**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**

**FACULTAD DE INGENIERÍA**

# Implementación de la capa MAC en un simulador de eventos discretos para el protocolo 802.15.4 basado en SimPy

**TESIS**

Que para obtener el título de

**Ingeniero en Telecomunicaciones**

**P R E S E N T A**

De la Cruz Meneses Alejandro

**DIRECTOR DE TESIS**

Dr. Luis Francisco García Jiménez

**Ciudad Universitaria, Cd. Mx., 2020**

# Agradecimientos

Agradezco a mi familia en general, todo el apoyo brindado durante mi formación académica, pero en especial a mi tío Pedro, a mi hermana Jazmín y a mi mamá, ya que sin su guía ni consejos, no habría podido alcanzar esta meta.

También agradezco a mis amigos y compañeros que me brindaron su ayuda, especialmente a Cinthya, José Alberto, Luis, Javier y Edgar.

Asimismo, las aportaciones y recomendaciones que surgieron durante la revisión de este trabajo para mejorarlo por parte de los sinodales. Por otro lado, agradezco el apoyo brindado por parte del proyecto DGAPA-PAPIIT IA105520.

# Índice general

# Índice de figuras

# Índice de tablas

# Acrónimos

**ACK** Acknowledgement

**AODV** Ad hoc On-Demand Distance Vector

**API** Application Programming Interface

**CA Collision Avoidence**

**CAP** Contention Access Period

**CCA** Clear Channel Assessment

**CFP** Contention-Free Period

**CFS** Contention-Free Slot

**CRC** Cyclic Redundancy Check

**CS** Carrier Sense

**CSMA** Carrier Sense Multiple Access

**CTS** Clear to Send

**CW** Contention Window

**DES** Discrete-Event Simulation

**DSSS** Direct Sequence Spread Spectrum

**ED** Energy Detection

**FCS** Frame Check Sequence

**FFD** Full-Function Device

**FIFO** First in, First out

**GHz** Gigahertz

**GTS** Guaranteed Time Slot

**GUI** Graphical User Interface

**IDE** Integrated Development Environment

**IEEE** Insitute of Electrical and Electronics Engineers

**IFS** Inter Frame Spacing

**IoT** Internet of Things

**ITU** International Telecommunication Union

**LIFS** Long Inter Frame Spacing

**LQI** Link Quality Indicator

**LR-WPAN** Low-Rate Wireless Personal Area Network

**MAC** Media access control

**MFR** MAC Footer

**MHR** MAC Header

**MHz** Megahertz

**MLME** MAC Sublayer Management Entity

**MPDU** MAC Protocol Data Unit

**MQTT** Message Queue Telemetry Transport

**NetSim** Network Simulator and Emulator

**NS2** Network Simulator 2

**NS3** Network Simulator 3

**OMNET ++** Objective Modular Network Testbed in C++

**OPNET** Optimized Network Engineering Tool

**OSI** Open System Interconnection

**PHR** PHY header

**PHY** Physical Layer

**PSSS** Parallel Sequence Spread Spectrum

**RFD** Reduced-Function Device

**RTS** Request to Send

**SAP** Service Access Point

**SHR** Synchronization Header Short Inter Frame Spacing

**WSN** Wireless Sensor Networks

# Resumen

La creación de Internet ha revolucionado muchos ámbitos de la vida cotidiana, uno de ellos es la forma de comunicar a los individuos a nivel local, regional e incluso mundial. Últimamente se busca ir más allá a través del concepto de *Internet de las cosas (IoT)*, el cual se enfoca en la interconexión de diferentes dispositivos con el fin de automatizar distintas tareas que se presentan en el día a día. Sin embargo, para la implementación de este tipo de redes se necesitan protocolos de comunicación que les permitan a los dispositivos establecer enlaces confiables y robustos para poder transmitir información. Dentro de los protocolos o estándares ocupados en la actualidad para el propósito descrito se encuentran IEEE 802.15.4, 6LoWPAN, ZigBee, Message Queue Telemetry Transport (MQTT), entre otros.

Comúnmente, antes de diseñar o implementar una red se realizan diferentes pruebas de comunicación para medir el rendimiento a través de las diferentes capas que componen a la red, al igual que la robustez con la que ésta cuenta. Por ejemplo, los protocolos de acceso al medio se encuentran implicados dentro de la capa *MAC*, los cuales establecen las reglas de convivencia entre los dispositivos que comparten el mismo medio de transmisión. Otro ejemplo pueden ser los protocolos de encaminamiento, que son los encargados de obtener las mejores rutas mediante el uso de parámetros como el tiempo de transmisión o la tasa de transferencia de datos.

Un problema que puede surgir al implementar una red, es que los protocolos que se escogen para su funcionamiento no se adaptan o no son lo suficientemente robustos ante las condiciones físicas del ambiente, es por esta razón que es de suma importancia simular el comportamiento de las redes antes de desplegarlas, ya que los diseños implementados se pueden someter a diferentes situaciones y así analizar cualquier tipo de problema general de la red, de igual forma se pueden implementar los cambios necesarios para mejorarlo.

Un simulador permite crear o hacer ajustes a los protocolos para mejorar su eficiencia a un costo muy por de bajo de una implementación real. Hoy en día existe una vasta cantidad de simuladores de eventos discretos en el mercado, tales como NS2, NS3, OMNeT++, OPNET, entre otros. Sin embargo, muchos de estos simuladores requieren de una licencia mensual o anual. Otros, aunque son de obtención libre (GNU), su curva de aprendizaje es muy amplia y aunado a esto, en muchos de estos simuladores no se pueden separar las diferentes capas del modelo OSI, por lo que si solo se desea analizar una capa, se requiere interactuar con todas las demás, lo que puede llega a ser contraproducente en algunos casos.

Por ello, en esta tesis se propone un simulador de eventos discretos generado con ayuda de la biblioteca SimPy de Python, cuya curva de aprendizaje es mucho más rápida a comparación de otros lenguajes de programación, además que este lenguaje permite interoperabilidad entre los diferentes sistemas operativos e incluso entre diferentes arquitecturas, por lo que no hay necesidad de compilar o ajustar las bibliotecas creadas en la implementación del simulador. En específico, este trabajo se concentra solo en la capa MAC del protocolo 802.15.4.

# Capítulo 1

# Introducción

Con la evolución de la tecnología, los simuladores se han convertido en un gran apoyo para todas aquellas personas que se dedican a la fabricación, implementación o creación de productos y servicios. Una simulación permite analizar problemas complejos para los cuales una implementación real puede ser muy costosa, también auxilia al proceso de innovación, ya que permite observar y experimentar al sistema con diversos parámetros, además de que pueden suponer un ahorro tanto en tiempo como en dinero. En términos generales y sin tomar en cuenta el fin práctico de un simulador, ya que estos pueden ser utilizados en la industria automotriz, sector aeronáutico, aeroespacial, entre otros muchos más ejemplos, es un software especializado con la capacidad de reproducir el comportamiento tanto del medio en el que se desarrolla cierta actividad o en el que se pone a prueba cierta sustancia, material o equipo [1]. Sin embargo, en ciertos casos, este tipo de herramientas no son tan accesibles para el usuario final, ya que pueden llegar a tener un costo monetario, y aunque otros son gratuitos, su curva de aprendizaje es muy amplia, esto quiere decir que se necesita invertir mucho tiempo para poder aprender a utilizar dichos simuladores. Además de que pueden llegar a presentar ciertas desventajas como resultados imprecisos y en ocasiones soluciones poco óptimas.

Para contrarrestar el problema anterior, se diseñan interfaces amigables con el usuario que reducen la curva de aprendizaje, sin embargo si se desea tener un mayor control del simulador, por ejemplo tener acceso a su código fuente, pueden presentarse inconvenientes como la difícil comprensión del lenguaje, poca capacidad de modificación, déficit en la documentación, entre otros.

Internet de las cosas *(IoT)* ha exponenciado la relevancia de los simuladores, debido a la gran cantidad de nodos que pueden existir dentro de una red de sensores inalámbricos *(WSN)*, ya que la construcción de un laboratorio de pruebas resultaría laborioso y costoso, al igual que realizar experimentos dentro de él. Es por esto que la simulación juega un papel fundamental en el estudio y desarrollo de *WSN* en el ámbito de *IoT*.

En una red de sensores IoT, la materia prima son los datos, los cuales son recolectados, enviados y procesados para que los dispositivos puedan tomar decisiones. Este tipo de redes están conformadas de sistemas inalámbricos distribuidos espacialmente, cuya principal característica es que con un mínimo consumo de energía tienen la capacidad de comunicarse [2].

Existen una gran cantidad de simuladores para *WSN* basados en modelos de simulación de eventos discretos *(DES)*, los cuales modelan el funcionamiento, comportamiento y rendimiento de un sistema de la vida real como una secuencia de eventos en el tiempo. Cada evento sucede en un tiempo determinado y transforma el estado de un sistema, teniendo en cuenta los recursos disponibles, las restricciones y las reglas de interacción [1]. De los simuladores más utilizados en el campo de las telecomunicaciones se encuentra NS2, el cual es *open source* utilizado en la investigación. Este software está desarrollado en el lenguaje C y en Otcl, sin embargo esto hace que la depuración sea compleja ya que se necesita el conocimiento de ambos lenguajes de programación. NS2 se utiliza en simulaciones de redes cableadas e inalámbricas, actualmente NS2 ya no se acepta en revistas indexadas, ya que se dejó de

mantener desde 2009 [3] y debido a esto, muchos investigadores empezaron a migrar a otros simuladores como NS3.

NS3 nace para remplazar y resolver las problemáticas presentes en NS2, por lo que no son compatibles, está escrito en C++ con enlaces opcionales de Python. Sin embargo, la documentación con la que cuenta es buena, pero no suficiente, además de que hay un número limitado de códigos en comparación con NS2.

OMNeT++ está basado en C++, es modular y libre bajo la licencia de GNU, proporciona una interfaz gráfica de usuario muy potente, lo que hace que los procesos de rastreo y depuración sean fáciles, aunque el número de protocolos disponibles no es muy vasto, el código fuente está disponible públicamente.

OPNET está desarrollado en el lenguaje C y Java, y para poder trabajar con él se necesita comprar una licencia muy costosa, contiene una interfaz de usuario en la cual se pueden seleccionar los objetos requeridos y configurarlos desde ésta, siempre y cuando se usen los protocolos disponibles, ya que de lo contrario se vuelve muy difícil la configuración.

NetSim usa Java como lenguaje de programación, para modificar el código fuente se requiere la ayuda del proveedor, es necesaria la compra de una licencia para poder trabajar con este simulador, sin embargo las universidades pueden obtenerlo a precios educativos con descuento, tiene una buena documentación, vídeos y soporte comunitario.

Por todo lo anterior, en esta tesis se propone un simulador de eventos discretos utilizando la biblioteca SimPy de Python, que es un marco de *DES* basado en procesos, los cuales pueden utilizarse para modelar componentes activos como clientes, vehículos o agentes, también cuenta con varios tipos de recursos compartidos. Las simulaciones se pueden realizar en tiempo real, en tiempo del procesador o de forma manual paso a paso. Las ventajas de esta biblioteca son que cuenta con una documentación bien detallada con tutoriales, guías que explican conceptos básicos, ejemplos y referencias a la API. Se utiliza el lenguaje de programación Python debido a que su sintaxis es muy simple y es fácil de leer, lo cual es clave para aquellos que comienzan a programar, de acuerdo con [4], hay 8.2 millones de desarrolladores que trabajan con este lenguaje, cuenta con una inmensidad de bibliotecas libres que facilitan el desarrollo.

Específicamente, en este trabajo de tesis se propone la creación de un simulador que se centra en el protocolo 802.15.4 en la capa de control de acceso al medio en redes inalámbricas de área personal con tasas bajas de transmisión de datos *(LR-WPAN)* empleando la herramienta SimPy de Python.

## 1.1. Definición del problema

Existen una vasta cantidad de simuladores de eventos discretos en el mercado, sin embargo para poder trabajar con ellos, en muchos de los casos, se debe de pagar una licencia mensual o anual, o en otros casos aunque el software es libre, la curva de aprendizaje es muy amplia y aunado a esto, en muchos de los simuladores no se pueden separar las diferentes capas del modelo OSI, lo que es un problema si solo se desea analizar una capa en específico, lo que puede ser contraproducente en algunos escenarios. Por ello, en esta tesis se proponen las bases para un simulador de eventos discretos basado en la biblioteca Simpy de Python de la capa MAC del protocolo 802.15.4.

## 1.2. Hipótesis

El uso de la biblioteca SimPy como base de un DES permite ventajas en la construcción de un simulador orientado a WSN como flexibilidad, repetibilidad, reusabilidad con una curva de aprendizaje menor a la de otros lenguajes de programación.

## 1.3. Metas

### 1.3.1. Meta general

Realizar un simulador de eventos discretos utilizando la biblioteca SimPy de Python, específicamente la capa MAC del protocolo 802.15.4.

### 1.3.2. Metas particulares

- Leer la documentación de la biblioteca SimPy para poder utilizarla y aprovecharla correctamente.
- Comprender el funcionamiento del estándar 802.15.4, haciendo énfasis en la capa MAC.
- Construir un simulador que tenga como motor a la biblioteca SimPy para simular el comportamiento de la capa MAC del protocolo 802.15.4.
- Probar diferentes escenarios para obtener resultados.

## 1.4. Metodología

El proceso para la creación del simulador se puede separar en dos etapas.

### 1.4.1. Primera etapa

En esta etapa se lleva a cabo la investigación y documentación relacionada con la biblioteca SimPy. Además de la investigación acerca del funcionamiento del protocolo 802.15.4, por ejemplo el tamaño de las tramas, constantes del protocolo, etcétera.

### 1.4.2. Segunda etapa

En esta etapa se busca filtrar la información recabada en la etapa anterior con el fin de utilizar la biblioteca SimPy para implementar el estándar 802.15.4 a través de un programa realizado en Python. Finalmente, se comprobará su funcionamiento a través de pruebas en diferentes escenarios que se aproximen a la realidad.

## 1.5. Contribución

Se espera que este estudio sirva como base para el desarrollo de otras tesis en las que se implementen las capas restantes del protocolo 802.15.4, y que en un futuro este simulador se haga público para uso académico.

## 1.6. Descripción del contenido

Este trabajo de tesis se presenta de la siguiente forma:

- El capítulo 2, explica y describe lo que es un simulador de eventos discretos, además de presentar un análisis y descripción de los simuladores más populares dentro del mundo de las redes, aunado a esto, también se mencionan algunos experimentos realizados para comparar el rendimiento y funcionamiento de estos.
- El capítulo 3, presenta una descripción general acerca del funcionamiento del paquete SimPy de Python, además de una breve introducción sobre ZigBee y el estándar IEEE 802.15.4, haciendo énfasis en la capa *MAC* de dicho estándar.

- El capítulo 4, describe cada una de las clases que conforman al simulador diseñado, el cual es el objetivo de esta tesis.
- El capítulo 5, presenta los experimentos propuestos y los resultados obtenidos, además de la explicación detallada de cada uno de estos.
- El capítulo 6, está conformado por las conclusiones, la verificación de la hipótesis y trabajo futuro.

# Capítulo 2

# Antecedentes

Un simulador de eventos discretos (DES) [1] se describe como un modelo computacional que evoluciona en el tiempo mediante cambios en las variables de estado. La mayoría de los simuladores de redes están basados en el modelo de eventos discretos. Actualmente, la simulación de redes es una de las metodologías más útiles y frecuentes para evaluar diferentes aspectos sin la necesidad de implementaciones físicas, ya que suele ahorrar tiempo, dinero y facilita el estudio de las características más relevantes de la red. Existe una gran variedad de simuladores en el mundo de las telecomunicaciones, por lo que elegir uno en específico es una tarea compleja, ya que algunos pueden estar especializados en redes inalámbricas, otros en redes cableadas. También hay que tomar en consideración las variaciones en los sistemas operativos, los requisitos de hardware y software, las características de los resultados que se obtienen, la escalabilidad y el costo.

Comúnmente, los nuevos protocolos antes de montarse a gran escala se prueban con herramientas de simulación o modelos analíticos. Si los resultados son prometedores, se comienza con la implementación en el mundo real, de lo contrario se modifican y mejoran. En [3] se menciona que el modelado analítico cuenta con desventajas, ya que los resultados deducidos no son precisos en términos de consumo de energía, memoria y procesamiento, además de que con la evolución de las tecnologías, las redes se han vuelto cada vez más complejas.

## 2.1. Simuladores de red

A continuación se describen los simuladores de red más populares en la comunidad científica.

