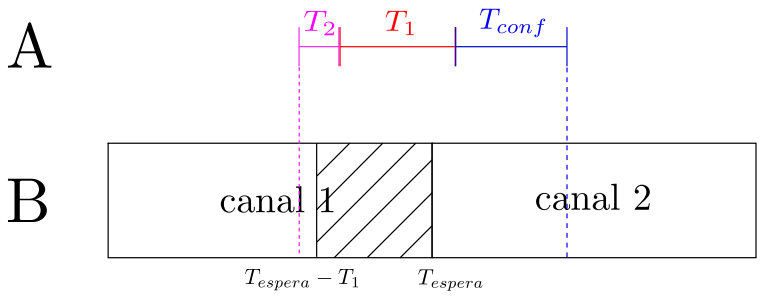


Figura 5.8: El usuario A llega al nuevo canal cuando B se encuentra en t, que tal que $T_{espera} - T_1 - T_2 < t < T_{espera} - T_1$

6. Para el análisis del retardo T_3 se supone lo siguiente. El SU A llega al canal en el momento que B se encuentra en t tal que $T_1' < t < T_1' + T_3$, donde $T_1' = T_{espera} - T_1 - T_2$. En este caso, si T_{conf} fuera menor que T_3, A sensaría a B cuando este ya se habría cambiado de canal. Esto impone una cota inferior para el T_{conf} tal que se debe cumplir que $T_{conf} > T_3$. La descripción de este problema se muestra en la figura 5.9.

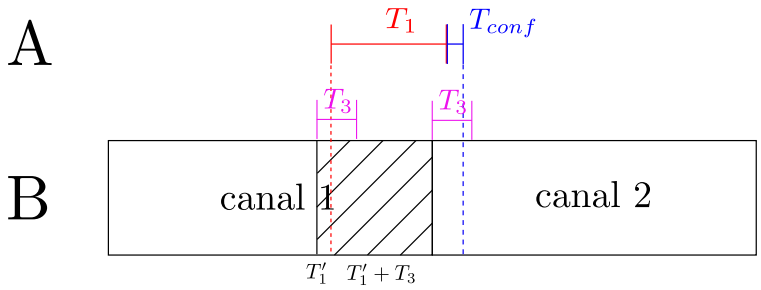


Figura 5.9: El usuario A llega al nuevo canal cuando B se encuentra en t, que tal que $T_1' < t < T_1' + T_3$

5.4. Resumen y Conclusiones

A pesar que el desarrollo de la radio cognitiva no ocupa el lugar central de este proyecto, se obtuvo con la misma una plataforma aceptable como soporte para la implementación y el testeo de los algoritmos bajo estudio.

En esta sección fueron presentados aquellos factores que influyen más fuertemente en el desempeño de la misma. En primer lugar se tienen los **falsos encuentros** los cuales impactan directamente en el análisis del tiempo que les lleva a los usuarios realizar el encuentro. A pesar que el estudio de los algoritmos de encuentro se basa en la medida de cantidad de saltos (sin importar estríctamente cuánto demoran en saltar) se entiende que el vínculo tiempo/salto es un parámetro importante a optimizar.

A nivel de la implementación se tiene como ventaja haber obtenido un mecanismo por el cual estos errores son mitigados, sin embargo, como se explica en la sección 5.1 un subconjunto de los falsos encuentros (los llamados "encubiertos") no es controlable, al menos con el esquema con que fue programada la radio cognitiva. Esta causa genera errores como ya se ha explicado.

En segundo lugar se trataron los inconvenientes asociadas al **sensado** y a la **fijación de umbrales de potencia**. Este aspecto es el que presenta menos incidencia en el desempeño de la radio cognitiva, pero el cual implicó un intenso período de estudio y pruebas de ensayo. La inclusión del problema de sensado de usuarios está fuera del alcance de este proyecto, sin embargo bajo la necesidad de contar con un mecanísmo para poder descriminar la presencia (o ausencia) de usuarios es que se recurre a este método. El resultado en este sentido es satisfactorio, pero se reconoce que tiene un alto grado de influencia del *hardware* y condiciones externas, lo cual no lo hace óptimo como solución. Es claro que técnicas más sofisticadas que implementen el sensado de las señales de usuarios harían un mejor diseño de las radios.

En último lugar se realiza un análisis de tiempos a partir de la selección de ciertos procesos claves (cambio de canal, comienzo de transmisión, etc). El estudio que se presenta es teórico e incluye los tiempos medidos con el equipamiento que

se contó. La justificaciones que se muestran son exahustivas, sin embargo en el desarrollo práctico se encontraron límites temporales que aseguran el buen funcionamiento de las radios y que parecen ser mayores de lo que se esperaría. Si bien el impacto de este punto implica tener un **proceso "lento"**, no afecta el análisis de los algoritmos ya que la unidad de medida que se estudia principalmente se basa en saltos y no en cuánto estos duren.

Una observación importante es que estos tres problemas se encuentran presentes en la implementación de la función de reconocimiento. Como se explica en la sección 5.3 se entiende que mejores técnicas para el reconocimiento de SU, que tal vez ignoren o utilicen mejor los factores mencionados, mejoraría sustancialmente el desempeño de las radios cognitivas.

Capítulo 6

Desempeño de la Implementación de Algoritmos de Encuentro en USRP

Con la radio cognitiva funcional y los resultados de los algoritmos, teóricos y de simulaciones en *Matlab*, se está en condiciones de verificar el desempeño de los mismos en la plataforma desarrollada. Para medir el desempeño de los Algoritmos es necesario atravesar dos tareas. En primer lugar registrar los datos relevantes para determinar la performance (sección Adquisición de datos 6.1 y en segundo lugar realizar el procesamiento de dicha información (seccion Tratamiento de datos 6.2).

6.1. Adquisición de datos

Para medir el desempeño de la implementación realizada se necesitan relevar tres datos: **cantidad de saltos hasta el encuentro**, **cantidad de falsos encuentros reconocidos** y **el canal en el que se da el encuentro**. Para realizar esta adquisición se agregan líneas de código al programa principal para que se escriba un archivo de texto cada vez que se da un encuentro.

El *software* se ejecuta en los dos equipos durante el tiempo necesario para obtener alrededor de 500 encuentros en cada uno, obteniéndose aproximadamente 1000 datos para cada algoritmo. Los datos se adquieren con los dispositivos distribuidos como se muestran en la imagen 6.1, a menos de 30 cm de distancia entre las antenas.