*Network Simulator Version 2* (NS-2) [5] es un simulador de red de eventos discretos de código abierto, se utiliza para simular protocolos de red con diferentes topologías, además de que se pueden simular redes cableadas e inalámbricas. Está construido en el lenguaje C, donde se define el mecanismo interno de los objetos de simulación, y en OTcl donde los usuarios pueden controlar el escenario de simulación y los eventos. NS-2 está registrado bajo la licencia GNU [6, 7] y es compatible con GNU / Linux, FreeBSD, Mac OS X y Windows. Actualmente se comentan en foros de investigación como *ResearchGate* que ya no se aceptan trabajos simulados con esta herramienta. En [6] se menciona que soporta una gran cantidad de modelos, tiene una simulación potente y flexible y una gran comunidad activa de usuarios. Sin embargo, necesita ser recompilado cada que se modifique el código de usuario. Más aún, cuenta con una estructura de código compleja y es difícil de analizar. Además de que si se desea realizar una simulación de gran escala, el tiempo de simulación puede crecer demasiado.

NS-3 [8] nace con el propósito de reemplazar a su antecesor NS-2, ya que también es un simulador de red de eventos discretos bajo la licencia GNU, éste se crea para ser implementado tanto en el ámbito educativo como en la investigación. NS-3 está escrito en lenguaje C++, pero tiene recursos opcionales hechos en Python. Es compatible con GNU / Linux, FreeBSD, Mac OS X y Windows. Algunas de sus ventajas mostradas en [6] son una alta modularidad en comparación a NS-2, es mucho más flexible que cualquier otro simulador, tiene una amplia

gama de uso. Sin embargo, tiene desventajas como documentación limitada, y una comunidad pobre de usuarios.

OMNeT++ [9] es un simulador de eventos discretos de código abierto que está diseñado en C++ y ofrece un IDE basado en Eclipse, es modular, extensible y basado en componentes. Este simulador [10] es una herramienta principalmente utilizada para construir simuladores de redes de comunicaciones, pese a que su propósito no sea este, ya que OMNeT++ tiene la intención de ser una plataforma de simulación en la que se pueda construir cualquier marco de simulación, por ejemplo simulaciones de sistemas complejos, redes o arquitectura de hardware. En [6, 11] se le atribuyen ventajas como la reutilización de módulos de forma libre, y un amplio soporte para GUI. Es compatible con Windows, Linux y Mac OS, sin embargo se menciona que el soporte para simulaciones tanto inalámbricas como cableadas es pobre y no ofrece una gran variedad de protocolos.

QualNet (Quality Networking) [12] es un software utilizado para la evaluación, planificación y capacitación de redes. Permite simular la conducta de redes de comunicaciones complejas a gran escala. Es compatible con UNIX, Windows (Windows 7 Home Premium Professional de 32-bits y 64-bits, Windows 8 y Windows 8 Pro 32-bits), MAC y Linux (CentOS -5.9, Red Hat Enterprise Linux 5.9, Ubuntu-12.04 LTS) [6], puede simular combinaciones de redes cableadas e inalámbricas, tiene una GUI fácil de usar, tiene capacidades sofisticadas de animación, así como una buena escalabilidad, ya que soporta alrededor $20000$ nodos. También puede correr en sistemas de clúster, multinúcleo y multiprocesador, pero tiene desventajas como que QualNet solo es una extensión comercial de GloMoSim, su instalación en Linux es compleja y la interfaz de usuario hecha en Java es lenta. En [10] se explica que las principales diferencias entre QualNet y GloMoSim son que el primero está diseñado en C++ y es mantenido por SNT, mientras que el segundo está escrito en C y es mantenido por *Parallel Computing*.

En [6] NetSim se describe como una herramienta de red estocástica de simulación de eventos discretos, la cual se utiliza para la investigación y experimentación de protocolos para redes. Este simulador cuenta con un entorno de desarrollo integrado, que funciona como interfaz entre el código del usuario, las bibliotecas de NetSim y el núcleo de simulación. Es descrito como un simulador muy efectivo, está desarrollado en C y Java, es compatible con Windows. Se le atribuyen ventajas como una GUI altamente interactiva con el usuario. El modelado con NetSim es simple y fácil de usar, cuenta con un animador por el cual se pueden analizar los paquetes de datos y de control, es fácil de aprender, pero tiene la desventaja de no ser gratuito y de estar solo disponible para Windows.

OPNET o, como se le renombró por parte de la empresa que lo compró, Riverbed Modeler [13] es un simulador de eventos discretos que cuenta con protocolos y tecnologías para diseñar, modelar y analizar redes, está hecho en C y C++. En [6] se menciona que los resultados de configuración y simulación de topologías son presentados de manera muy intuitiva y visual, su interfaz gráfica es muy amigable con el usuario, ya que desde esta se pueden modificar los parámetros de simulación. Es muy útil en la implementación de redes complejas con una gran cantidad de dispositivos, siempre y cuando se encuentren en el rango de $210$ a $290$ nodos por topología. OPNET cuenta con muchas herramientas que les permiten a sus usuarios realizar un análisis fácil. Es compatible con Windows, Linux y plataformas Solaris, de sus ventajas más destacables es que cuenta con un modelado inalámbrico personalizable, tiene un potente motor de simulación de eventos discretos. Sin embargo, la operación de la GUI es compleja, no permite simulaciones con una gran cantidad de nodos conectados, la precisión de los resultados queda limitada debido a la resolución del muestreo, la simulación llega a ser ineficiente si hay muchos periodos de inactividad y por último, este simulador tiene un alto costo de licencia.

Algunos otros simuladores como GloMoSiM, TOSSIM, J-SIM, NCTUns, DRMSim, SSFNet, GrooveNet y TraNS también están descritos y documentados en [6], sin embargo no se hace énfasis en esta tesis acerca de ellos debido a su poca popularidad.

La tabla 2.1 presenta las características más relevantes de los simuladores previamente mencionados.

| Nombre | Uso académico | Tipo de licencia | Curva de aprendizaje | Lenguajes | Plataformas que soportan | Documentación | Escalabilidad |
|---|---|---|---|---|---|---|---|
| NS - 2 | Alto | Libre | Alta | C++ y OTCL | GNU/Linux, FreeBSD, Mac OS X y Windows | Excelente | Limitada |
| NS - 3 | Medio | Libre | Alta | C++ y Python | GNU/Linux, FreeBSD, Mac OS X y Windows | Buena | Limitada |
| OMNeT++ | Alto | Libre | Media | C++ | GNU/Linux, Mac OS X y Windows | Buena | Suficiente |
| QualNet | Alto | Comercial | Media | C++ | UNIX, Windows, Mac y Linux | Excelente | Grande |
| NetSim | Alto | Comercial | Baja | C y Java | Windows | Excelente | Grande |
| OPNET | Alto | Comercial | Media | C y C++ | GNU/Linux, FreeBSD, Mac OS X y Windows | Buena | Grande |
| SimPy | Medio | Libre | Baja | Python | Cualquier sistema operativo que soporte Python | Excelente | Limitada |

Tabla 2.1: Información general de cada simulador.

Por otro lado, en [3] se realiza la simulación del protocolo *ad hoc on demand distance vector (AODV)* para comparar el rendimiento de los simuladores NS-2, NS-3, OMNET++ y GloMoSiM, ya que este se encuentra disponible en todos ellos. Para llevar a cabo la prueba primero se estableció la conexión entre ciertos nodos predeterminados. En la simulación cada nodo contaba con un tiempo de transmisión de aproximadamente $0.2$ segundos, el número de nodos incrementaba de $400$ a $2000$, el tiempo de ejecución se estableció en $500$ segundos y el área de simulación era de $1000$*$1000$ metros. Los resultados muestran que NS-2 utiliza aproximadamente $50\,\mathrm{MB}$ al final de la simulación, mientras que NS-3 usa $35\,\mathrm{MB}$, que fue el simulador que menor cantidad de memoria usó. Conforme al aumento en el número de nodos, los simuladores mostraban un crecimiento lineal en el consumo de memoria. Con respecto al uso del CPU, se muestra que tanto NS-2 como NS-3 hacen un uso muy similar y es mucho mayor en comparación con GloMoSiM y OMNET ++, ya que estos dos utilizaron un consumo de hasta 35 % con una pequeña diferencia entre ellos, mientras que NS-2 y NS-3 utilizaron casi el 100 %. Conforme al tiempo de procesamiento, NS-2 obtuvo el mayor tiempo, además de que éste aumenta rápidamente al incrementar el número de nodos, lo que demuestra que NS-2 no es escalable. En términos de tiempo de cálculo y escalabilidad, NS-3 es el mejor de ellos.

En [14] se compara el rendimiento de cinco simuladores de código abierto, los cuales son NS-2, NS-3, OMNeT ++, JiST y SimPy, el modelo de simulación comparativo no existe en ninguno de los simuladores, por lo que se tuvo que realizar desde cero, se trata de una red básica con topología en forma de cuadrado, en ésta el nodo emisor ubicado en una esquina de la red genera un paquete cada segundo y lo transmite a sus vecinos, estos a su vez, después de un segundo de retraso comienzan a inundar la red con mensajes, todos los enlaces que conforman a la red tienen la misma probabilidad de éxito. El receptor se encuentra ubicado en la esquina opuesta al transmisor. Para asegurar que los resultados en todos los simuladores se obtuvieran bajo las mismas condiciones, se realizó una simulación en la que la probabilidad de éxito variaría entre $0$ y $1$, la cantidad de nodos varía desde $4$ hasta $3025$ nodos y el tiempo de simulación fue de $600$ segundos. En cuanto a los resultados, se obtuvo que conforme a la pérdida de paquetes, todos los simuladores obtuvieron una curva muy similar, excepto SimPy que produjo tasas de perdida un poco mayor a los demás, pero tolerables. Respecto al tiempo de ejecución tomando en consideración el tamaño de la red, se obtuvo que SimPy no es escalable, ya que conforme crece la red le toma más tiempo realizar la simulación, esto se observa con $3025$ nodos en donde se tardó $1225$ segundos en ejecutar la simulación, casi $14$ veces más tiempo de lo que le tomo a JiST. Por otro lado, se observó que si se disminuye la probabilidad de éxito, el tiempo de simulación disminuye. Esto ocurre por igual en todos los simuladores, sin embargo cuando la probabilidad es mayor a $0.5$, se observa que el tiempo de ejecución

de SymPy aumenta mucho más rápido conforme crece la probabilidad, a comparación de los otros simuladores. En cuanto al uso de memoria tomando en cuenta el tamaño de la red, se obtuvo que JiST es el que ocupa mayor espacio a comparación de los otros simuladores. Específicamente, en el caso de $3025$ nodos, JiST ocupa aproximadamente $140\,\mathrm{MB}$, que es más del doble de lo que ocupa NS-3.

En conclusión de [14] se obtuvo que NS-3, OMNeT ++ y JiST son capaces de realizar simulaciones de red a gran escala de manera eficiente, JiST es el que se ejecuta en menor tiempo, pero el que más recursos consume, así que el simulador con el mejor desempeño obtenido generalmente en los experimentos es NS-3, pero la desventaja es que este simulador no cuenta con la misma colección de modelos que su antecesor.

## 2.2. Estado del arte en la implementación de un simulador del estándar IEEE 802.15.4.

Hasta ahora, este trabajo de tesis es el primer trabajo en el que se propone realizar un simulador de la capa MAC del estándar IEEE 802.15.4 con ayuda del paquete SimPy, sin embargo hay otros trabajos realizados con otras herramientas en los que se han desarrollado simuladores para dicho estándar, a continuación se describen algunos de ellos.

En [15], se utiliza como herramienta el *software MATLAB* para desarrollar un simulador basado en el protocolo IEEE 802.15.4. En este trabajo el desempeño del simulador se compara con modelos analíticos, el producto final es conformado por una GUI para el usuario y una herramienta gráfica para visualizar los eventos de las capas física y MAC.

En [16], por ejemplo, se implementa un pequeño subconjunto del protocolo IEEE 802.15.4 para poder lograr una comunicación punto a punto. Esta implementación utiliza el lenguaje C, se destaca que para que el código se pueda portar a cualquier plataforma, se necesitan hacer pequeños cambios en él.

En [17], los autores también implementa un pequeño subconjunto del protocolo IEEE 802.15.4 para poder lograr una comunicación punto a punto, presentan un método para comparar el consumo actual de los sistemas embebidos de transferencia inalámbrica de datos. Para lograr lo anterior, se utiliza el lenguaje Verilog.

# Capítulo 3

# Marco teórico

En este capítulo se explica de forma detallada el funcionamiento de la biblioteca SimPy de Python y del protocolo 802.15.4, haciendo énfasis en la capa MAC.

## 3.1. SimPy

Fue desarrollado originalmente por Klaus G. Müller y Tony Vignaux en el año 2002 y en 2008 Ontje Lünsdorf's y Stefan Scherfke se hicieron cargo del proyecto [18]. Está basado en Simula y Simscript, pero utilizando Python estándar.

SimPy es un paquete especializado para la simulación de eventos discretos orientado a procesos. SimPy genera eventos que se ordenan por prioridad, tiempo de simulación o por un identificador de evento. Cuando un evento se desencadena, éste pude generar una devolución de llamada y/o retornar algún valor. Esta biblioteca procesa eventos secuencialmente, si existen dos eventos al mismo tiempo, el que se registra primero es el que se procesa primero *(FIFO)*. Es muy importante destacar que pese a lo anterior, los procesos se pueden ejecutar en paralelo, ya que la discretización de la escala del tiempo puede hacer que los eventos parezcan realizarse en el mismo momento. Los procesos en SimPy son generados por funciones generadoras de Python y estas pueden utilizarse para crear componentes activos, además los procesos se encuentran en un entorno donde pueden interactuar entre ellos o con el medio a través de eventos. Los eventos atraviesan por las siguientes etapas: *no activado*, *activado* y *procesado*.

SimPy cuenta con tres tipos de recursos compartidos. Los primeros son llamados *resources*, estos simulan puntos de congestión de capacidad limitada como servidores, túneles, entre otros que pueden ser utilizados por un número limitado de procesos a la vez. Los *containers* que modelan la producción y el consumo de algún fenómeno como el consumo de agua o un estante de mercancía de un almacén. Y finalmente los *shores* que producen y permiten el consumo de objetos. A los recursos se les puede atribuir la capacidad que el usuario deseé, también se les puede colocar cierto nivel de prioridad y realizar simulaciones más reales con esta cualidad.

Se puede controlar el tiempo de simulación dependiendo de las necesidades del usuario, ya que se puede configurar en tiempo real o de forma manual paso a paso a través de los eventos. SimPy puede realizar una simulación hasta que se quede sin eventos que ejecutar, hasta que se alcanza cierto tiempo predeterminado o hasta que se de cierta condición. Una de sus principales ventajas es que es muy fácil de instalar. [19]

A continuación se describen algunas de las funciones que se encuentran dentro de la biblioteca y que se utilizan dentro de este trabajo.

- Environment(), se utiliza para generar el espacio en donde se va a dar la simulación, almacena los eventos en una lista y realiza un seguimiento del tiempo de simulación actual.
- Environment.timeout(), se utiliza para incrementar el tiempo de simulación, preferentemente cuando se ejecuta un evento.

- Environment.now(), permite obtener el tiempo de simulación actual.
- Environment.process(), crea una instancia de proceso que se puede utilizar en interacciones entre procesos como esperar a que termine un proceso o interrupciones.
- interrupt(), interrumpe un proceso que se encuentra en ejecución.
- request(), es un método que genera un evento de petición para poder utilizar el recurso o también espera hasta que este se encuentre disponible.
- release(), sirve para liberar el recurso una vez que se haya terminado de ocupar.
- until, se debe de igualar a un valor de tiempo y se utiliza como argumento para Environment.run(), dicho periodo de tiempo es lo que durará la simulación.
- Environment.run(), da inicio a la simulación y si se le pasa algún argumento, detiene la simulación hasta que se cumpla con la condición especificada en este.
- Resource(*env*, capacity = *n*) crea un recurso que se utilizará en el ambiente *env*, con una capacidad de *n*.

## 3.2. ZigBee

ZigBee es un estándar que fue desarrollado por *ZigBee Alliance*, el cual define protocolos de comunicación para redes inalámbricas de corto alcance y baja velocidad de datos, se implementa principalmente en dispositivos alimentados por baterías, que además cumplen con una baja tasa de datos, un bajo costo y una larga duración en la vida de la batería [2]. Opera en las bandas de frecuencia de $868\,\mathrm{MHz}$, $915\,\mathrm{MHz}$ y $2.4\,\mathrm{GHz}$. La tasa máxima de velocidad de datos es de $250\,\mathrm{kbps}$.

Muchas de las aplicaciones de este protocolo están orientas a permanecer la mayor parte del tiempo en modo de ahorro de energía, por lo que el tiempo que dedican a cualquier otra actividad suele ser muy pequeño. Esto da como resultado que los dispositivos funcionen durante mucho tiempo antes de que se les agote su batería. En otras palabras, su ciclo de trabajo suele ser muy pequeño. ZigBee está bajo el estándar IEEE 802.15.4, donde se define sus protocolos de capa física (*PHY*) y de capa de control de acceso al medio (*MAC*) [20].

### 3.2.1. Relación entre ZigBee e IEEE 802.15.4

Para establecer una red de comunicaciones, por lo general, los diseñadores se basan en un modelo de capas, en el que cada capa tiene definida su función dentro de la red, por lo general cada capa se comunica con las capas que están directamente por encima y por debajo de ella.

Las capas que conforman al protocolo para redes inalámbricas ZigBee se muestra en la figura 3.1, las cuales están basadas en el modelo *Open System Interconnection (OSI)*, el cual consta de siete niveles. Las capas reciben datos de una capa superior y los pasan a una capa inferior, así hasta llegar al enlace físico por el cual se transfieren los datos a otro dispositivo en el que se repite el proceso en sentido inverso [21].

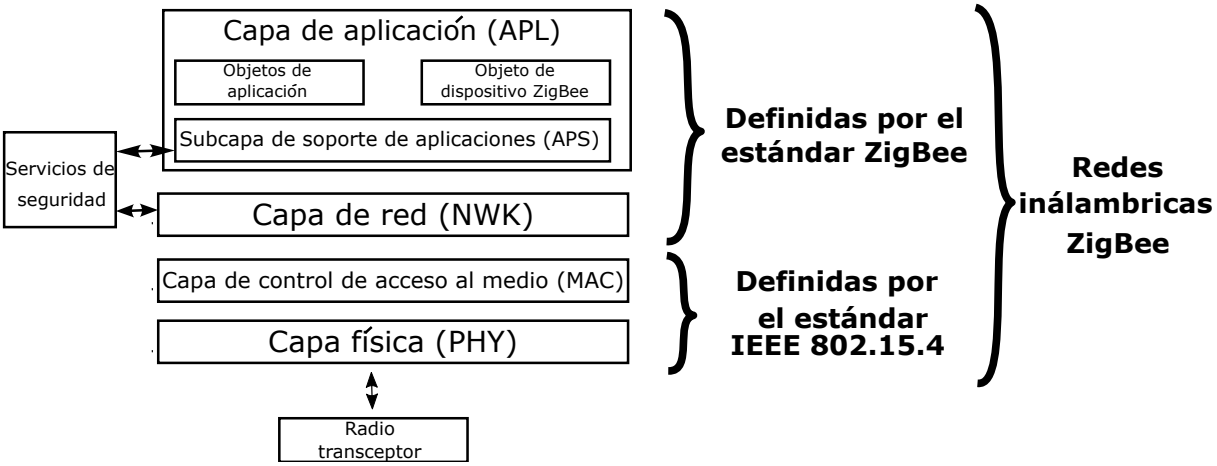



Figura 3.1: Capas del protocolo ZigBee.

Como se muestra en la figura 3.1, ZigBee define las especificaciones para la capa de red y la capa de aplicación, y toma la definición de la capa física y de la capa *MAC* establecidas en el estándar 802.15.4 para dar paso al protocolo para redes inalámbricas ZigBee. Cabe especificar que el estándar IEEE 802.15.4 es totalmente independiente de ZigBee.

## 3.3. IEEE 802.15.4

El estándar 802.15.4 fue lanzado en el año 2003 por el comité de estándares IEEE 802. En este comité se definen las especificaciones para las capas *PHY* y *MAC* para redes inalámbricas. Este estándar se encarga de establecer una red de comunicaciones confiable mediante enlaces de radiofrecuencia, permitiendo el uso de *broadcasts* para direccionar varios dispositivos a la vez. La confiabilidad que proporciona se da a través de la detección de errores, los mensajes de recepción exitosa y las retransmisiones. Las ventajas que presenta este estándar a comparación de otros es la simpleza de la implementación, su flexibilidad de red, bajos costos, su reducido consumo de energía y su baja tasa de transmisión de datos [22].

ZigBee está diseñado para trabajar en las bandas de frecuencia de $868\,\text{MHz}$ ($868 - 868.6\,\text{MHz}$), $915\,\text{MHz}$ ($902 - 928\,\text{MHz}$) y $2.4\,\text{GHz}$ ($2400 - 2483.5\,\text{MHz}$). En la tabla 3.1 se pueden observar los detalles sobre las formas en que se usan dichas bandas de frecuencia. Las bandas de $868\,\text{MHz}$ y $915\,\text{MHz}$ se agrupan en $868/915\,\text{MHz}$, debido a que si un transceptor admite una, también debe admitir la otra [23].

La banda más popular en la comunidad es la banda de $2.4\,\text{GHz}$ porque tiene la velocidad de datos más rápida y mayor cantidad de canales, pero el estándar IEEE $802.11$ también trabaja en esa banda, por lo que la coexistencia de ambos puede ser un problema, además de que las señales de altas frecuencias se les dificulta penetrar muros y otros objetos. Sin embargo, las técnicas de *direct sequence spread spectrum (DSSS) o parallel sequence spread spectrum (PSSS)* ayudan a mejorar el rendimiento de los receptores en un ambiente con multitrayectos [2].

| Frecuencia. [MHz] | Número de canales | Modulación | Chip Rate (kchip/s) | Bit Rate (kbps) | Symbol Rate (ksymbol/s) | Método de esparcimiento. |
|---|---|---|---|---|---|---|
| 868 - 868.6 | 1 | BPSK | 300 | 20 | 20 | Binary DSSS |
| 902 - 928 | 10 | BPSK | 600 | 40 | 40 | Binary DSSS |
| 868 - 868.6 (opcional) | 1 | ASK | 400 | 250 | 12.5 | 20 - bit PSSS |
| 902 - 928 (opcional) | 10 | ASK | 1600 | 250 | 50 | 5 - bit PSSS |
| 868 - 868.6 (opcional) | 1 | O-QPSK | 400 | 100 | 25 | 16 - array orthogonal |
| 902 - 928 (opcional) | 10 | O-QPSK | 1000 | 250 | 62.5 | 16 - array orthogonal |
| 2400 - 2483 .5 | 16 | O-QPSK | 2000 | 250 | 62.5 | 16 - array orthogonal |

Tabla 3.1: Tasa de datos y frecuencias de operación del protocolo IEEE 802.15.4.

Los dispositivos en este protocolo se clasifican de dos formas: *full-function devices (FFDs) y reduced-function devices (RFDs)*. Los primeros son capaces de realizar cualquier tarea descrita en el estándar y pueden tomar cualquier rol que se les asigne en la red, como coordinador, coordinador PAN o *device*, mientras que el segundo cuenta con capacidades limitadas. Por ejemplo, un dispositivo *FFD* puede entablar comunicación con cualquier dispositivo dentro de la red, mientras que un dispositivo *RFD* solo puede hablar con un dispositivo *FFD* [2].

Un coordinador es un dispositivo que es capaz de transmitir mensajes, mientras un coordinador PAN es el controlador principal de la red de área personal. Finalmente, un dispositivo (device) no puede ser ninguno de los dos anteriores.

El estándar 802.15.4 especifica solo dos topologías posibles, las figuras 3.2 y 3.3 muestran estas topologías. La topología estrella consta de un conjunto de dispositivos donde cualquiera de estos puede comunicarse exclusivamente con el coordinador PAN. Por otro lado, la topología punto a punto, en la cual cada dispositivo puede comunicarse con cualquier otro dispositivo cercano a él (cualquier FDD puede ser un coordinador PAN).

El estándar especifica que una red siempre debe de ser creada por un coordinador PAN, el cual debe de ser único dentro de la red, además de que éste tiene que asignar una dirección única a cada dispositivo de la red, además debe iniciar, finalizar y encaminar los mensajes a través de la red [24]. Es importante destacar que en una red punto a punto se pueden formar topologías tipo *mesh* o tipo árbol.

En [25] se explica que la topología punto a punto también cuenta con un coordinador PAN, pero es diferente a la topología tipo estrella en que cualquier dispositivo es capaz de comunicarse con cualquier otro dispositivo siempre que estén dentro del alcance del otro, esta característica permite implementar formaciones de red más complejas. Algunas aplicaciones como el control y monitorización industrial, redes inalámbricas de sensores, seguimiento de activos e inventarios, agricultura inteligente y seguridad se ven beneficiadas de dicha topología de red.

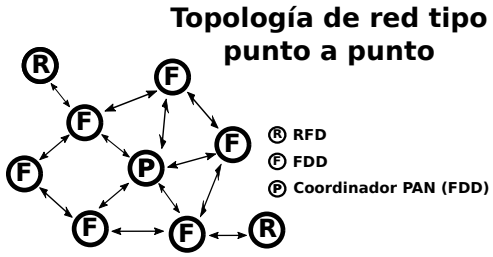



Figura 3.2: Topología de red punto a punto que permite el estándar IEEE 802.15.4.

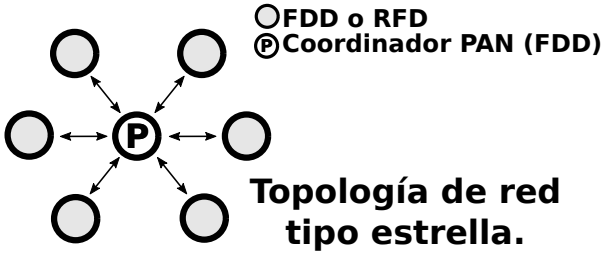



Figura 3.3: Topología de red tipo estrella que permite el estándar IEEE 802.15.4.

IEEE 802.15.4 utiliza un método simple para permitir que múltiples dispositivos usen el mismo canal de frecuencia para comunicarse, este método es conocido como *Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA)*. Cuando un dispositivo quiere transmitir, éste tiene que verificar que el canal este libre mediante el método *clear channel assessment (CCA)* [26], que basa su decisión en la cantidad de energía espectral que se presenta en el canal (*energy detection (ED)*) o detectando el tipo de señal que ocupa el canal (*carrier sense (CS)*) y con base a eso toma una decisión. Si el canal no se encuentra disponible, el dispositivo se espera un tiempo aleatorio y vuelve a tratar de transmitir, esto se repite hasta que el canal se encuentre libre o hasta que el dispositivo alcance su número máximo de reintentos de retransmisión.

Existe otra forma de acceder al canal denominada *contention-free method* [24], en esta metodología el coordinador PAN le asigna un slot de tiempo, denominado *guaranteed time slot (GTS)*, a un dispositivo que desee transmitir sin tener que utilizar *CSMA-CA*. Para utilizar este mecanismo es necesario que toda la red se encuentre sincronizada. Para cumplir con esto se utiliza un mensaje denominado *Beacon* con el fin de sincronizar los relojes de todos los dispositivos dentro de la red, dicho mensaje solo puede ser enviado por el coordinador PAN. Este proceso es llamado *beacon-enabled PAN*, también es opcional utilizar la trama *superframe*, en la cual pueden existir tres tipos de periodos. El primero de ellos se denomina *contention access period* (CAP), donde todos los dispositivos que quieran transmitir deben utilizar *slotted CSMA-CA* para obtener acceso al canal. El segundo período se denomina *contention-free period* (CFP), el cual es utilizado para garantizar un intervalo de tiempo para un dispositivo en específico. Finalmente, *inactive period*, el cual permite que un dispositivo ingrese al modo de ahorro de energía.

IEEE 802.15.4 define tres tipos de comunicación, las cuales son coordinador a *device*, *device* a coordinador y entre *devices*. Los tres tipos se pueden dar en una topología punto a punto, pero solo los dos primeros se dan en una topología tipo estrella.

Para verificar que los datos que se transmiten se reciben correctamente, este protocolo utiliza una secuencia de verificación de trama de 16 bits (FCS) basada en la verificación de redundancia cíclica (CRC) de la Unión Internacional de Telecomunicaciones (ITU).

Cuando un dispositivo quiere unirse a una red, debe de enviar una solicitud de asociación al coordinador, este puede aceptar o rechazar dicha solicitud, pero si el dispositivo tiene la intención de abandonar la red, debe de solicitarlo por medio de la disociación para notificar al coordinador.

### 3.3.1. Capa PHY del 802.15.4

Esta capa proporciona dos servicios: el servicio de datos PHY y el servicio de administración PHY. El servicio de datos PHY permite la transmisión y recepción de tramas a través del canal de radio. Mientras que el servicio de administración tiene la responsabilidad de activación y desactivación del transceptor, *energy detection (ED)*, *link quality indicator(LQI)*, selección de canal y *clear channel assessment* (CCA).

La forma de comunicarse entre dispositivos es a través de tramas, cuya estructura se muestra en la figura 3.4, básicamente está compuesto de tres partes: *Synchronization header (SHR), PHY header (PHR), y PHY payload*. El primero permite que el receptor se sincronice y bloquee el flujo de bits. El segundo contiene el tamaño de la trama y el último contiene la carga útil, que se conforma de la información de capas superiores [23].

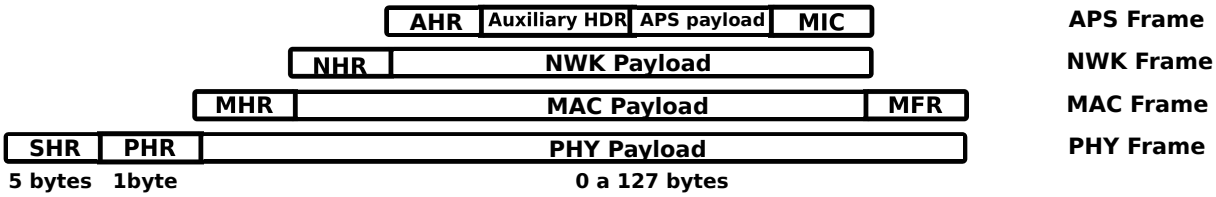



Figura 3.4: Desglosamiento de la trama PHY.

### 3.3.2. Capa MAC

La subcapa MAC proporciona dos servicios: el servicio de datos MAC y el servicio de gestión MAC que interactúa con el punto de acceso de servicio (SAP) de la entidad de gestión de subcapa MAC (MLME) (conocido como MLME-SAP). El servicio de datos MAC permite la transmisión y recepción de unidades de datos MAC (MPDU) a través del servicio de datos PHY. Las características de la subcapa MAC son gestión de acceso al canal, gestión de GTS, validación de trama, entrega de trama reconocida, asociación y disociación.

Esta capa proporciona una interfaz entre la capa física y la capa de red, es responsable de generar *beacons* y sincronizar los dispositivos mediante *beacon-enabled network*. [27].

Debido a la naturaleza de las comunicaciones por radio, un dispositivo receptor puede recibir y decodificar transmisiones de todos los dispositivos que cumplan con el estándar IEEE 802.15.4 que operen en el mismo canal y estén dentro de su rango de comunicaciones, junto con la interferencia de otras fuentes. Por lo tanto, la subcapa MAC podrá filtrar las tramas entrantes y presentar solo las tramas que son de interés para las capas superiores.

#### 3.3.2.1. Estructura de las tramas MAC

Las estructuras de trama son diseñadas para mantener la complejidad al mínimo y al mismo tiempo hacerlas lo suficientemente robustas para la transmisión en un canal ruidoso [27]. Las tramas que se generan en esta capa [17] están conformadas por 3 partes: el *MAC header (MHR)*, que contiene información acerca de direccionamiento y seguridad, el *MAC payload* cuyo tamaño es variable y está conformado por comandos o datos, y el *MAC footer (MFR)* que transporta información de correción de errores *frame Check Sequence (FCS)*. El detalle de cada una de estas partes se puede observar en la figura 3.5. En algunos documentos se menciona el campo *Auxiliary Security HDR*, que se ubica dentro del MHR, pero este es opcional.

| octetos: 2 | 1 | 0/2 | 0/2/8 | 0/2 | 0/2/8 | 0/5/6/10/14 | Variable | 2 |
|---|---|---|---|---|---|---|---|---|
| Frame control | Secuence number | Destination PAN identifier | Destination address | Source PAN identifier | Source Address | Auxiliary security header | Frame payload | FCS |
| | | Addressing fields | | | | | | |
| MHR | | | | | | | MAC Payload | MFR |

Figura 3.5: Desglose de la trama MAC.

En el estándar IEEE 802.15.4 se definen cuatro tipos de trama MAC, cabe especificar que cada una tiene un propósito en específico, las cuales se describen a continuación.