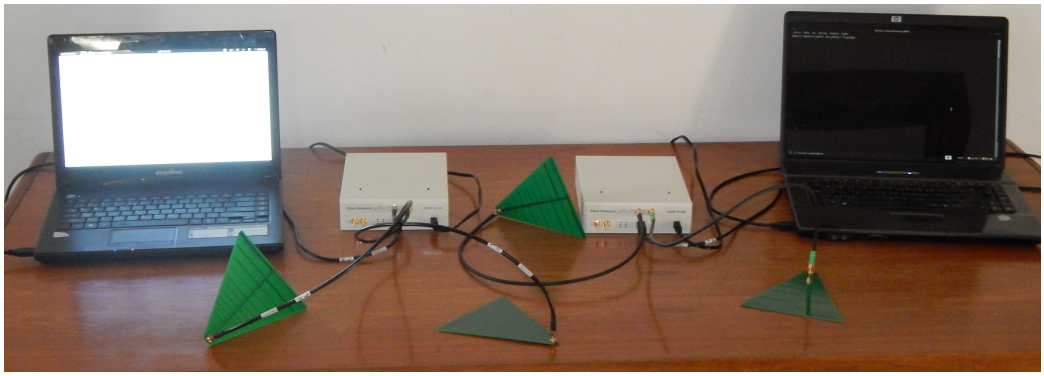


Figura 6.1: Banco de pruebas

6.2. Tratamiento de datos

Como se menciona en secciones anteriores, hay casos en que los usuarios se cruzan en un canal sin que den los tiempos para que comience la comunicación. Estos casos se llaman **falsos encuentros** y se distinguen dos tipos, detectados y encubiertos. A nivel del análisis de datos es importante tener en cuenta que los primeros resetean el contador de saltos mientras que los últimos no. La contrapartida de los falsos encuentros encubiertos en el funcionamiento significa un problema grave, ya que cuando estos suceden (además de disminuir el desempeño) una de las radios comienza a trasmitir sin su contraparte en el canal y se pierden datos.

Se debe considerar que el desempeño de los algorítmos se ve influido por el desempeño de la radio cognitiva. Por ejemplo, un falso encuentro encubierto se registrará como un encuentro, cuando claramente esto se da por el funcionamiento de la radio y no por las propiedades objetivas del algorítmo.

Tomando en cuenta esto, se realizan tres tipos de tratamientos de datos para medir distintos desempeños: el del conjunto de radio y algoritmo, el del conjunto pero si no sucedieran los falsos encuentros encubiertos y por ultimo, el de los algoritmos sin tener en cuenta el funcionamiento de la radio. A continuación se explica cómo se realiza el tratamiento para obtener dichas medidas de desempeño.

- **Desempeño conjunto:** En este caso se mide el desempeño real de todo el conjunto implementado. Por lo tanto, como los falsos encuentros encubiertos no resetean el contador de saltos, se deben sumar los saltos al primer encuentro real posterior al falso encuentro encubierto. Por ejemplo, una línea arroja que se dió un encuentro en el canal 12 tras 15 saltos. El encuentro anterior se dió tras 12 saltos pero es detectado como falso encuentro encubierto. Entonces el encuentro en el canal 12 verdaderamente se dió tras $15 + 12 = 27$ saltos.

- **Desempeño conjunto mejorado:** Este caso considera que se mejora el problema de los falsos encuentros encubiertos y se cuentan los mismos como encuentro reales, mientras que los falsos encuentros detectados no se cuentan. Por lo tanto se estudia el desempeño de del la radio implementada si evitaran

los falsos encuentro encubiertos, lo cual sería el primer problema a mejorar en trabajos futuros.

- **Desempeño de Algoritmo:** En este caso se quiere medir solamente el desempeño de los algoritmos, sin tomar en cuenta los problemas en la implementación de la radio. Para hacer esto se consideran los dos tipos de falsos encuentros como encuentros reales. Por lo tanto, se obtiene el mejor desempeño que se podría tener mejorando el desarrollo y también los datos más adecuados para comparar con las simulaciones realizadas en *MatLab* expuestas en el capítulo 3. Es importante aclarar que, debido a la forma en que se estima la cantidad de saltos, el análisis tiene sentido solamente si el parámetro que se compara es el promedio de TTR.

Estos tres tipos de tratamientos de datos se realizan para cada uno de los algoritmos, dando así una idea general del desempeño de la radio desarrollada y de los algoritmos en cuestión.

6.2.1. Desempeño conjunto

A continuación se presentan los histogramas y la distribución empírica del desempeño conjunto de la radio implementada en los *USRP* y de los algoritmos.