- Trama de datos.
  También denominadas tramas de información [20], son las que se encargan de transportar datos que son proporcionados por la capa de red, en la figura 3.6 se observa la estructura de una trama de este tipo.

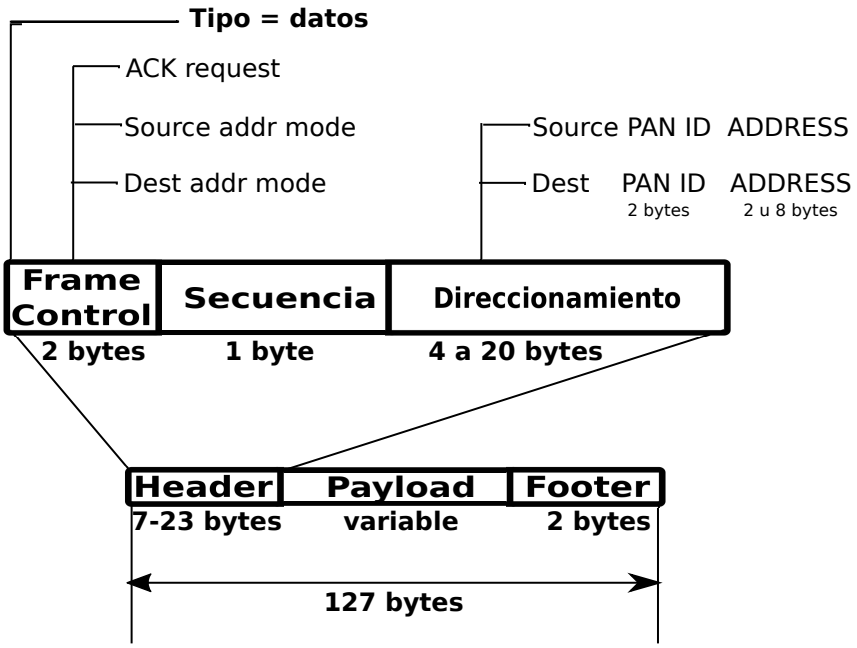



Figura 3.6: Estructura de la trama de datos MAC.

- Trama ACK.
  Este tipo de tramas son pequeñas [21], debido a que no contienen información de direccionamiento ni carga útil, se utiliza para confirmar la recepción exitosa de un paquete, siempre y cuando haya sido solicitada por el nodo del quien se recibió el paquete y solo se puede mandar como respuesta a tramas de datos y de comandos, en la figura 3.7 se observa la estructura de una trama de este tipo.

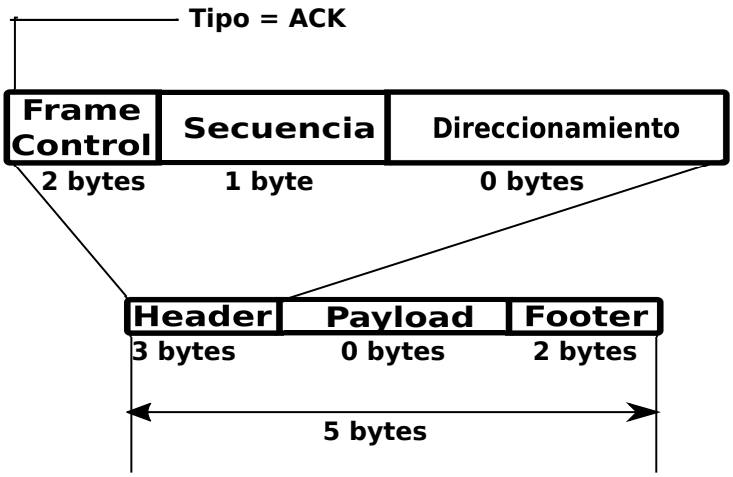



Figura 3.7: Estructura de la trama ACK MAC.

- Trama de comandos.
  Esta trama permite realizar funciones administrativas en la red [20], por ejemplo (ver figura 3.8):

  1. *Association request*: mensaje que manda un dispositivo para unirse a una red.
  2. *Association response*: mensaje que manda el coordinador para aceptar o denegar una asociación.
  3. *Disassociation notification*: mensaje que se genera para informar que un dispositivo sale de la red.
  4. *Data request*: este mensaje lo utiliza un dispositivo para solicitarle al coordinador que le entregue los datos que tenga pendientes para él.
  5. *PAN ID conflict notification*: mensaje que se genera para informar al coordinador que su ID PAN es repetido.
  6. *Orphan notification*: mensaje que se genera para informar que un dispositivo ha perdido comunicación con su coordinador.
  7. *Coordinator realignment*: mensaje que se genera por el coordinador de la red para restablecer la comunicación con un dispositivo huérfano.
  8. *Beacon request*: mensaje que genera un dispositivo para identificar a los coordinadores en su área de recepción.
  9. *GTS request*: mensaje que genera un dispositivo para solicitar una cantidad de *time slots* para su uso exclusivo.

  Un dispositivo *FFD* es capaz de transmitir o recibir cualquier tipo de comando, mientras que un dispositivo RFD solo puede transmitir los comandos *association request, disassociation notification, data request, PAN ID conflict notification y Orphan notification*, y solo es capaz de recibir los comandos *association response, disassociation notification y coordinator realignment*.

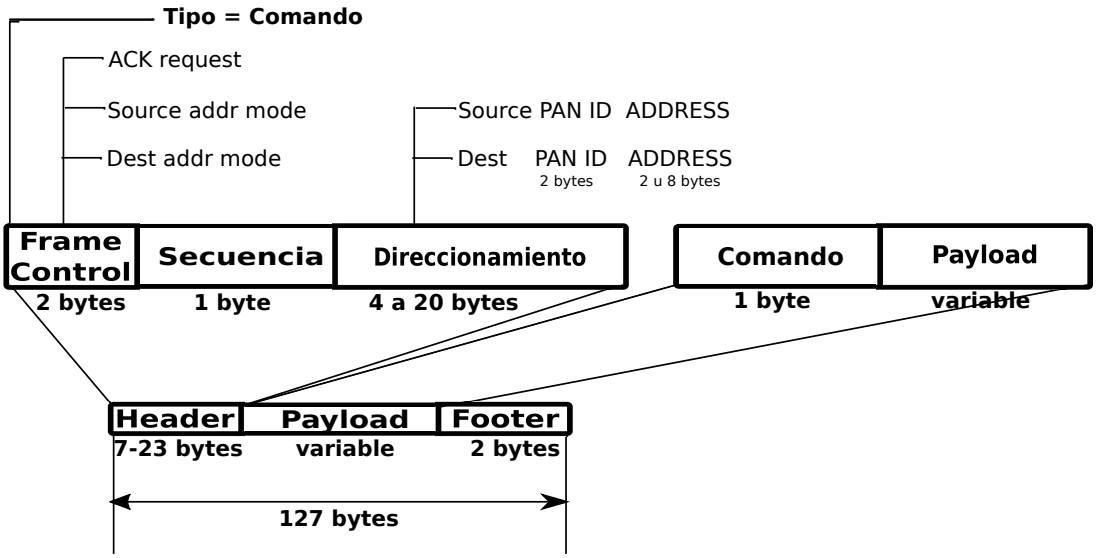



Figura 3.8: Estructura de la trama de comandos MAC.

- Trama beacon.
  Esta trama es utilizada únicamente por el coordinador para transmitir beacons [21], los cuales sirven para sincronizar el reloj de todos los dispositivos dentro de la red, también proporciona información acerca de los paquetes que tiene el coordinador pendientes para cierto dispositivo, el cual deberá comunicarse con él para obtenerlos, este proceso es conocido como transmisión indirecta. En una *beacon enabled network* es opcional utilizar la trama *super frame*, la cual ayuda a definir el *GTS*, este se encarga de especificar si el *GTS* será para transmitir o para recibir. El campo de dirección pendiente en la carga útil, contiene la dirección de los dispositivos para los que tiene paquetes pendientes el coordinador, en la figura 3.9 se observa la estructura de una trama de este tipo.

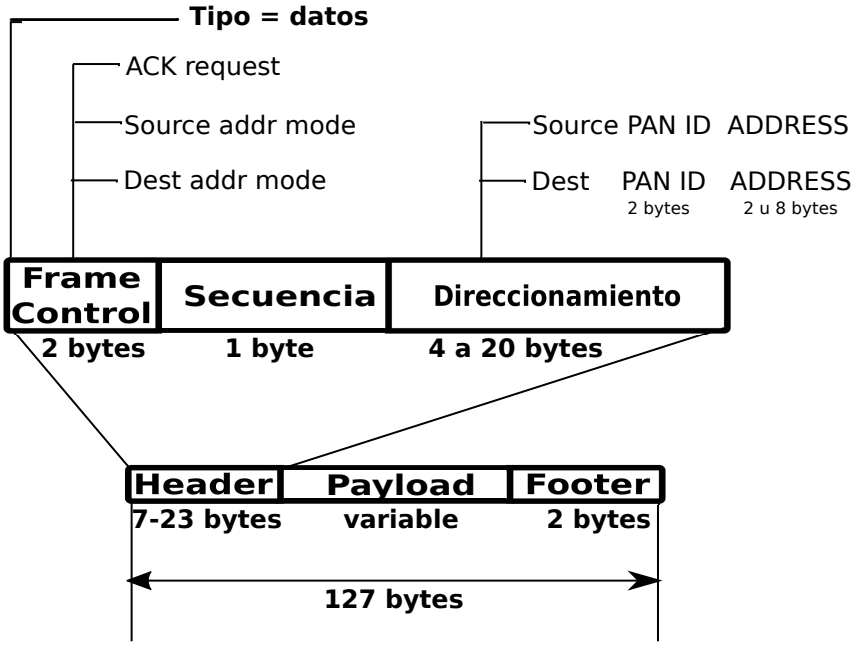



Figura 3.9: Estructura de la trama beacon MAC.

Por ultimo, dentro de cada mensaje se debe de especificar la red en el que este se originó y

a la que va dirigido y esto se realiza con el identificador de red (PAN ID), la dirección de cada dispositivo es única y sirve para identificarlo, esta se puede representar en 64 bits (dirección extendida) o en 16 bits (dirección corta).

#### 3.3.2.2. Acceso al medio

En una red de comunicaciones siempre se plantea el problema sobre cómo administrar el medio de transmisión, ya que todos los enlaces comparten el mismo canal, por lo que los dispositivos que deseen transmitir tienen que cumplir con ciertas reglas especificadas en los protocolos de acceso al medio, estas se crean con el propósito de minimizar las colisiones que se llegan a presentar por un intento de acceso al medio simultáneo provocado por dos o más dispositivos, también para aprovechar al máximo el medio.

En la figura 3.10 se puede ver el diagrama de flujo para cada una de las versiones del método *CSMA-CA*, que utiliza la capa MAC del protocolo 802.15.4, en ambos casos cada que un dispositivo desea transmitir, debe de ejecutar el método *CCA* para garantizar que ningún otro dispositivo esté utilizando el canal, en este algoritmo se consideran tres variables, las cuales son el exponente de *Backoff (BE)*, el número de *Backoffs (NB)* y la longitud de la ventana de contención *(CW)*, más adelante se explicará a detalle la variable *BE*, en tanto, *NB* es un contador que realiza un seguimiento de la cantidad de veces que el dispositivo colisiona y vuelve a intentar acceder al canal, tiene un valor inicial de cero y un valor máximo de *macMaxCSMABackoffs*.

Es importante destacar que al utilizar cualquiera de las dos versiones de *CSMA-CA*, es probable que se presenten colisiones, ya que este método solo trata de evitarlas basándose en la poca probabilidad de que dos o más dispositivos elijan el mismo número aleatorio.

#### 3.3.2.3. Unslotted CSMA-CA

El estándar IEEE 802.15.4 [20] emplea el método *CSMA-CA*. Si un dispositivo desea transmitir, este debe de esperar un cierto tiempo aleatorio antes de acceder al medio, el cual se calcula eligiendo un número entero de periodos entre 0 y $2^{BE}$, donde *BE* se denomina como *exponetial Backoff* y se incrementa cada vez que haya un intento fallido de transmisión, teniendo como valor inicial $3$ (*macMinBE*) y como valor máximo $5$ (*aMaxBE*), cada periodo dura $20$ símbolos (*aUnitBackoffPeriod*). En la expresión 3.6 se puede apreciar la duración en segundos. Una vez transcurrido dicho periodo de tiempo, el dispositivo trata de identificar si el canal está libre por medio del método *CCA*, esta función tarda 8 símbolos en realizarse, esto se representa en segundos en la igualdad 3.1, para 2.4 GHz.

$$CCA_{time} = \frac{8[sym]}{62500\left[\frac{sym}{s}\right]} = 128[\mu s] \tag{3.1}$$

Si se llegará a presentar otra colisión y el valor del *exponetial Backoff* es igual a *aMaxBE*, se intenta transmitir el paquete $4$ veces más (*macMaxCSMABackoffs*) y si después de esto no ha sido posible la transmisión, este se desecha.

#### 3.3.2.4. Slotted CSMA-CA

Este tipo de acceso al medio es utilizado en una red con beacon [20], la cual se rige por *time slots*, el *super frame* divide el periodo activo en $16$ intervalos de tiempos iguales. Cuando un dispositivo desea transmitir, primero debe de ubicar los extremos de los *time slots*, el primer extremo es cuando se inicia la transmisión de una trama beacon, con este a su vez empieza también el primer periodo de *backoff*, *slotted CSMA-CA* establece que una vez localizado el límite de un periodo de *backoff*, el dispositivo debe esperarse cierto tiempo aleatorio, el cual se obtiene eligiendo un número entero de periodos entre $0$ y $2^{BE} - 1$, la nomenclatura es la misma que en las redes sin beacon en las que se usa *unslotted CSMA-CA* y también el valor de *BE* se incrementa con cada intento fallido de transmisión, estando acotado a $5$ (*aMaxBE*), cada periodo corresponde a un *aUnitPeriodBackoff*, el cual se muestra en la ecuación 3.6.

Si al transcurrir dicho periodo de tiempo se detecta el canal libre, el dispositivo deberá de esperar un tiempo *CW (contention window)* de dos *time slots* antes de transmitir. Si se desea ahorrar energía, existe la capacidad de *Battery Life Extention*, la cual limita el valor de *BE* a un máximo de 2, lo que permite que el dispositivo espere menos tiempo antes de transmitir, esto siempre y cuando el tráfico sea bajo, ya que no habrá necesidad de retransmitir y dormirá por más tiempo.

Al ser un método de acceso al canal ranurado, los dispositivos solo pueden transmitir dentro de estos periodos de tiempo, ocasionando que si dos o más dispositivos desean transmitir en periodos cercanos, el sistema los fuerza a hacerlo dentro del mismo *time slot* y lo único que separa estas transmisiones es la elección de un número aleatorio. La probabilidad de que se presente una colisión de $n$ estaciones en este método está dada por la expresión 3.2, esta se cumple siempre y cuando $n \leq 2^{BE}$.

$$P_c = 1 - \frac{2^{BE}!}{(2^{BE})^n((2^{BE}) - n)!} \tag{3.2}$$

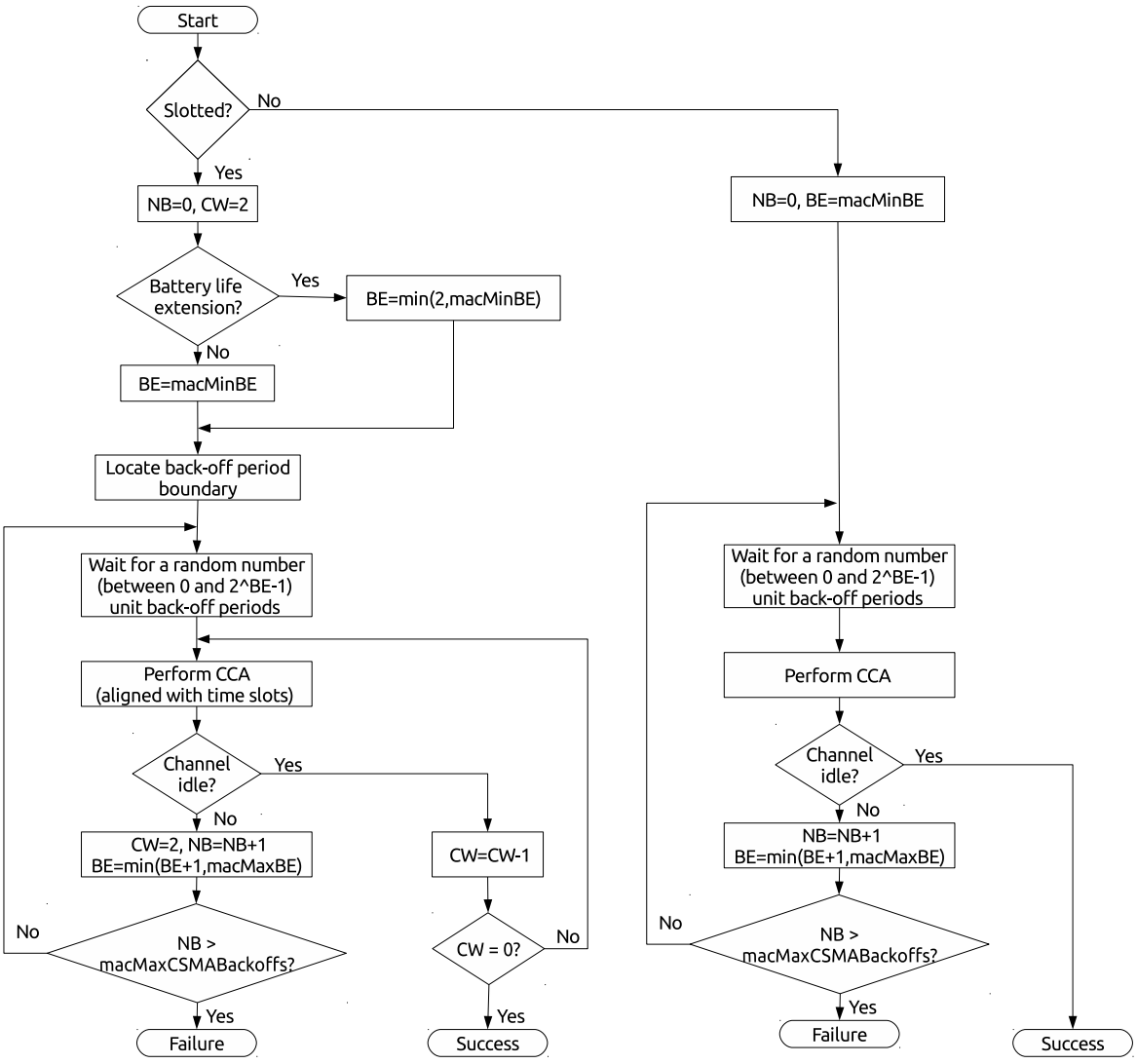



Figura 3.10: Algoritmo CSMA-CA.