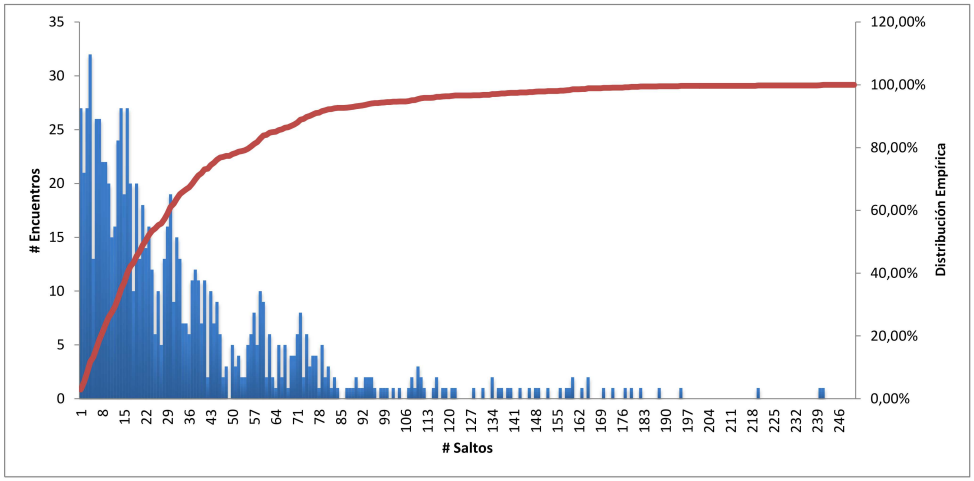


Figura 6.2: Histograma de desempeño conjunto con distribución empírica para MCA

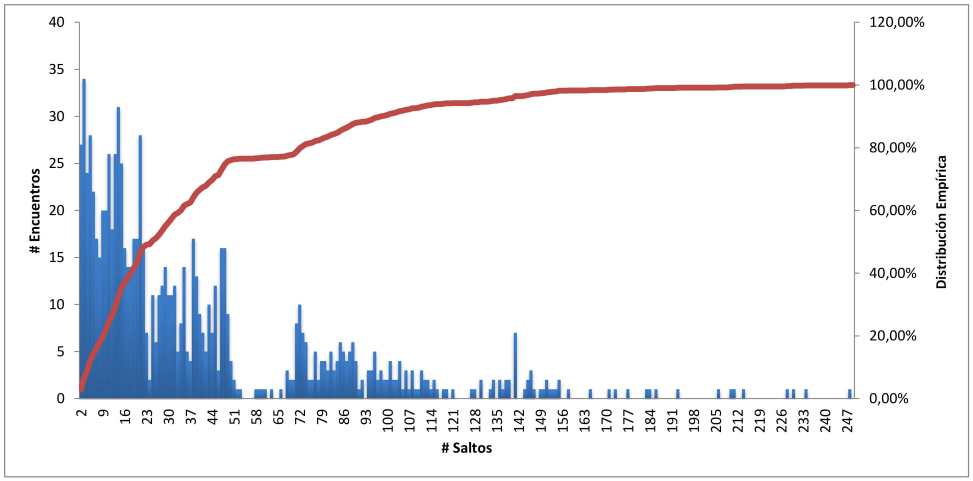


Figura 6.3: Histograma de desempeño conjunto con distribución empírica para JS

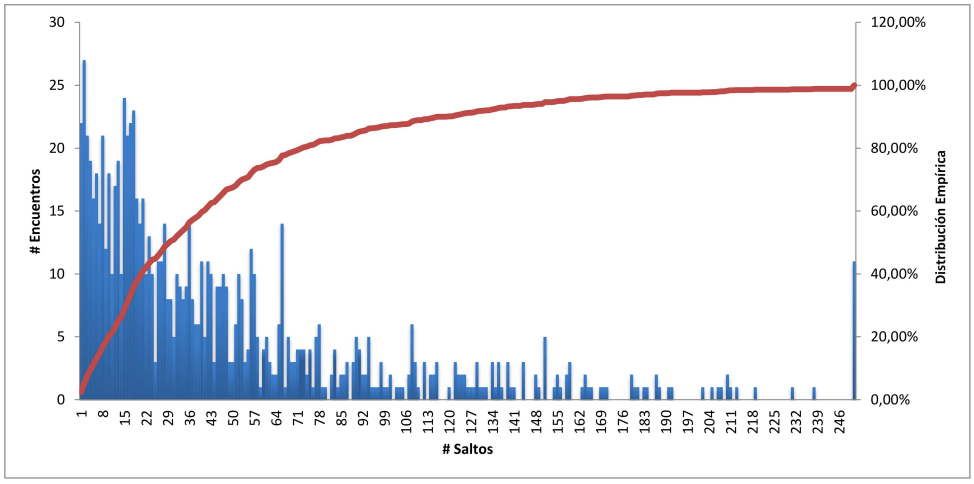


Figura 6.4: Histograma de desempeño conjunto con distribución empírica para MMCA

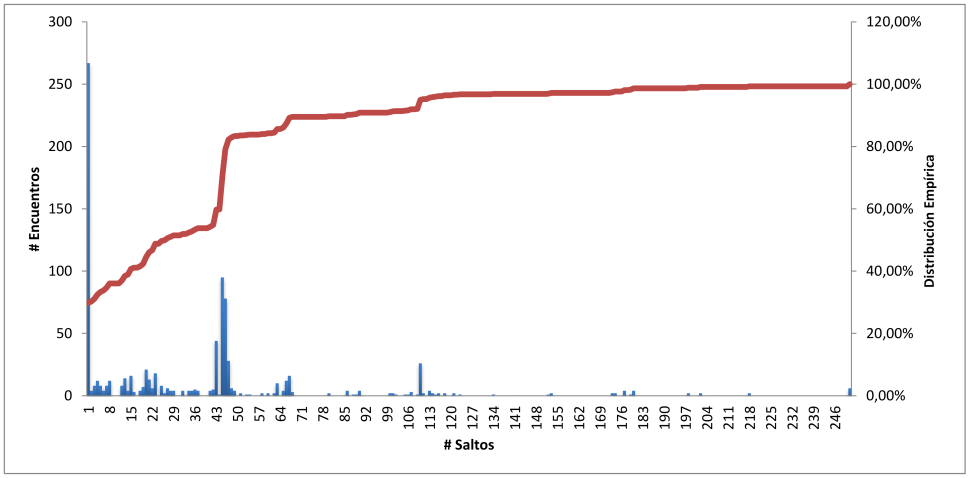


Figura 6.5: Histograma de desempeño conjunto con distribución empírica para CRSEQ

Como se puede apreciar en los histogramas presentados, se añade la distribución empírica, esto es, qué porcentaje de la cantidad total de encuentros dados para cada algoritmo se da en menos de x cantidad de saltos. Ahora, a modo de resumen se presenta los datos para los cuatro algoritmos.

Algoritmo	TTR promedio	MTTR	TTR< 60
Jump-Stay	40	248	76, 8 %
MCA	34	241	83, 8 %
MMCA	54	1744	74, 2 %
CRSEQ	36	338	84, 2 %

Tabla 6.1: Datos de desempeño conjunto en implementación

El valor más importante a comparar entre algoritmos es el *TTR* promedio, ya que da una idea general del desempeño del algoritmo. Además junto con este valor se calcula el *MTTR*, para así obtener un mejor panorama del algoritmo en su peor caso.

Por lo tanto en esta comparación se podría decir que el algoritmo que tiene mejor desempeño es el *MCA* seguido por *CRSEQ*, a continuación *JS* y por último *MMCA*. Basandose en que el *MTTR* de todos los algoritmos menos de *MMCA* son del mismo orden y que para este último el porcentaje de éxito en menos de 60 saltos es el más bajo, se concluye que *MMCA* es el que presenta un peor desempeño mientras que los otros algorítmos tienen un desempeño similar.

6.2.2. Desempeño conjunto mejorado

Como se explica al comienzo de la sección, en este procedimiento de tratamiento de datos se consideran los falsos encuentros encubiertos como encuentros reales. El registro de estos resultados apunta a la mejora futura en el sentido de eliminar estos falsos encuentros. En ese caso se deberían obtener resultados de performance como se presentan. En los siguientes histogramas se muestra el desempeño conjunto simulando dicha mejoría.

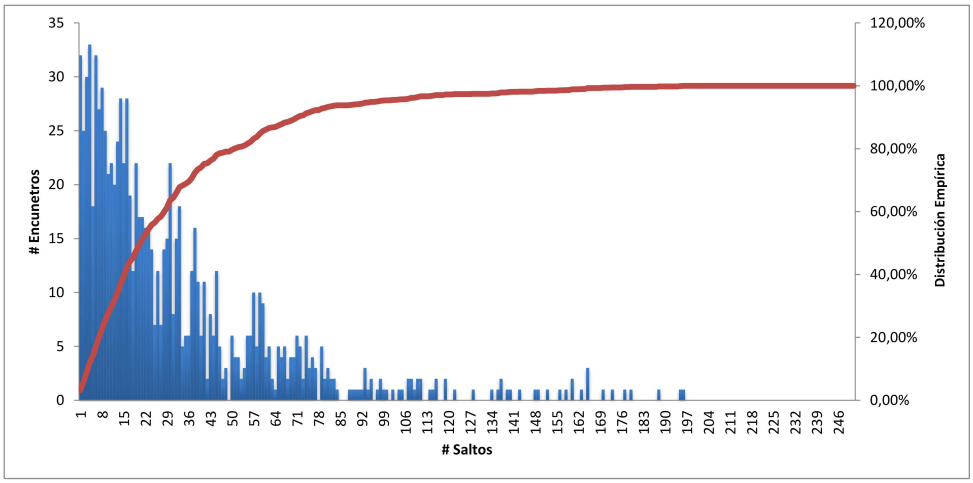


Figura 6.6: Histograma de desempeño conjunto mejorado con distribución empírica para MCA

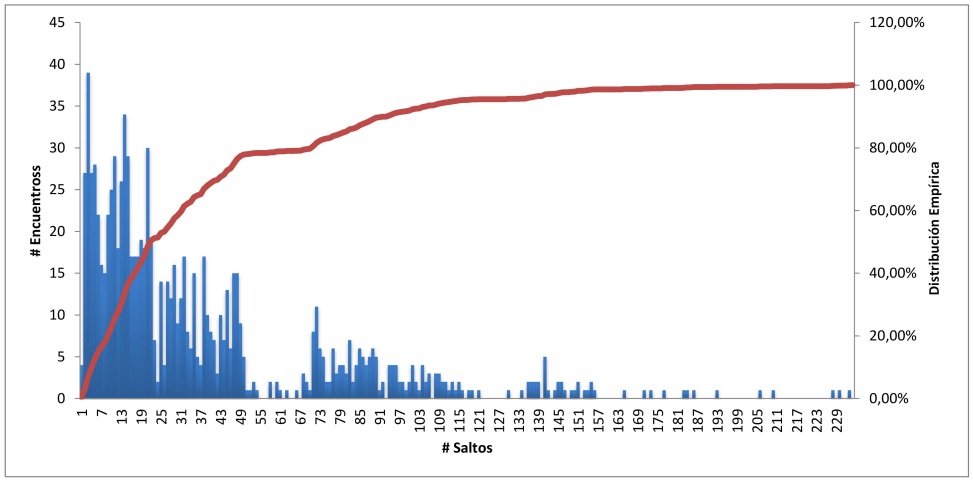


Figura 6.7: Histograma de desempeño conjunto mejorado con distribución empírica para JS

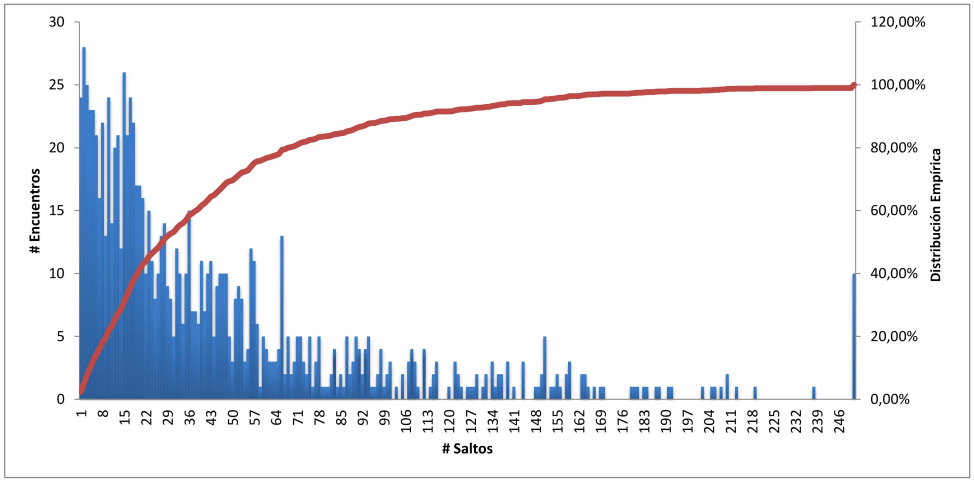


Figura 6.8: Histograma de desempeño conjunto mejorado con distribución empírica para MM-CA

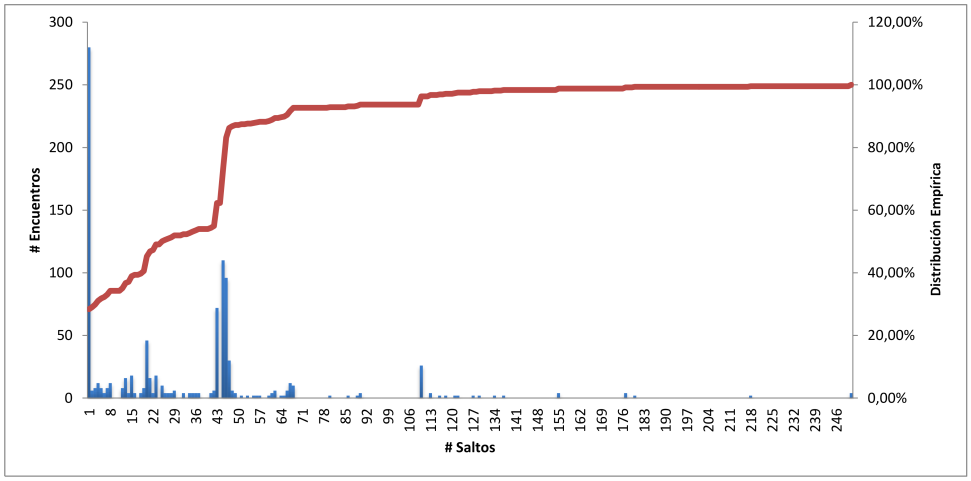


Figura 6.9: Histograma de desempeño conjunto mejorado con distribución empírica para CRSEQ

Como en la sección anterior, también se muestra la distribución empírica de encuentros. A partir de las gráficas, se obtiene los siguientes datos para comparar el desempeño de los algorítmos de encuentro.

Algoritmo	TTR promedio	MTTR	TTR< 60
Jump-Stay	38	233	$78,8\,\%$
MCA	32	196	$85,6\,\%$
MMCA	53	1741	$76,3\,\%$
CRSEQ	32	338	$88,4\,\%$

Tabla 6.2: Datos de desempeño conjunto mejorado en implementación

En este caso se puede apreciar que el orden de mejor desempeño entre los algoritmos se mantiene similar al desempeño conjunto. En este caso, como era de esperarse, el desempeño mejora (levemente) para los cuatro algorítmos, consecuencia de no tomar en cuenta los falsos encuentros.

6.2.3. Desempeño de algoritmo

Para esta clase de tratamiento de datos no tiene valor la realización de histogramas y distribución empírica, ya que los datos son manipulados, como se explica anteriormente, para obtener una estimación del desempeño del algoritmo sin tomar en cuenta la implementación de la radio. El valor de mayor importancia para la medida de cantidad de saltos es el TTR promedio, pero para ser coherente con los tratamientos de datos anteriores se presenta el $MTTR$ (teniendo en cuenta que dicho dato no es confiable, se debe utilizar con precaución al momento de sacar conclusiones).