### 3.3.3. Espacio entre tramas

Cuando se presenta un enlace de comunicación entre dispositivos, el dispositivo que transmite debe de esperar un periodo de tiempo antes de entrar en el proceso de transmisión nuevamente, esto para que el receptor procese la información que recibe, a este evento se le conoce como *Inter Frame Spacing (IFS)* [21], cuya duración depende de la longitud de la trama que se envió, ya que si una trama no cuenta con un tamaño mayor a $18\,\text{bytes}$ (*aMaxSIFSFrameSize*), la duración es de $12$ símbolos (*aMinSIFSPeriod*) lo que se traduce como un periodo corto y toma el nombre de *Short Inter Frame Spacing (SIFS)*, pero por el contrario, si la trama es mayor a lo especificado anteriormente, el tiempo de espera es de $40$ símbolos lo que se traduce como un periodo largo y toma el nombre de *Long Inter Frame Spacing (LIFS)*. Considerando la velocidad de transmisión de símbolos para la banda de $2.4\,\text{GHz}$ mostrada en la tabla 3.1, se puede calcular la duración en segundos para un SIFS y un LIFS, como se muestra en las expresiones 3.3 y 3.4, respectivamente.

$$SIFS = \frac{12[sym]}{62500\left[\frac{sym}{s}\right]} = 192[\mu s] \tag{3.3}$$

$$LIFS = \frac{40[sym]}{62500\left[\frac{sym}{s}\right]} = 640[\mu s] \tag{3.4}$$

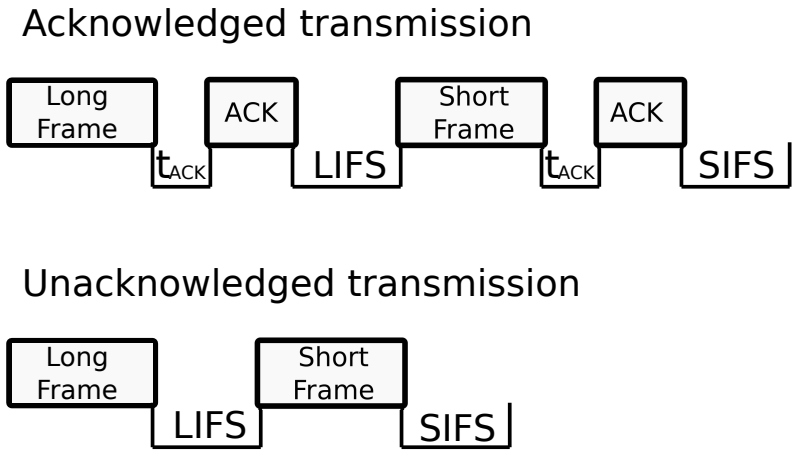



Figura 3.11: *Interframe Spacing (IFS)* en una transmisión con *acknowledged* y sin *acknowledged*.

Como se observa en la figura 3.11, en el primer escenario las tramas transmitidas transportan en su interior una solicitud de acuse de recepción, por lo que el dispositivo transmisor no puede empezar un SIFS o un LIFS hasta que reciba la trama ACK que confirme la recepción de la trama transmitida. El tiempo que transcurre entre el mensaje transmitido y la trama ACK recibida se simboliza como $t_{ACK}$. Si las tramas transmitidas no llevan consigo una solicitud de acuse de recepción, el SIFS o LIFS empieza a transcurrir una vez que el mensaje se haya terminado de enviar.

### 3.3.4. Confirmación de recepción

Cuando una trama se envía, se puede solicitar al dispositivo receptor un acuse de recepción, esta solicitud debe de ser contestada con una trama ACK, si esta no es recibida por el dispositivo transmisor durante un cierto tiempo de espera, dicho dispositivo retransmitirá el paquete, ya que dará por hecho que la trama no fue recibida, este proceso se repetirá hasta que se reciba la trama ACK o hasta que se retransmita el mismo paquete 3 veces, dicho valor es el número máximo de retransmisiones *(aMaxFrameRetries)* [20].

Una trama ACK no se transmitirá antes de que transcurran 12 símbolos *(aTurnaround-Time)*, ni después de que hayan pasado $20$ símbolos *(aUnitBackOffPeriod)*, este periodo de tiempo es representado como $t_{ACK}$, el cual se encuentra entre los valores calculados en las ecuaciones 3.5 y 3.6 para $2.4\,\mathrm{GHz}$.

$$aTurnaroundTime = \frac{12[sym]}{62500\left[\frac{sym}{s}\right]} = 192[\mu s] \tag{3.5}$$

$$aUnitBackOffPeriod = \frac{20[sym]}{62500\left[\frac{sym}{s}\right]} = 320[\mu s] \tag{3.6}$$

Por otra parte, el dispositivo que espera una trama ACK, esperará como máximo que transcurran $54$ símbolos *(macACKWaitDuration)*, cuya equivalencia en tiempo es el que se muestra en la ecuación 3.7 para $2.4\,\mathrm{GHz}$.

$$macACKWaitDuration = \frac{54[sym]}{62500\left[\frac{sym}{s}\right]} = 864[\mu s] \tag{3.7}$$

Un dispositivo que envía una trama sin indicar que se solicita confirmación, puede suponer que la transmisión se recibió con éxito y no realizará el procedimiento de retransmisión [27].

### 3.3.5. Detección de errores

La trama MAC cuenta con el campo *FCS*, cuya función es detectar errores, este emplea el concepto de verificación de redundancia cíclica *(CRC)* mediante el polinomio CRC-16 ( ver ecuación 3.8), de la *International Telecommunication Union (ITU)* [21].

$$CRC_{16} = x^{16} + x^{12} + x^5 + 1 \tag{3.8}$$

Para generar el campo *FCS*, los bits de los campos *MHR* y *MAC payload* del dispositivo transmisor son considerados coeficientes de un polinomio, el cual es dividido por otro, que es conocido tanto por el receptor, como por el transmisor. El residuo de esta operación es el valor del campo *FCS*. Cuando el receptor recibe la trama, comprueba que la comunicación fue exitosa al obtener el mismo resultado que el que se encuentra en el campo de *FCS*, si ambos resultados son diferentes, el receptor concluirá que la trama tiene errores.

Un ejemplo de división polinómica basado en la aritmética binaria es el que se muestra a continuación, en donde el numerador sería el número binario 10111 y el dividendo sería 100, los polinomios correspondientes a dichos valores son los que se muestran en las expresiones 3.9 y 3.10, respectivamente.

$$10111 \rightarrow 1 \times x^4 + 0 \times x^3 + 1 \times x^2 + 1 \times x^1 + 1 \times x^0 = x^4 + x^2 + x + 1 \tag{3.9}$$

$$100 \rightarrow 1 \times x^2 + 0 \times x^1 + 0 \times x^0 = x^2 \tag{3.10}$$

Una vez calculados los polinomios, solo resta realizar la división como se muestra en 3.12.

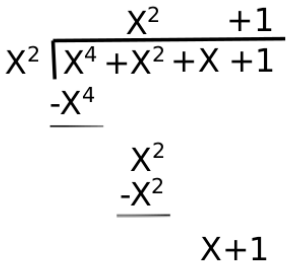



Figura 3.12: División polinómica.

El residuo de la operación mostrada en 3.12 es $x + 1$ cuyo valor binario sería 011.

Para generar el campo FCS en el estándar IEEE 802.15.4 se siguen los siguientes pasos:

1. Determinar el polinomio $M(x)$, el cual representa los bits que conforman a los campos *MHR* y *MAC payload*.

2. Multiplicar el polinomio $M(x)$ por el polinomio $x^{16}$, para generar $M(x) \times x^{16}$.

3. Por último, dividir el polinomio $M(x) \times x^{16}$ entre el polinomio $CRC_{16}$.

## 3.3.6. Comunicación entre dispositivos

### 3.3.6.1. Transferencia de datos de un *device* a un coordinador

1. *Nonbeacon-enabled network*

   Si un *device* requiere mandar información al coordinador, debe de transmitir una trama de datos utilizando *CSMA-CA*, tan pronto como el canal se encuentre libre, y si la trama lo especifica, el coordinador devolverá una trama *ACK* inmediatamente sin utilizar *CSMA-CA* para confirmar la recepción de los datos [21] (ver figura 3.13).

2. *Beacon-enabled network*

   Si un *device* quiere comunicarse con un coordinador, debe de transmitir el mensaje utilizando *slotted CSMA-CA* dentro del periodo *CAP*, el coordinador solo devolverá una trama *ACK* inmediatamente sin utilizar *CSMA-CA* para confirmar la recepción de los datos, si esta es solicitada por el transmisor. Si el *device* realizó una solicitud de *GTS* y le es aceptada por el coordinador, el *device* podrá transmitir durante el periodo *CFS* el tiempo que se le haya asignado sin tener que utilizar *CSMA-CA* [20] (ver figura 3.13).

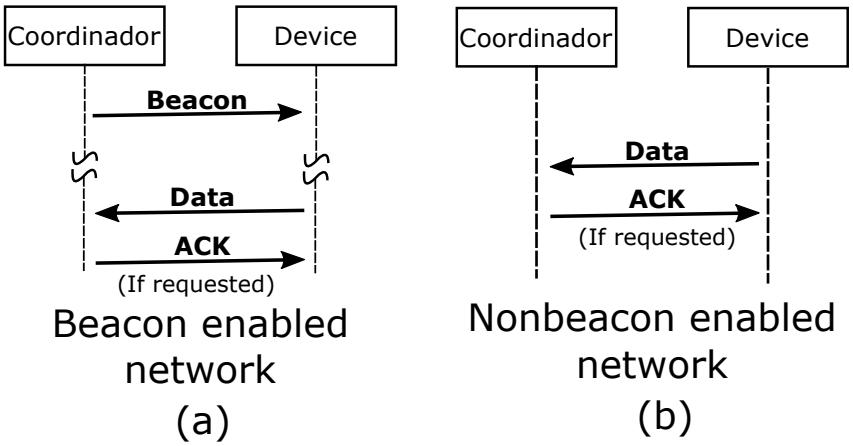



Figura 3.13: Transferencia de datos de un *device* a un coordinador en una *Beacon-enabled network* (a) y en una *Nonbeacon-enabled network (b).*

#### 3.3.6.2. Transferencia de datos de un coordinador a un *device*

1. *Nonbeacon-enabled network*

   Si un coordinador requiere mandar información a un *device* en particular, espera a que este le mande una trama *Data request*, a la que deberá de contestar con una trama *ACK*, para después empezar a mandar los datos correspondientes a dicho *device*, utilizando *CSMA-CA* y si la trama lo especifica, el *device* devolverá una trama *ACK* inmediatamente sin utilizar *CSMA-CA* para confirmar la recepción de los datos [21] (ver figura 3.14).

2. *Beacon-enabled network*

   Si un coordinador requiere mandar información a un *device* en particular, éste al transmitir una trama *beacon* lo indica dentro de ella, para después esperar a que dicho *device* le mande una trama de *Data request* indicando que está listo para recibir, entonces el coordinador le deberá de contestar con una trama *ACK* sin tener que utilizar *CSMA-CA*, una vez realizado el proceso anterior, utilizando *slotted CSMA-CA* el coordinador transmitirá la trama de datos al *device*, enviará la trama *ACK*, para confirmar la recepción de los datos [20] (ver figura 3.14).

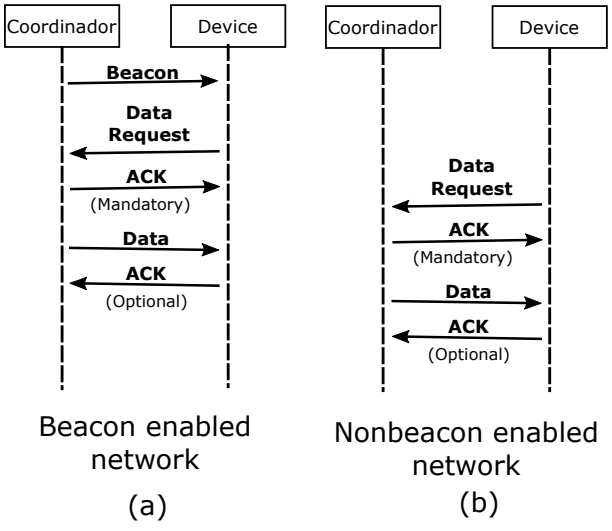



Figura 3.14: Transferencia de datos de un coordinador a un *device* en una *Beacon-enabled network* (a) y en una *Nonbeacon-enabled network (b).*

#### 3.3.6.3. Transferencia de datos *Peer-to-Peer*

En una topología punto a punto, cada dispositivo puede comunicarse directamente con cualquier otro dispositivo.

### 3.3.7. Inicialización de una red

Para crear una red [20], el coordinador PAN realiza el siguiente procedimiento:

- Mide la energía presente en los canales a los que tiene permitido el acceso, para poder seleccionar el que presente mejores condiciones para poder transmitir.
- Puede realizar una búsqueda pasiva *(passive scan)*, la cual consiste en escuchar los *beacons* transmitidos por otros coordinadores PAN que estén ocupando el canal para identificar los *ID's PAN* que ya están ocupados y poder así realizar la elección de uno nuevo.
- También puede realizar una búsqueda activa *(active scan)*, en la cual el coordinador PAN emite *beacons request* en cada uno de los canales, a los que tiene permitido el acceso, y espera que los otros coordinadores PAN activos le respondan con un *beacon* para poder armar una lista de *ID's PAN* utilizados y así poder escoger uno nuevo.
- Al contar con un *ID PAN* el coordinador PAN comienza a transmitir *beacons* periódicos en una *beacon-enabled network* o espera en silencio en una red *nonbeacon-enabled network*

### 3.3.8. Asociación

Cuando un dispositivo quiere unirse a una red, este debe de encontrar un canal en condiciones óptimas para transmitir y de igual forma a un coordinador PAN con el que pueda comunicarse, a este le enviará una trama de *associaton request* a la cual el coordinador le responderá a su vez con una trama de *association response* en donde le notificará si le es aceptada su petición o rechazada.

### 3.3.9. Desasociación

Es un procedimiento que realiza un dispositivo perteneciente a una red, para notificar al coordinador que tiene la intención de dejar la red, los motivos pueden ser que el coordinador PAN quiera que el dispositivo abandone la red o que el mismo dispositivo desee dejarla, esto se realiza mediante el mensaje *dissasociation notification*.