Algoritmo	TTR promedio	MTTR
Jump-Stay	24	165
MCA	20	163
MMCA	34	1688
CRSEQ	21	137

Tabla 6.3: Datos de desempeño de algoritmos en implementación

En este caso también se mantiene el orden en cuanto al desempeño, *MCA* es el de mejor desempeño, luego *CRSEQ*, *JS* y por ultimo *MMCA*. Como se esperaba se obtienen mejorías claras en ambos valores para los cuatro algoritmos. Estos valores sí son comparables a los obtenidos en simulaciones realizadas en *MatLab*.

6.3. Comparación con simulaciones

Para realizar la comparación con los últimos datos presentados se realizan nuevas simulaciones en *MatLab* para los cuatro algoritmos bajo estudio. Con estas nuevas ejecuciones se obtienen datos bajo un modelo similar al utilizado en la adquisición de datos con la radio implementada en *USRP*. De esta manera, los datos presentados a continuación se obtienen simulando aleatoriamente la presencia de un usuario primario en cada nuevo canal al que se arriba, con la misma probabilidad que en la implementación.

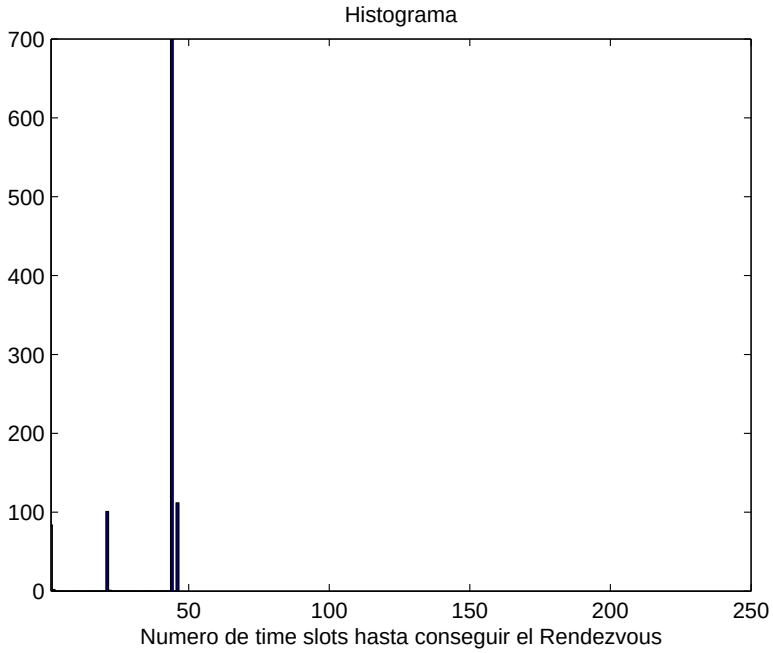


Figura 6.10: Histogramas de simulaciones en matlab de CRSEQ

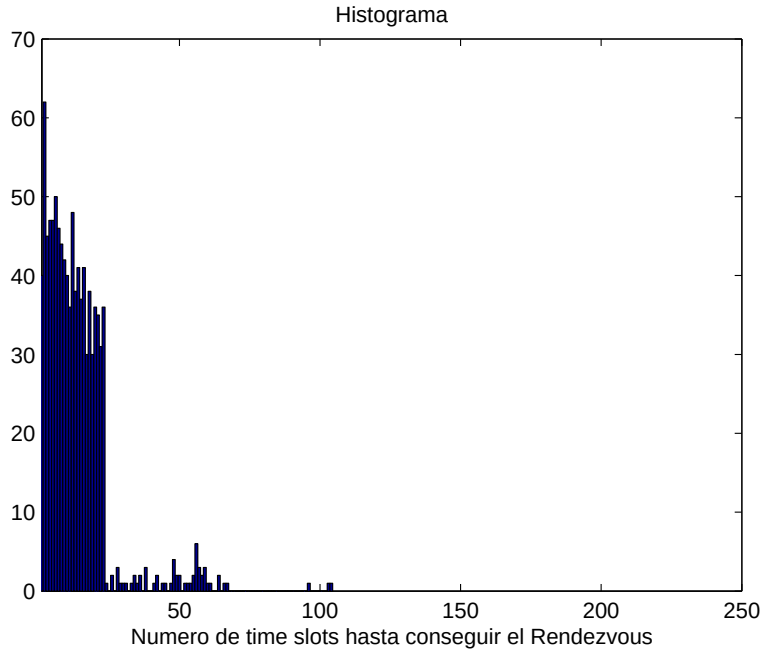


Figura 6.11: Histogramas de simulaciones en matlab de MCA

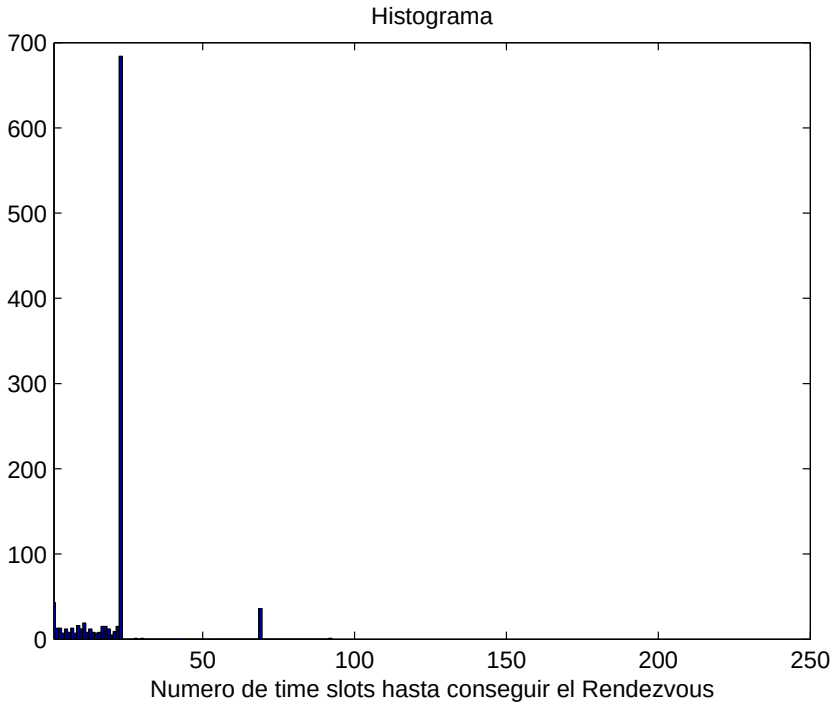


Figura 6.12: Histogramas de simulaciones en matlab de JS

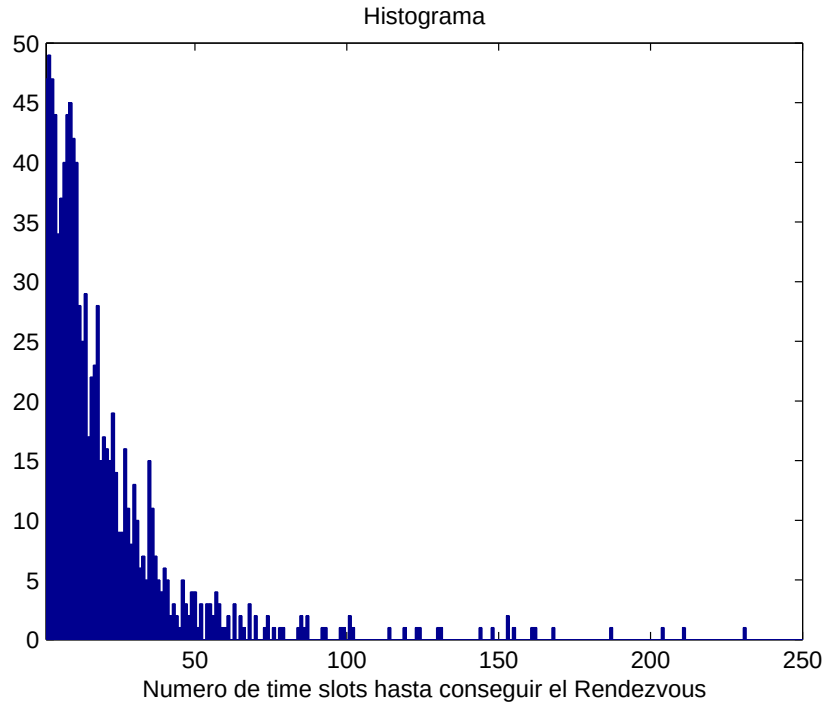


Figura 6.13: Histogramas de simulaciones en matlab de MMCA

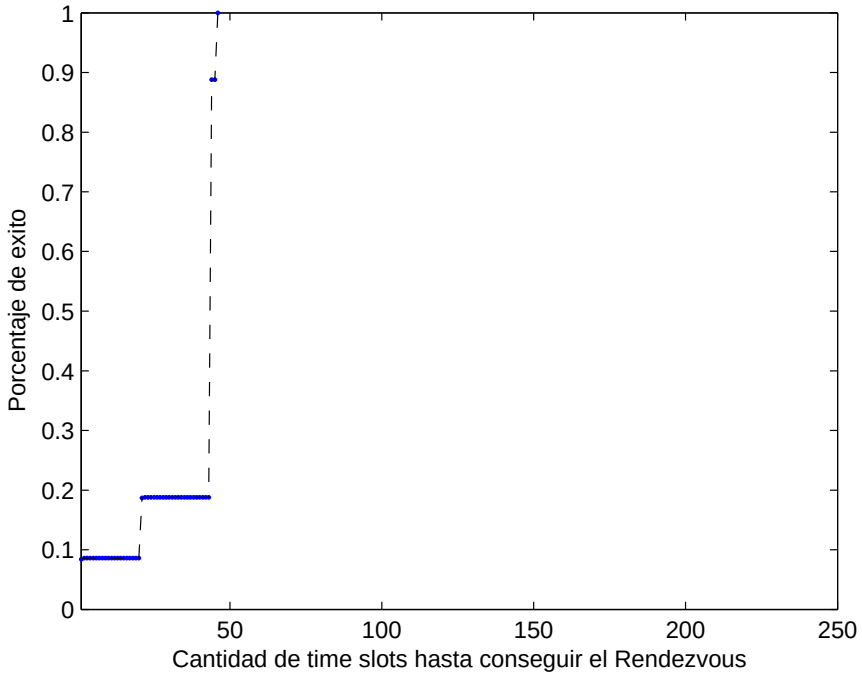


Figura 6.14: Distribución empírica de simulaciones en matlab de CRSEQ

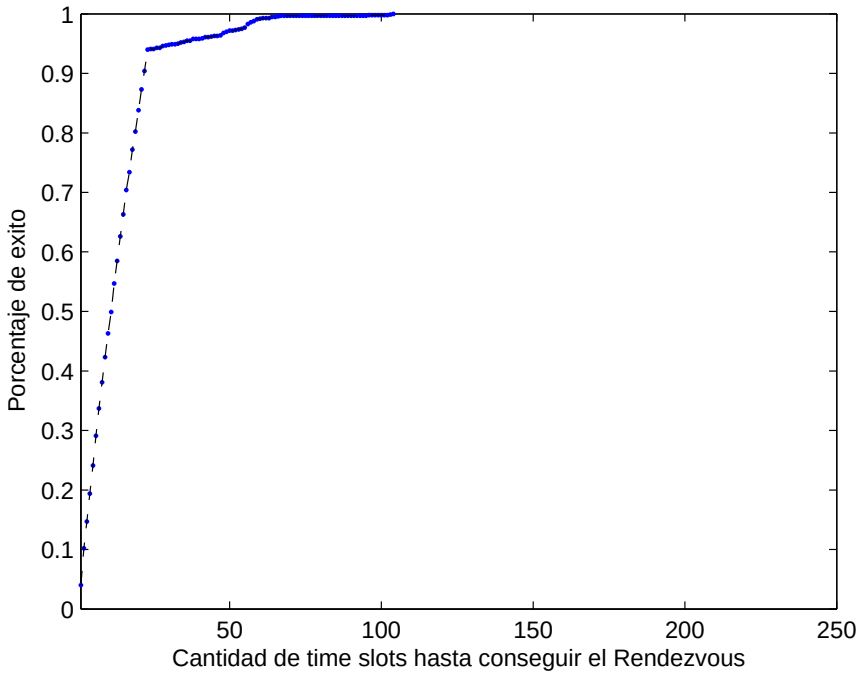


Figura 6.15: Distribución empírica de simulaciones en matlab de MCA

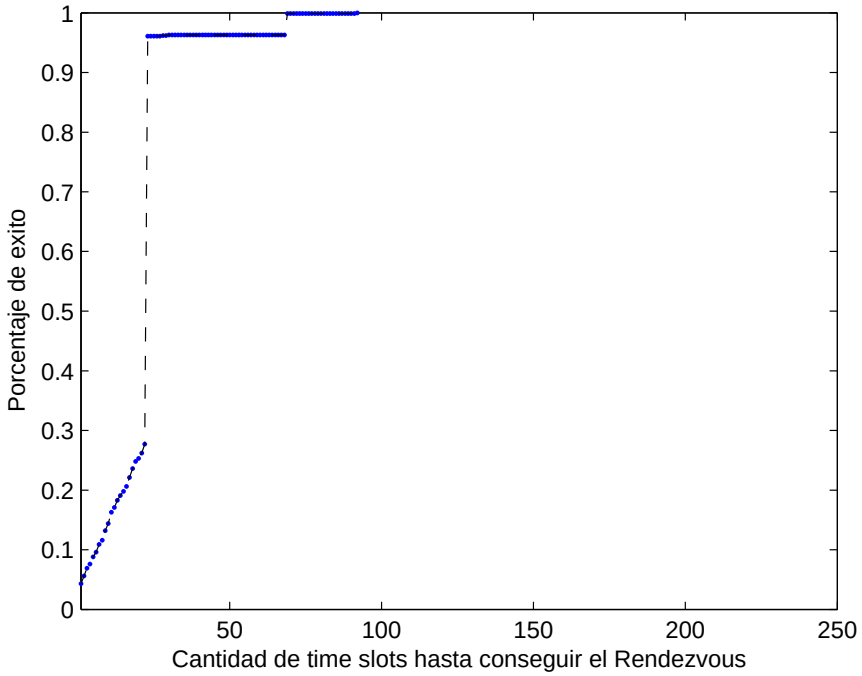


Figura 6.16: Distribución empírica de simulaciones en matlab de JS

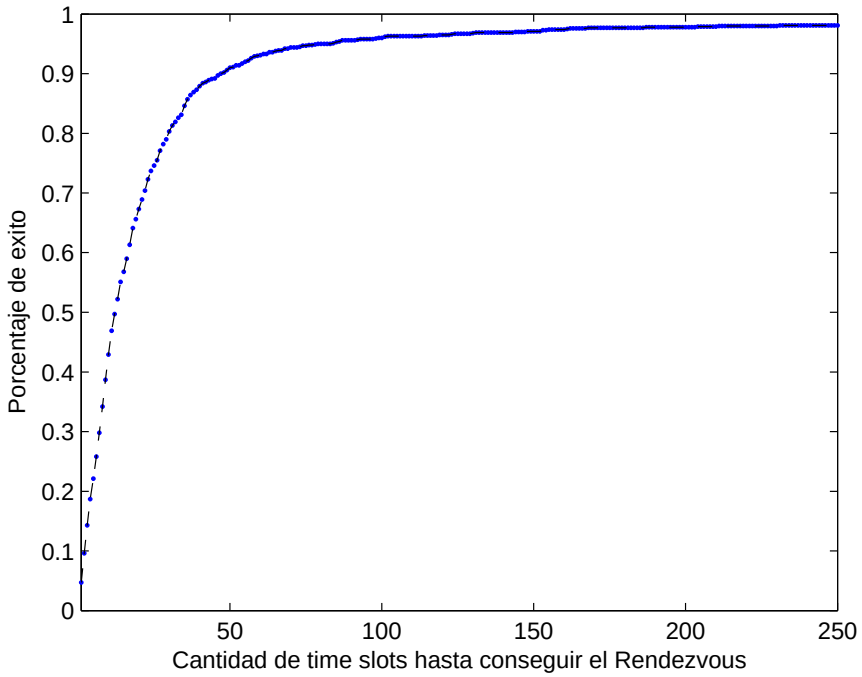


Figura 6.17: Distribución empírica de simulaciones en matlab de MMCA