### 3.3.10. Problema de la terminal oculta y de la terminal expuesta

El problema del nodo oculto [21], se presenta cuando dos o más terminales no se detectan entre sí, ocasionando que dichas terminales al escanear el canal lo detecten libre, cuando no lo está, y transmitan al mismo tiempo generando una colisión en el dispositivo receptor, el cual recibe la información de los dispositivos transmisores sin poder discernir que es lo que envía cada uno. Un ejemplo de este escenario se muestra en la figura 3.15, en la que se observa que el nodo A y C se encuentran muy distanciados, de tal forma que no se logran percibir entre sí, sin embargo ambos nodos pueden comunicarse con el nodo B. Si el nodo C comienza a transmitir información al nodo B, el nivel de energía de su señal en la ubicación del nodo A sería tan bajo que el dispositivo A no sería capaz de detectar que el canal se encuentra ocupado, por lo tanto consideraría que es libre de transmitir, esta misma situación se repetiría en C si A estuviera transmitiendo a B. Más aún, si tanto el nodo A como el nodo C deciden mandar un mensaje al nodo B en el mismo instante, esto provocaría una colisión de paquetes en B.

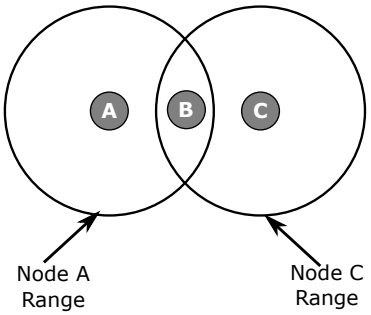



Figura 3.15: Terminal oculta.

Por otro lado, el problema de la terminal expuesta [20] es ejemplificado en la figura 3.16, en la que se observa que el nodo E puede mandar mensajes al nodo D, mientras el nodo F puede mandar mensajes al nodo G. El nodo D no percibe la energía de la señal del nodo F, por lo tanto, puede comunicarse con el nodo E, pero CSMA-CA no permite que el nodo E transmita ya que este se encuentra dentro del rango de transmisión del nodo F, y esto ocasiona que el proceso de *CCA* del nodo E considere que el canal está ocupado mientras transmita el nodo F.

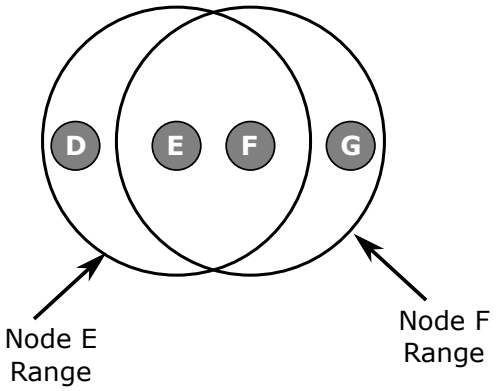



Figura 3.16: Terminal expuesta.

### 3.3.11. Solución al problema de la terminal oculta y expuesta en IEEE 802.15.4

En otros estándares, como el 802.11, se utiliza el mecanismo *request-to-send/clear-to-send (RTS/CTS)* para reducir el número de colisiones provocadas por estos dos problemas. No obstante, dicho sistema no es incluido en la capa de control de acceso al medio (MAC) del estándar IEEE 802.15.4 [20].

En [28] se explican las principales razones por las que la capa MAC del protocolo IEEE 802.15.4 no considera la implementación del mecanismo de enlace RTS/CTS, las cuales son:

- La adopción de dicho mecanismo añade sobrecarga al protocolo.
- Al no contar con el protocolo de enlace RTS/CTS se reduce la complejidad del sistema.

Sin embargo, en el experimento realizado en [28], se propone y analiza el uso del método RTS/CTS en IEEE 802.15.4 para una red *nonbeacon-enable*. Este utiliza la función de concatenación de paquetes, con la cual se logra disminuir la cantidad de colisiones de paquetes que son provocadas por los problemas de la terminal oculta y expuesta, lo que permite disminuir el número de paquetes retransmitidos y, por ende, mejorar la eficiencia del canal.

En [20] se recomienda cambiar de ubicación a los dispositivos o incrementar su potencia de transmisión para garantizar que los demás nodos puedan detectar las señales que transmiten,

esto con el fin de solucionar el problema de la terminal oculta. En cuanto al problema de la terminal expuesta, se propone modificar la posición de los dispositivos o reducir la potencia de transmisión de los nodos al mínimo requerido para una comunicación confiable. Ambas soluciones son a nivel físico, ya que a nivel de software, la capa MAC del protocolo 802.15.4 hace poco para ayudar a resolver dichos problemas.

# Capítulo 4

# Simulador de la capa MAC del protocolo 802.15.4

Una red *beacon enabled* se basa en la sincronización de los dispositivos, la cual nunca es perfecta debido al desfasamiento que se llega a presentar en los relojes de los equipos con el paso del tiempo. Esta cualidad es muy costosa porque entre mayor sea la exactitud del reloj, mayor es el precio de la tecnología que se implementa. Aunado a esto, es un costo energético sincronizar los dispositivos cada determinado tiempo. Para evitar tener una red con estas desventajas, la mayoría de los sistemas implementados en la realidad se basan en una red *nonbeacon enabled*. Por ser el tipo de red con más ventajas. Por este motivo, en esta tesis se simula una red *nonbeacon enabled*, mediante el uso de los procesos descritos en el capítulo 3. Para ello, la implementación de la capa MAC del protocolo 802.15.4 se realizó mediante la biblioteca SimPy, la cual es un marco de simulación de eventos discretos [18].

El simulador está conformado por 7 clases, las cuales fueron programadas en su totalidad y al convivir en un entorno reproducen el comportamiento de la capa MAC del protocolo 802.15.4. La estructura y el código que componen al simulador se muestran en los apéndices A-G. Cabe resaltar que esta tesis está orientada únicamente a modelar la capa MAC del estándar 802.15.4.

## 4.1. Clase nodo

La clase nodo contiene las funciones principales especificadas en el estándar IEEE 802.15.4 (ver capitulo 3). Entre estas funciones se encuentran, por ejemplo, el tiempo que un nodo debe de esperar por una trama *ACK* después de transmitir un paquete de datos o comandos, el número de intentos de retransmisión de un paquete después de alcanzar el valor máximo del *exponential backoff*. En esta clase, las funciones se clasificaron como principales, secundarias y auxiliares.

Las funciones principales son aquellas mencionadas en el capítulo 3, y son los procesos esenciales dentro de la simulación, por ejemplo, el método *CSMA_CA_UNS()*, el cual realiza el procedimiento mostrado en la figura 3.10. El método *CCA()* que se encarga de realizar el proceso *clear channel assessment*. También se encuentra la creación, transmisión y recepción de paquetes, entre otras (ver apéndice A).

Las funciones secundarias son aquellas cuya función es auxiliar a las primarias, básicamente realizan cálculos como el método *probability(P)* el cual recibe como parámetro la probabilidad de éxito de un evento y con base a esta regresa un estado (verdadero o falso). Otro ejemplo, es el método *getEB()* el cual calcula el valor aleatorio del tiempo que un nodo deberá de esperar antes de transmitir.

Finalmente, las funciones auxiliares son métodos que apoyan a otras funciones que no se pueden ejecutar directamente durante la simulación, un ejemplo es la función *auxACKRx()* la cual se encarga de llamar a la función *ACKRx()* y de realizar la configuración pertinente para que dicha función se ejecute con normalidad, en este caso, el primer método puede ser

interrumpido por el segundo, ya que todos los procesos se llevan a cabo dentro de la función *ACKRx()*, si no existiera *auxACKRx()* y los procesos se intentaran interrumpir desde el método *ACKRx()*, la simulación fallaría debido a que SimPy no permite que una función se interrumpa así misma.

El detalle del funcionamiento de los métodos que componen a esta clase se puede consultar en el apéndice A.

## 4.2. Clase nodo PAN

Esta clase es la encargada de crear e inicializar a los nodos que conforman la red junto con sus procesos. A cada nodo se le asigna un rol dentro de la red como coordinador o *device* y los comandos que puede utilizar.

Para crear la red de comunicaciones, primero se construye el canal, a través del cual se darán los procesos de transmisión y recepción de paquetes, para después asignarle a cada nodo generado una lista con los dispositivos con los que podrá comunicarse, conocidos como vecinos.

Es importante mencionar que el nodo PAN hereda todos los métodos y atributos de la clase nodo, debido a que el dispositivo que se asigne como coordinador de la red no pierde su naturaleza como nodo.

El detalle del funcionamiento de las métodos que componen a esta clase se pueden consultar en el apéndice B.

## 4.3. Clase paquete

La clase para generar objetos del tipo paquete está conformada por atributos que identifican al nodo fuente, así como al nodo al que va dirigido. También cuenta con parámetros característicos del paquete como el número de secuencia, el tamaño en bytes, el número de intentos de retransmisión, el tipo de trama (datos, comandos o *acknowledge*) y si se requiere confirmación de recepción.

Cada trama está conformada por el *MAC Header*, el *MAC payload* y el *MAC footer*. Estos tres campos conforman el *PHY payload*, y la suma de este campo más los campos de *Synchronization header* y el *PHY header* dan como resultado el tamaño del paquete a transmitir. El detalle de los métodos que componen a esta clase se pueden consultar en el apéndice C.

## 4.4. Clase canal

La clase canal esta diseñada de tal forma que puedan coexistir varias trasmisiones simultáneas sin importar que estas colisionen o que sean exitosas, ya que al tratarse de un medio inalámbrico más de un nodo puede cumplir con las condiciones establecidas por el estándar y transmitir en un mismo instante de tiempo.

Esta clase es la que se encarga de discernir si un enlace de comunicación entre dos nodos se da correctamente o es interrumpido por alguna otra transmisión, ya que dentro de sus atributos se almacenan datos como los nodos que se encuentran transmitiendo y los paquetes que estos envían. Si se presenta una colisión, esta clase es la encargada de indicar que los paquetes implicados contienen errores y es a causa de interferencia.

El detalle del funcionamiento de los métodos que componen a esta clase se pueden consultar en el apéndice D.

## 4.5. Clase impresión

Esta clase es la encargada de reunir los datos más relevantes de cada proceso que se ejecuta durante la simulación y a su vez mostrarlos de una forma resumida y entendible con el fin de que el usuario pueda realizar un mejor análisis de la información. Las funciones de

esta clase se componen de los métodos que se enfocan a la impresión de datos en la consola y un método que está destinado a la compilación de datos en un archivo de texto.

La tabla 4.1 muestra un resumen de las acciones que se imprimen en la consola junto con los datos que se repiten con mayor frecuencia entre los procesos. Si el proceso cuenta con un atributo de la tabla, la casilla contendrá la palabra "Sí", de lo contrario se observará la palabra "No", también es importante destacar que el dato "Estatus" para cada proceso puede tomar valores diferentes, así que en caso de que el proceso cuente con este atributo, se mostrarán los posibles estados de este.

| **Acción** | **Requiere ACK** | **Tipo de trama** | **Número de secuencia** | **Nodo receptor** | **Nodo transmisor** | **Tiempo actual** | **Estatus** |
|---|---|---|---|---|---|---|---|
| Creación de paquete | Sí | Sí | Sí | Sí | Sí | Sí | No |
| Transmisión | Sí | Sí | Sí | Sí | Sí | Sí | Empieza<br>Termina<br>Se tira |
| Transmisión de un ACK | No | Sí | Sí | Sí | Sí | Sí | Empieza<br>Termina<br>Esperando tiempo de ACK |
| Recepción | Sí | Sí | Sí | Sí | Sí | Sí | Empieza<br>Termina |
| Recepción de un ACK | No | Sí | Sí | Sí | Sí | Sí | Empieza<br>Termina<br>Colisiono<br>Recepción fallida |
| *CCA* | No | No | No | No | Sí | Sí | Empieza<br>Termina |
| *Exponential Backoff* | No | No | No | No | Sí | Sí | Empieza<br>Termina<br>Se pausa<br>Se reanuda |
| *IFS* | No | No | No | No | Sí | Sí | Empieza<br>Termina |
| Colisiones | No | No | No | Sí | Sí | Sí | No |

Tabla 4.1: Datos que se muestran en la terminal con mayor frecuencia.

En la tabla 4.2 se muestran los posibles campos para cada acción. Es importante mencionar que la impresión en consola es controlada por el usuario, es decir, éste puede habilitarla o deshabilitarla en caso de que no requiera tanta información, pero siempre que se ponga en funcionamiento el simulador, se generará un archivo de texto, el cual contendrá un informe de los procesos de transmisión y recepción de paquetes. Este archivo de texto muestra los eventos más importantes que ocurren durante la simulación, este cuenta con un formato amigable para el usuario, ya que es importante poder interpretar la información recabada en él, con el fin de facilitar la manipulación de los datos, por ejemplo, puede ser de gran utilidad cuando se requiera filtrar algún evento en especifico. En la figura 4.1 se muestra el formato que tiene cada línea del documento.

| **Acción** | Transmisión | Transmisión de un ACK | Recepción | Recepción de un ACK | *CCA* | *Exponential Backoff* |
|---|---|---|---|---|---|---|
| **Campos** | 1. Tamaño del paquete<br>2. Número de retransmisión del paquete<br>3. Número de paquetes transmitidos<br>4. Número de paquetes descartados<br>5. Valor del *exponential backoff*<br>6. Número de repeticiones del valor máximo del *exponential backoff* | 1. Tamaño del paquete<br>2. Número de paquetes transmitidos<br>3. Número de paquetes descartados<br>4. Tiempo de *ACK* | 1. Estado del paquete<br>2. Número de paquetes recibidos | 1. Estado del paquete | 1. Estado del canal | 1. Valor del *exponential backoff*<br>2. Número de repeticiones del valor máximo del *exponential backoff*<br>3. Tiempo de *exponential backoff* |

Tabla 4.2: Campos particulares de cada proceso.

| ***Evento*** | ***Nodo transmisor*** | ***Nodo receptor*** | ***Tipo de paquete*** | ***Número de retransmisiones del paquete*** | ***El paquete contiene errores*** | ***Tamaño del paquete*** | ***Estatus del evento*** | ***Tiempo*** |
|---|---|---|---|---|---|---|---|---|

Figura 4.1: Formato de la estructura del archivo de texto.

Análisis de cada campo:

1. Evento, este campo puede tomar los valores *r* de recepción, *t* de transmisión, *c* de creación de un paquete, *dt* o *dr* de paquete desechado en el transmisor o en el receptor, respectivamente.
2. El nodo transmisor es el nodo fuente del que proviene el paquete.
3. El nodo receptor es el nodo al que va destinado la trama.
4. El tipo de paquete puede ser de comandos (COM), datos (DAT) o *acknowledge* (ACK).
5. El quinto campo indica el número de veces que se ha transmitido un paquete.
6. El sexto campo señala si un paquete contiene errores, ya sea por interferencia o por colisión.
7. El séptimo campo indica el tamaño de la trama en Bytes.
8. El estatus del evento indica si el proceso ha comenzado o terminado.
9. El último campo indica el tiempo en el que ocurre la acción.

El detalle del funcionamiento de los métodos que componen a esta clase se pueden consultar en el apéndice E.

## 4.6. Simulador

El usuario debe modificar el *script* mostrado en el apéndice F, en el cual introducirá los parámetros para el funcionamiento del simulador. Si se desea seguir algún parámetro de la simulación, que no se muestre en el archivo de texto ni en la impresión de datos en la consola, o realizar algún cambio al código fuente, el usuario tendrá que interactuar con una o

más clases de las que se han mencionado con anterioridad, pero si no es el caso, únicamente se tiene que configurar el *script* y ejecutarse como se muestra en la línea 4.1.

```
$ python simulacion.py
```

4.1: Comando para ejecutar el *script* del simulador.

Es importante mencionar que para su ejecución se necesita tener instalado Python 2.7 y que la terminal este ubicada en la carpeta donde se almacenan los archivos, el *script* a ejecutar es el que se muestra en el apéndice *F*. Los parámetros a configurar en el *script* se muestran en el código 4.2.