Además de obtener estos nuevos histogramas, también se obtienen nuevos valores para el porcentaje de éxito. Contrastando estos resultados con la comparación realizada en el capítulo 3 queda clara la diferencia de las hipótesis utilizadas para la realización de las simulaciones. A partir de los histogramas de las simulaciones se obtienen los datos de la tabla 6.4. Estos valores son comparables con el tratamiento de datos que llamamos "Desempeño de Algoritmo". Para apreciar esta comparación se añade la columna *TTR promedio (USRP)* a la tabla 6.4.

Algoritmo	TTR promedio	MTTR	TTR promedio (USRP)
Jump-Stay	21	92	24
MCA	13	108	20
MMCA	39	2753	34
CRSEQ	38	46	21

Tabla 6.4: Datos de desempeño de algoritmos en simulaciones

Lo primero para destacar es que se mantiene *MCA* como el algoritmo de mejor desempeño y *MMCA* el de peor desempeño. La diferencia con los datos obtenidos de la radio implementada es el intercambio de posiciones entre *JS* y *CRSEQ*. Esto se puede atribuir al hecho que *CRSEQ* es un algoritmo secuencial, como se explica en la sección 3.2. En consecuencia se genera una lista de canales fija con el orden de salto predefinifo y esta lista es la misma para ambos usuarios. Por lo tanto, si

los usuarios deciden cambiar de canal en el mismo momento se encontrarán en el primer canal de la lista, lo cual también explica la forma que tienen los histogramas de la radio implementada para *CRSEQ*, con un porcentaje de éxito de alrededor de 50 % para un salto. En la simulaciones en *MatLab* se programa el código de modo que no se de la sincronización, de lo contrario siempre que no haya un *PU* ocurriría el encuentro en el primer canal al que se salta. Esto no sucede en la implementación porque la misma radio desarrollada introduce los tiempos necesarios para evitar la sincronización, pero en distinta medida que lo estipulado para la simulaciones. Por su parte, *JS* y *MCA* son algoritmos que generan la secuencia aleatoriamente, lo que se traduce en valores similares en la implementación y en la simulación.

Por lo tanto con todos los datos presentados se concluye que el mejor algoritmo de los estudiados es el *MCA* y el peor es el *MMCA*. Mientras que con un desempeño similar se encuentran el *CRSEQ* y *JS*, con una ventaja para el primero.

Comparando con la estimación de desempeño realizada en el capítulo 3, se observa que la misma estaba acertada, ya que se esperaba que *MCA* y *CRSEQ* fueran los algoritmo de mejor desempeño en la implementación. Además, se destacan los resultados obtenidos para el desempeño de JS, ya que presenta una performance mucho mejor a la estimada en el capítulo 3 y por lo tanto similar a lo especificado por los autores.

Capítulo 7

Conclusiones

Mediante este proyecto se logró realizar el estudio de cuatro algoritmos que permiten el encuentro de dos usuarios en un canal común luego de haber abandonado la comunicación inicial. Además de esto se lograron implementar los mismos en *GNU radio* con la ayuda de las *USRP*, obteniendo así una plataforma para probar los algoritmos en un ambiente real.

El estudio de los algoritmos constó de tres procesos que aportaron distintas fuentes de información. En primera instancia las **propuestas teóricas** por las cuales los autores de distintas publicaciones presentan el funcionamiento del algoritmo así como también sus resultados de desempeño. En segundo lugar se lograron estudiar los algoritmos mediante la **programación de los mismos en *Matlab***. Esto, como primer acercamiento permitió definir criterios del modelo bajo el cual se trabajaría, muestra que el planteo de la algoritmia en las publicaciones se condecía con los resultados y no presentaba mayor complejidad a nivel de programación; a su vez logra arrojar los primeros resultados cualitativos y cuantitativos sobre los algoritmos. Por último, como fuente útil de información, se obtiene la **implementación en *USRP*** y el desarrollo práctico de los diferentes métodos de encuentro en una comunicación experimental. De esta manera, se puede decir que el proyecto constó de dos grandes tareas: la **comparación de los algoritmos** y el **desarrollo e implementación de la plataforma** para probarlos.