Es importante mencionar que para esta parte del simulador se utilizó Python 2.7, debido a que el trabajo se inició en esta versión, sin embargo es fácilmente escalable a la versión Python 3.5, ya que hay solo dos funciones que cambian entre versiones: *filter()* y *raw_input()*. Por lo que solo se tienen que buscar las funciones equivalentes.

```
RANDOM_SEED = 45 #Random number to repeat the results
SIM_TIME = 0.5  #Simulation time
printFunctions = [False, True] #to enable or disable printing to console and text file, respectively
timeBetweenPack = 0.2 #Generate a packet approximately every x seconds
probCOMFrame = 0.2 #Probability of creating a command package every time a data package is created
probError = 0.1 #Probability that a package contains errors, without considering collisions
probACK = 0.5 #Probability that a package requires ACK
nodePANName = 'A'
#First test
grafo = {'A':['B'], 'B':['A']}
#Second test
#grafo = {'A':['B', 'C', 'D', 'E', 'F', 'G'], 'B':['A'], 'C':['A'], 'D':['A'], 'E':['A'], 'F':['A'], 'G':['A']}
#Third test
#grafo = {'A':['B', 'C', 'D', 'E'], 'B':['A'], 'C':['A', 'I', 'H'], 'D':['A', 'L', 'K', 'J'],
#    'E':['F', 'G', 'A'], 'F':['E'], 'G':['E'], 'H':['C'], 'I':[ 'C'], 'J':['D'], 'K':['D'],
#    'L':['D']}
```

4.2: Parámetros a configurar para el correcto funcionamiento del simulador.

A continuación se describe cada uno de los valores a configurar:

- RANDOM_SEED, a esta variable se le puede asignar cualquier valor y sirve para poder reproducir los mismos resultados cada vez que se ponga en funcionamiento el simulador.
- SIM_TIME, es el valor del tiempo, en segundos, que va a durar la simulación.
- printFunctions, es un arreglo de valores booleanos. El primer elemento de esta lista sirve para habilitar o deshabilitar la impresión en consola, el segundo elemento habilita o deshabilita la compilación de información en el archivo de texto.
- timeBetweenPack, es el valor de tiempo promedio que un nodo tiene que esperar para generar un nuevo paquete.
- probCOMFrame, es el valor de probabilidad a favor de que se genere un paquete de comandos cada que vez que se genera uno de datos.
- probError, es el valor de probabilidad a favor de que un paquete contenga errores.
- probACK, es el valor de la probabilidad a favor de que un paquete requiera un acuse de recepción.

- nodePANName, sirve para especificar el nombre del coordinador PAN de la red.
- grafo, en esta variable se almacena un diccionario que contiene la topología de red a simular. Las llaves del diccionario corresponden a los nodos y el valor de las llaves es una lista que contiene a los vecinos de cada nodo. Por ejemplo, el grafo correspondiente al diccionario:

$$\{'A' : ['B','C','D','E'],'B' : ['A'],'C' : ['A','I','H'],'D' : ['A','L','K','J'],'E' : ['F','G','A'],$$

$$'F' : ['E'],'G' : ['E'],'H' : ['C'],'I' : ['C'],'J' : ['D'],'K' : ['D'],'L' : ['D']\}$$

es el que se muestra en la figura 4.2.

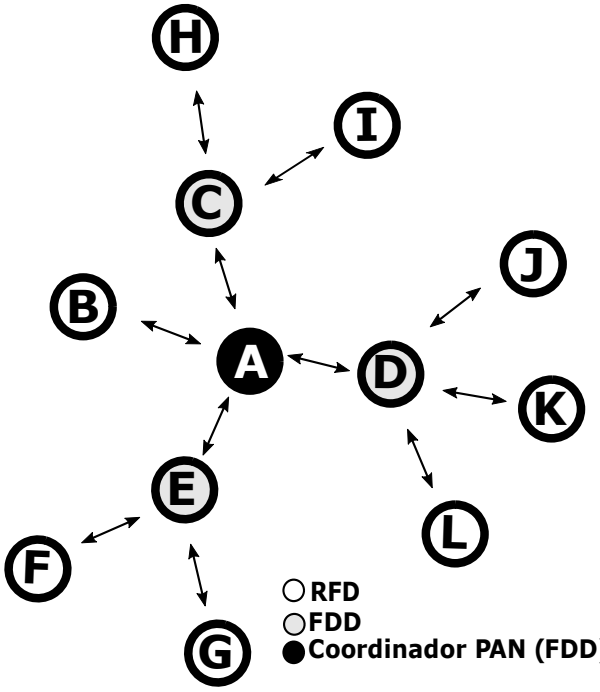



Figura 4.2: Red de prueba con topología tipo árbol.

El detalle del funcionamiento de los métodos que componen a esta clase se pueden consultar en el apéndice F.

## 4.7. Animación

Para poder realizar la animación, el *script* utiliza las bibliotecas *networkx* y *vpython*. La primera se especializa en el estudio de las funciones de redes complejas, y la segunda facilita la creación de pantallas y animaciones en 3D navegables.

Es importante destacar que este *script* está dirigido a los usuarios que se les dificulte analizar la información tanto de la consola como del archivo de texto y necesiten algo más gráfico para entender como se van desarrollando los procesos, por lo tanto, es elección del usuario si se crea o no la animación.

Para crear la animación, primero se debe de ejecutar el proceso de simulación descrito con anterioridad, ya que el *script* de animación tiene como entrada el archivo de texto resultante de la simulación. La ejecución del *script* de animación se muestra en la línea 4.3.

```
$ python3 Interface2.py
```

4.3: Comando para ejecutar el *script* de animación.

Para ejecutar correctamente el *script* se necesita tener instalado python 3 y que la terminal este ubicada en la carpeta donde se almacenan los archivos, el *script* a ejecutar es el que se muestra en el apéndice *G*, el usuario debe de configurar los parámetros que se muestran en el código 4.4.

```
graph = {'A':['B', 'C', 'D', 'E'], 'B':['A'], 'C':['A', 'I', 'H'], 'D':['A', 'L',
  'K', 'J'],
   'E':['F', 'G', 'A'], 'F':['E'], 'G':['E'], 'H':['C'], 'I':[ 'C'], 'J':['D'], 'K
  ':['D'],
   'L':['D']}
f = open ('Registro.txt', 'r')
nodePANName = 'A'
```

4.4: Parámetros a configurar para el correcto funcionamiento de la animación.

A continuación se describe cada uno de los valores a configurar:

- `graph`, en esta variable se almacenará un diccionario que contenga la topología de red, es importante que sea igual al que se coloco en el simulador.
- `f`, es un apuntador al archivo donde se almacenan los datos de la simulación.
- `nodePANName`, sirve para especificar el nombre del coordinador PAN de la red.

Un ejemplo de animación se muestra en la figura 4.3.

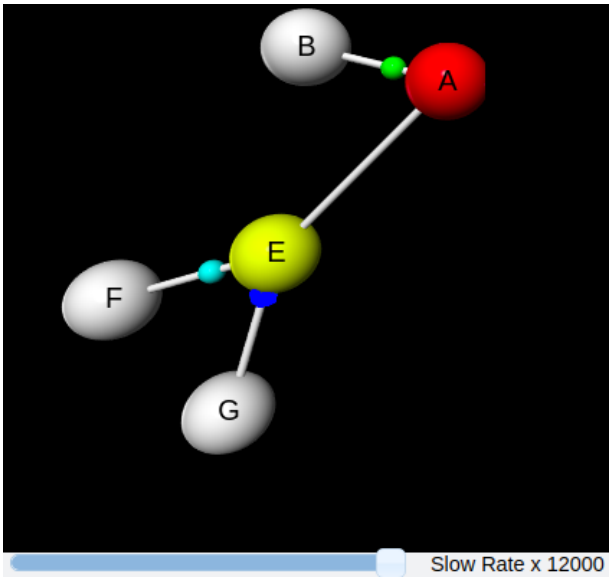



Figura 4.3: Ejemplo de los objetos que pueden existir en una animación.

Los objetos que pueden existir en la animación son:

1. Coordinador PAN, es representado en forma de esfera de color rojo.
2. *Full-function devices (FFDs)*, son representados en forma de esfera de color amarillo.
3. *Reduced-function devices (RFDs)*, son representados en forma de esfera de color blanco.
4. Trama de datos, son representadas en forma de esfera de color cyan.

5. Trama de comandos, son representadas en forma de esfera de color azul marino.

6. Tramas *ACK*, son representadas en forma de esfera de color verde.

Por último, la animación cuenta con un *scroll* el cual sirve para reducir la velocidad de los procesos y así poder observarlos de mejor manera. El valor mínimo de la velocidad es 100, debido a que con este valor la animación aún es perceptible, y el valor máximo es de 12000. Un ejemplo de su funcionamiento es el que se muestra en la figura 4.3, donde se puede observar que el scroll se encuentra posicionado en su valor máximo, y esto significa que la animación se reproduce 12000 veces más lento.

El detalle del funcionamiento de los métodos que componen a esta clase se pueden consultar en el apéndice G.

# Capítulo 5

# Resultados y análisis de datos

En este capítulo se presentan tres escenarios con distinta densidad de nodos con el fin de analizar el comportamiento del simulador. Para ello se toman las siguientes consideraciones:

- Cada nodo se ve forzado a crear un paquete de datos con una frecuencia de $0.2$ segundos, sin embargo, el simulador también es capaz de generar paquetes cada cierto tiempo aleatorio definido por medio de la distribución de *Poisson*.
- La probabilidad de que se genere una trama de comandos cada vez que se origina una de datos se estableció en 20 %.
- La probabilidad de que un paquete contenga errores al ser recibido, sin considerar los errores por colisión, se estableció en 10 %.
- La probabilidad de que un paquete requiera un acuse de recepción se estableció en 50 %.

Se realizó la configuración de está forma con el fin de monitorizar el correcto funcionamiento del simulador ante casos particulares que tienen poca probabilidad de ocurrir, con el fin de abarcar la mayor cantidad de eventos. Para los tres escenarios el tiempo de simulación fue de $0.5$ segundos, esto se estableció así por cuestiones de visualización y para verificar el correcto funcionamiento del simulador conforme a lo descrito en el capitulo 3. Además, se buscó que los procesos graficados en los esquemas de tiempos fueran entendibles y perceptibles. Para lograrlo, en cada prueba se escogió un pequeño intervalo del tiempo simulado, en el que se presentan la mayor cantidad de eventos posibles. Sin embargo, cabe resaltar que el simulador es capaz de simular cualquier cantidad de tiempo.

La primera prueba consiste en una red con topología punto a punto de dos nodos. La segunda se basa en una red con topología estrella de siete nodos, y en la última prueba se establece una red con topología tipo árbol con $12$ nodos. Se propusieron estos tres escenarios debido a que son topologías aceptadas por el estándar IEEE 802.15.4, además de que con dicha cantidad de nodos se hace más fácil el análisis y la visualización de los datos.

Con el fin de entender de mejor manera los resultados, la salida de cada una de las pruebas consiste de:

1. Una figura en la que se muestra el registro de datos en el archivo de texto.
2. Un gráfico de tiempo que explica los resultados obtenidos en el archivo de texto.
3. Una figura que muestra una parte de los datos que imprime la consola.
4. Una figura de tiempo que ilustra los resultados de la impresión en consola.
5. Una figura que muestra un *frame* de la animación.

Es importante destacar que el rango de cobertura de cada nodo se obtiene a partir del grafo que el usuario ingresa al simulador, ya que a través del diccionario se intuye cuales dispositivos podrán establecer un canal de comunicación. De igual forma, para construir

la animación, no es necesario que el usuario ingrese la posición de cada nodo, ya que el programa se encarga de ubicarlos en el espacio de trabajo utilizando la misma estructura de datos antes mencionada.

Para un mejor análisis de los datos se puede activar la impresión por consola, ya que esta opción muestra información más detallada de cada proceso. Los diagramas que se muestran en estos experimentos son realizados con la información proveniente de la terminal, ya que proporciona una mayor cantidad de datos. Sin embargo, dependiendo de la información requerida por el usuario, se debe activar o desactivar esta opción, ya que si se desea un análisis sin tantos detalles, es mucho más práctico el archivo de texto. Además de otras ventajas como que es más rápido el proceso, y más fácil de visualizar.

## 5.1. Prueba con dos nodos

Esta prueba consiste de una red simple compuesta por solo dos nodos, un coordinador PAN y un dispositivo *RFD* como se muestra en la figura 5.1. El nodo *A* se encuentra ubicado en $(0, 0)$ y el nodo *B* está posicionado en $(-1, 3)$.

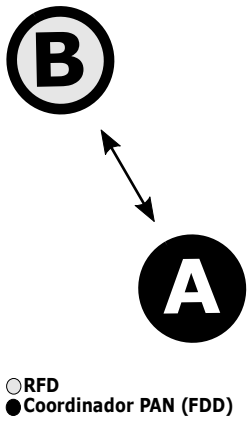



Figura 5.1: Topología de dos nodos.

### 5.1.1. Datos del archivo de texto

Los resultados obtenidos en el archivo se muestran en la figura 5.2. A través de estos datos se generó el esquema de tiempo de la figura 5.3.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | c | A | B | DAT | - | - | 110 | - | 4e-06 |
| 2 | c | B | A | DAT | - | - | 130 | - | 4e-06 |
| 3 | r | A | B | DAT | - | None | 110 | Comienza | 0.000772 |
| 4 | t | A | B | DAT | 0 | - | 110 | Comienza | 0.000772 |
| 5 | r | A | B | DAT | - | False | 110 | Termina | 0.004292 |
| 6 | t | A | B | DAT | 0 | - | 110 | Termina | 0.004292 |
| 7 | t | B | A | DAT | 0 | - | 130 | Comienza | 0.004932 |
| 8 | r | B | A | DAT | - | None | 130 | Comienza | 0.004932 |
| 9 | r | B | A | DAT | - | False | 130 | Termina | 0.009092 |
| 10 | t | B | A | DAT | 0 | - | 130 | Termina | 0.009092 |
| 11 | c | A | B | DAT | - | - | 139 | - | 0.200004 |
| 12 | c | A | B | COM | - | - | 137 | - | 0.200004 |
| 13 | c | B | A | DAT | - | - | 145 | - | 0.200004 |
| 14 | c | B | A | COM | - | - | 141 | - | 0.200004 |
| 15 | t | A | B | DAT | 0 | - | 139 | Comienza | 0.200132 |
| 16 | r | A | B | DAT | - | None | 139 | Comienza | 0.200132 |
| 17 | t | A | B | DAT | 0 | - | 139 | Termina | 0.20458 |
| 18 | r | A | B | DAT | - | False | 139 | Termina | 0.20458 |
| 19 | t | A | B | COM | 0 | - | 137 | Comienza | 0.205348 |
| 20 | r | A | B | COM | - | None | 137 | Comienza | 0.205348 |
| 21 | r | A | B | COM | - | False | 137 | Termina | 0.209732 |
| 22 | t | A | B | COM | 0 | - | 137 | Termina | 0.209732 |
| 23 | t | B | A | DAT | 0 | - | 145 | Comienza | 0.211204 |
| 24 | r | B | A | DAT | - | None | 145 | Comienza | 0.211204 |
| 25 | r | B | A | DAT | - | False | 145 | Termina | 0.215844 |
| 26 | t | B | A | DAT | 0 | - | 145 | Termina | 0.215844 |
| 27 | t | A | B | ACK | 0 | - | 11 | Comienza | 0.216036 |
| 28 | r | A | B | ACK | - | None | 11 | Comienza | 0.216036 |
| 29 | r | A | B | ACK | - | False | 11 | Termina | 0.216388 |
| 30 | t | A | B | ACK | 0 | - | 11 | Termina | 0.216388 |

Figura 5.2: Ejemplo de datos en el archivo de texto de la simulación con dos nodos.

En esta primera prueba se puede observar que se transmitieron un total de $6$ paquetes, desde que inicio la simulación hasta los $216.388\,\mathrm{ms}$, de los cuales ninguno contenía errores y todos fueron recibidos correctamente. El nodo *A* transmitió $4$ paquetes al nodo *B*, mientras que el nodo *B* transmitió $2$ paquetes al nodo *A*.

Cada vez que se inicia el proceso de transmisión, se indica el tipo de trama, el nodo receptor y el tamaño del paquete. Para el caso cuando se transmite una trama *ACK* se identifica con un rectángulo rojo que contiene una letra *"T"* en su interior, mientras para el caso de una recepción de una trama *ACK* se identifica con un rectángulo rojo que contiene una letra *"R"*. Para la recepción de cualquier otro tipo de paquete, se indica solo la acción debido al hecho de que solo son dos nodos, y es evidente la fuente de la que proviene la trama.