De la primera se obtiene buenos resultados. En la comparación de *TTR* y *MTTR* obtenido mediante las simulaciones en *Matlab* con respecto a los pronunciados teóricamente se encuentra una buena correspondencia, sobre todo teniendo en cuenta que el modelo que se simuló, según se explica en el capítulo 3, no es exactamente igual a ninguno de los que se proponen en los distintos papers. Al mismo tiempo, de la comparación de *TTR* obtenido en *Matlab* y obtenido en *GNU radio/USRP* se aprecia también una buena correspondencia. Con respecto a esta última comparación se observa una diferencia con el *MTTR* medido, la cual queda claramente justificada a través de casos excepcionales que suceden en la comunicación implementada en los *USRP*.

La segunda gran tarea implica el desarrollo e implementación de la plataforma de comunicación. Se invierte un gran esfuerzo y dedicación debido al trabajo práctico tanto con *GNU* radio como con *USRP*. Los pilares en los cuales se trabajó fueron: el establecimiento de una comunicación utilizando una modulación digital, sensado de niveles de potencia de señales y procesamiento de la señales particulares para el reconocimiento de usuarios. En estos temas se logra una profundidad de estudio necesaria para implementar correctamente la plataforma y, cuyo resultado y conocimiento aprendido quedan registrados en esta documentación y en el código de software desarrollado.

Un desafío importante fue enfrentarse a los **falsos encuentros**. Su reconocimiento, justificación y manejo influyeron en el curso del proyecto añadiendo un grado de dificultad extra con el cual no se contaba en principio. En la documentación el tema se trata exahustivamente de modo de clarificar de qué manera se puede explorar su solución a futuro.

Con respecto a los algoritmos, ***MCA*** es el que obtuvo un mejor desempeño en cualquiera de los escenarios en los que se logró probar. En el caso de ***JS*** en las simulaciones no se obtienen resultados buenos, a pesar que la idea detrás del algoritmo es la más interesante, sin embargo, en la implementación en *GNU radio/USRP* se obtiene una mejora sustancial del rendimiento, posicionándolo con valores de *TTR* aceptables y de acuerdo con lo que se esperaba. Con ***CRSEQ*** se logran resultados dentro de los esperados. En este caso los valores de *TTR* son buenos pero el algoritmo deja una serie de dudas provenientes de cómo está conformado. Las secuencias preestablecidas y el carácter determinístico del *channel hopping* hacen que frente a ciertas situaciones (canales particulares ocupados) el desempeño se degrade sustancialmente. En último lugar ***MMCA*** termina teniendo el rendimiento más pobre, esto a priori no presenta una sorpresa, pues ya desde las simulaciones en *MatLab* es el que presenta el peor rendimiento.

Quedan varios desafíos como trabajo futuro. Desde una óptica general se puede ampliar el número de algoritmos a estudiar, para esto se cuenta con este trabajo como antecedente y herramienta para ser utilizada en la prueba de los mismos. Queda solamente el trabajo de programación del algoritmo en sí, ya que la plataforma para probarlo se encuentra realizada.

No es menor el problema de reconocimiento de usuarios secundarios, el cual se implementó de una manera práctica teniendo en cuenta el alcance del proyecto pero quedando como problema abierto a ser estudiado e implementado de una manera óptima. Este punto, como se ha mencionado en el desarrollo de la documentación, tiene un impacto directo en la evaluación de los algoritmos lo cual lo torna importante. El proyecto deja sentada las bases en la utilización de ciertos recursos de *GNU radio*, pero en los cuales se puede profundizar más para obtener distintos resultados. Concretamente se pueden estudiar otros flujos de comunicación, utilizando distintas modulaciones. Como recomendación queda el hecho de

utilizar la plataforma con *OFDM* (*Orthogonal Frequency Division Multiplexing*); también se puede investigar más en el funcionamiento de algunos bloques particulares (sensado de potencia, manejo de mensajes y colas, etc).

En síntesis, la experiencia fue enriquecedora para el grupo desde varios puntos de vista, desde el académico y técnico hasta el personal, pero principalmente es reconfortante alcanzar los objetivos propuestos en tiempo y forma. También vale la pena volver a resaltar el hecho de proveer al cliente un plataforma modular para que la investigación en el área de las Redes de Radios Cognitivas prosiga.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] http://gnuradio.org/redmine/wiki/gnuradio.

[2] http://www.atsc.org/cms/.

[3] http://www.ettus.com/.

[4] http://www.globalsecurity.org/military/systems/ground/jtrs.htm.

[5] http://www.ursec.gub.uy.

[6] M. Nekovee A.M. Wyglinski and Y.T. Hou. *Cognitive Radio Communications and Networks: Principles and Practice*. 2009.

[7] Carlos Cordeiro, Kiran Challapali, Dagnachew Birru, and Sai Shankar N. Ieee 802.22: An introduction to the first wireless standard based on cognitive radios.

[8] J. M. Shin D. M. Yang and C. H. Kim. Deterministic rendezvous scheme in multi-channel access networks. *Electronics Letters*, 46, 2010.

[9] FCC. Notice of proposed rulemaking (nprm 03 322): Facilitating opportunities for flexible, efficient and reliable spectrum agile radio technologies. *ET Docket No. 03 108*, 2003.

[10] Xiaowen Chu Hai Liu, Zhiyong Lin and Yiu-Wing Leung. Jump-stay rendezvous algorithm for cognitive radio networks. *IEEE Transactions on parallel and distributed system*, 23, 2012.

[11] D. Yang J. Shin and C. Kim. A channel rendezvous scheme for cognitive radio networks. *IEEE Communications Letters*, 2010.

[12] Andres Navarro Cadavid Julio Hector Aguilar Renteria. Radio cognitiva - estado del arte. 9, 2011.

[13] J Mitola. The software radio. *IEEE National Telesystems Conference*, 1992.

[14] R. W. Thomas N. C. Theis and L. A. DaSilva. Rendezvous for cognitive radios. *IEEE Transactions on Mobile Computing*, 10, 2011.

[15] Nicholas C. Theis. The modular clock algorithm for blind rendezvous, 2009.

[16] X.W. Chu Z.Y. Lin, H. Liu and Y.-W. Leung. Jump-stay based channel hopping algorithm with guaranteed rendezvous for cognitive radio networks. *IEEE INFOCOM*, 2011.

[17] X.W. Chu Z.Y. Lin, H. Liu and Y. W. Leung. Taxonomy and challenges of rendezvous algorithms in cognitive radio networks. *IEEE ICNC*, 2012.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el jueves 10 julio, 2014.
http://iie.fing.edu.uy/