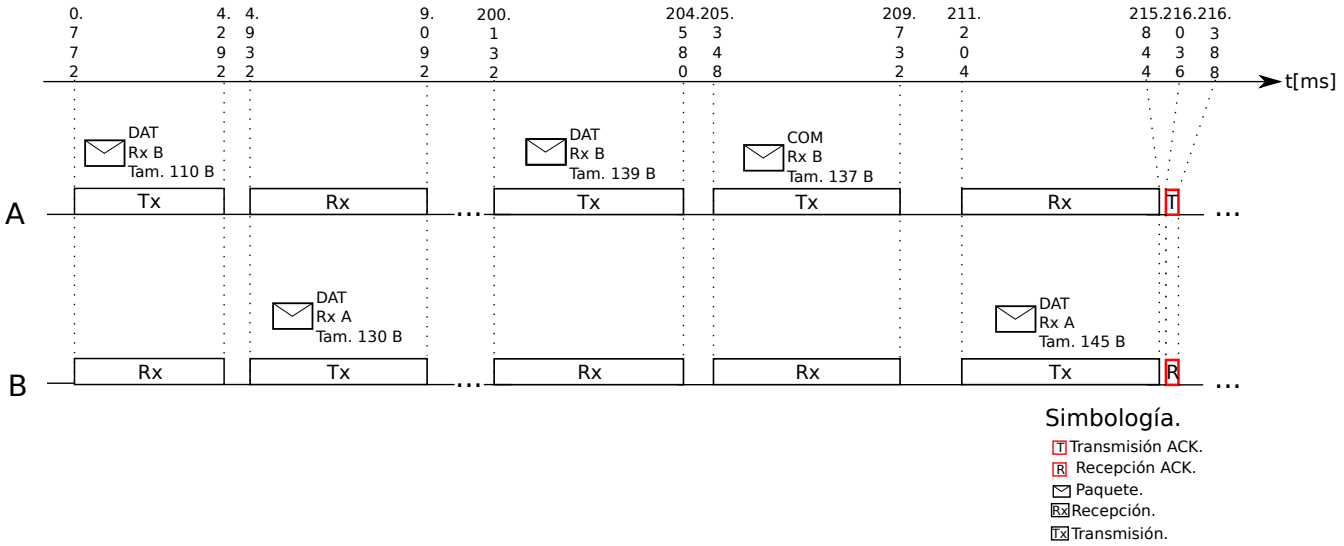



Figura 5.3: Representación de los datos obtenidos en el archivo de texto de la simulación con dos nodos.

### 5.1.2. Datos de la terminal

En la figura 5.4 se muestra solo una parte de los datos arrojados en la terminal. A través de estos datos se creó el esquema de tiempo mostrado en la figura 5.5. Este esquema muestra los mismos datos mostrados en la figura 5.3, pero esta vez se añaden los procesos *LIFS*, *SIFS*, tiempo de *ACK*, tiempo de *exponential backoff* y tiempo de *CCA*, los cuales complementan el análisis. En la figura 5.4 también se muestra la acción que toma cada proceso y los parámetros que la componen, como se indicó en el capitulo 4.

```
Accion: Recepcion                Estatus: Comienza
Tipo de paquete: DAT             Requiera ACK: False
Nodo Rx: B                       Contiene errores: None
Nodo Tx del paquete: A           Colisiono: ---
Numero de secuencia:0            Paquetes Rx:0
Nodos Tx: ['A']
Tiempo: 0.000772
====================================================================
Pulsa una tecla para continuar...
====================================================================
Accion: Transmision              Tamano en Bytes del paquete: 110
Tipo de paquete: DAT             Requiere ACK: False
Nodo Tx: A                       Numero de secuencia:0
Numero de retransmision: 0       Estatus: Comienza
EB:3                             Paquetes Tirados:0
Repeticiones del EB maximo:0     Paquetes Tx:0
Nodo Rx: B                       Tiempo: 0.000772
====================================================================
Pulsa una tecla para continuar...
====================================================================
Accion: Recepcion                Estatus: Termina
Tipo de paquete: DAT             Requiera ACK: False
Nodo Rx: B                       Contiene errores: False
Nodo Tx del paquete: A           Colisiono: False
Numero de secuencia:0            Paquetes Rx:1
Nodos Tx: []
Tiempo: 0.004292
====================================================================
Pulsa una tecla para continuar...
====================================================================
Accion: Transmision              Tamano en Bytes del paquete: 110
Tipo de paquete: DAT             Requiere ACK: False
Nodo Tx: A                       Numero de secuencia:0
Numero de retransmision: 0       Estatus: Terminado
EB:3                             Paquetes Tirados:0
Repeticiones del EB maximo:0     Paquetes Tx:1
Nodo Rx: B                       Tiempo: 0.004292
====================================================================
```

Figura 5.4: Ejemplo de datos en la terminal de la simulación con dos nodos.

En la figura 5.5 se puede realizar un seguimiento del tiempo que un nodo permanece a la espera hasta que cumple su proceso de tiempo de *exponential backoff*. Dicho evento puede ser continuo o puede estar fraccionado en periodos de tiempo, ya que este proceso se contabiliza cada vez que el canal está libre, de lo contrario se interrumpe y se reanuda una vez que el medio se desocupe. Un ejemplo de esto se puede observar en la línea de tiempo del nodo *B*, considerando el lapso de tiempo que abarca desde los $0.004\,\text{ms}$ a los $0.772\,\text{ms}$, en el que dicho nodo está a la espera de que se cumpla su tiempo de *exponential backoff*. Se puede ver como este proceso se ve interrumpido en $0.772\,\text{ms}$, debido a que el canal es ocupado por el nodo *A*, ya que este inicia su proceso de transmisión, por lo que el nodo *B* se tiene que esperar y solo podrá continuar con su proceso hasta que el nodo A termine de transmitir, es decir, en el tiempo $4.292\,\text{ms}$. En la expresión 5.1 se puede ver como se calcula el tiempo total que duró el evento del nodo *B*.

$$t_{EB} = (0.772\,\text{ms} - 0.004\,\text{ms}) + (4.804\,\text{ms} - 4.292\,\text{ms}) = 1.28\,\text{ms} \tag{5.1}$$

Además, en el esquema de la figura 5.5 también se pueden distinguir cada uno de los elementos que conforman a la transmisión sin *acknowledge* y con *acknowledge*. Un ejemplo para el primer caso sería en la línea de tiempo del nodo *A*, desde el instante $0.772\,\text{ms}$ hasta los $4.932\,\text{ms}$, y para el segundo caso sería en la línea de tiempo del nodo *B*, desde el instante $211.204\,\text{ms}$ hasta los $217.028\,\text{ms}$.

Cada vez que se inicia el proceso de transmisión se indica el tipo de trama, el nodo receptor, si el paquete requiere *ACK*, el número de secuencia y el tamaño del paquete, y en el caso de que se transmita una trama *acknowledge* se sigue la misma convención que en el esquema

de la figura 5.3.

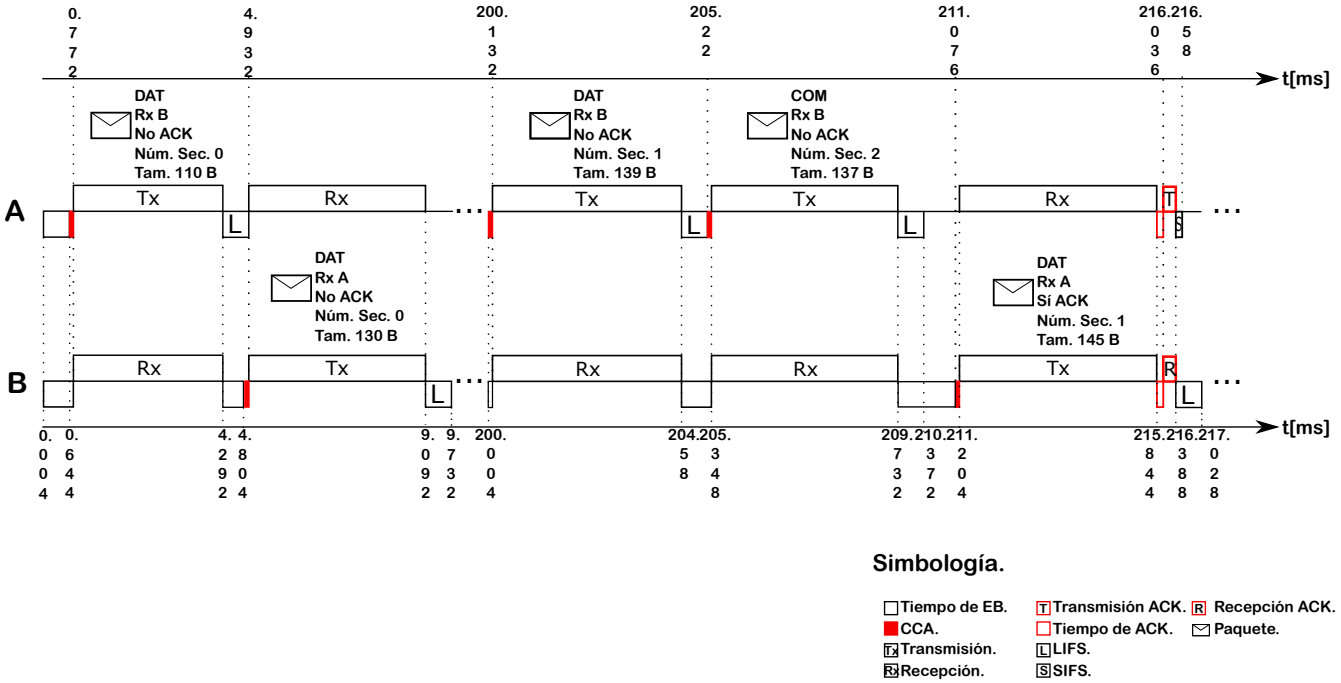



Figura 5.5: Representación de los datos obtenidos en la terminal de la simulación con dos nodos.

### 5.1.3. Animación

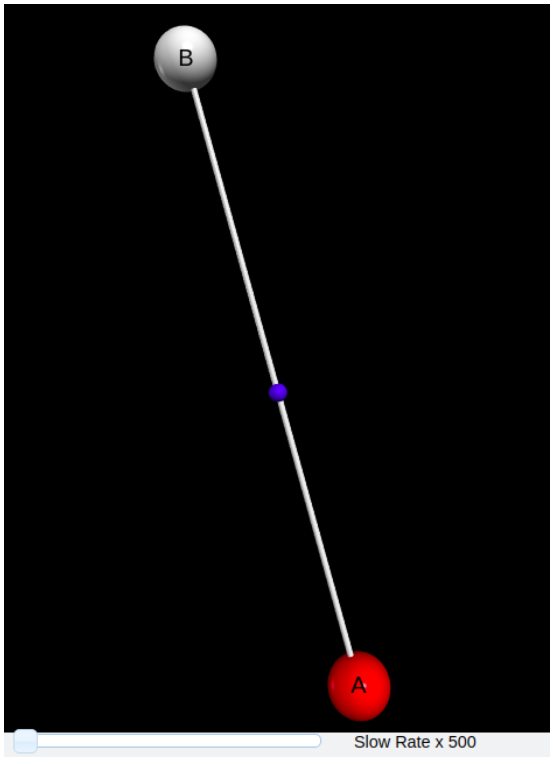



Figura 5.6: Animación de la simulación con dos nodos.

Al realizar la animación de los resultados obtenidos en el archivo de texto, el cual se muestra en la figura 5.2, se obtuvo el *frame* de la figura 5.6, en el que se muestra el momento en el que el nodo *A* transmite un paquete de comandos al nodo *B*. Específicamente, este evento tuvo lugar en el periodo de tiempo que abarca desde los $205.220\,\mathrm{ms}$ hasta los $209.732\,\mathrm{ms}$.

## 5.2. Prueba con topología tipo estrella.

Esta prueba consiste en una red con topología en estrella conformada por siete nodos, de los cuales seis son dispositivos *RFD* y solo uno es el coordinador PAN, como se muestra en la figura 5.7. El nodo *A* se encuentra ubicado en $(0, 0)$, el nodo *B* está posicionado en $(-1.5, 1.5)$, el nodo *C* se localiza en $(-2, -1)$, el nodo *D* se coloco en $(1.5, -1.5)$, el nodo *E* se ubica en $(0.5, 2)$, el nodo *F* está localizado en $(-1, -2)$ y el nodo *G* se posicionó en $(2, 1)$.

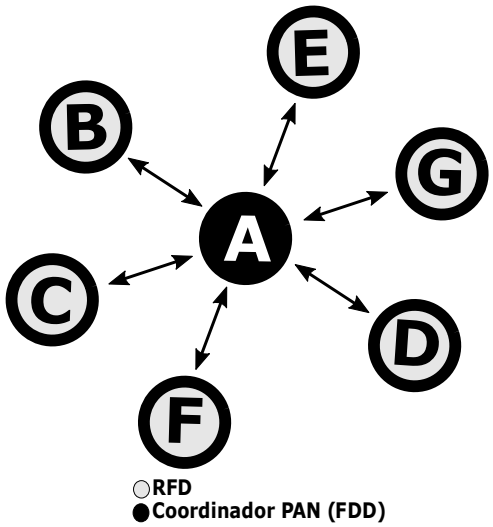



Figura 5.7: Topología tipo estrella.

### 5.2.1. Datos del archivo de texto

La figura 5.8 muestra los datos obtenidos del archivo de texto generado en la ejecución del programa. A partir de estos datos se generaron los esquemas de tiempo de las figuras 5.9 y 5.10. En esta prueba los esquemas de tiempo se dividen en dos partes, esto con el fin de facilitar el análisis de cada uno de los procesos.

```
61 t       G       A       DAT     2       -       137     Comienza        0.016964
62 r       G       A       DAT     -       None    137     Comienza        0.016964
63 t       F       A       DAT     3       -       132     Comienza        0.01802
64 r       F       A       DAT     -       None    132     Comienza        0.01802
65 t       D       A       DAT     2       -       145     Comienza        0.018756
66 t       B       A       DAT     2       -       115     Comienza        0.018756
67 r       D       A       DAT     -       None    145     Comienza        0.018756
68 r       B       A       DAT     -       None    115     Comienza        0.018756
69 t       E       A       DAT     2       -       144     Comienza        0.020292
70 r       E       A       DAT     -       None    144     Comienza        0.020292
71 t       G       A       DAT     2       -       137     Termina         0.021348
72 dr      G       A       DAT     -       True    137     Termina         0.021348
73 t       F       A       DAT     3       -       132     Termina         0.022244
74 dr      F       A       DAT     -       True    132     Termina         0.022244
75 t       B       A       DAT     2       -       115     Termina         0.022436
76 dr      B       A       DAT     -       True    115     Termina         0.022436
77 dt      F       A       DAT     4       -       132     -       0.023108
78 t       D       A       DAT     2       -       145     Termina         0.023396
79 dr      D       A       DAT     -       True    145     Termina         0.023396
80 t       C       A       DAT     2       -       136     Comienza        0.02394
81 r       C       A       DAT     -       None    136     Comienza        0.02394
82 t       E       A       DAT     2       -       144     Termina         0.0249
83 dr      E       A       DAT     -       True    144     Termina         0.0249
84 t       G       A       DAT     3       -       137     Comienza        0.02714
85 r       G       A       DAT     -       None    137     Comienza        0.02714
86 t       B       A       DAT     3       -       115     Comienza        0.027268
87 r       B       A       DAT     -       None    115     Comienza        0.027268
88 t       C       A       DAT     2       -       136     Termina         0.028292
89 dr      C       A       DAT     -       True    136     Termina         0.028292
90 t       C       A       DAT     3       -       136     Comienza        0.029284
91 r       C       A       DAT     -       None    136     Comienza        0.029284
92 t       B       A       DAT     3       -       115     Termina         0.030948
93 dr      B       A       DAT     -       True    115     Termina         0.030948
94 t       G       A       DAT     3       -       137     Termina         0.031524
95 dr      G       A       DAT     -       True    137     Termina         0.031524
96 dt      B       A       DAT     4       -       115     -       0.031812
97 t       E       A       DAT     3       -       144     Comienza        0.032292
```

Figura 5.8: Ejemplo de datos en el archivo de texto de la simulación con una topología tipo estrella.

Los esquemas resultantes de esta prueba muestran los eventos que se presentaron en el periodo de tiempo que abarca desde los $16.964\,\mathrm{ms}$ hasta los $39.172\,\mathrm{ms}$, se puede observar que se realizaron $11$ retransmisiones, de las cuales ninguna fue recibida exitosamente por el nodo receptor, el cual es el mismo para todos los dispositivos que transmitieron durante dicho periodo de tiempo. Todos los dispositivos, a excepción del nodo *A* y el nodo *F*, retransmitieron en $2$ ocasiones. El nodo *A* no transmitió ni retransmitió, y el nodo *F* solo retransmitió una vez.

El estándar 802.15.4 dicta que después de una transmisión fallida, cualquier dispositivo puede retransmitir un máximo de $3$ ocasiones. En esta prueba se muestra el escenario donde todos los nodos, a excepción del nodo *A*, retransmiten su paquete antes de desecharlo. En el esquema de tiempo de la figura 5.9 y 5.10 se representa esta acción por medio de una flecha roja, la cuál sucede en el tiempo $864\,\mu\mathrm{s}$ después de que se transmite la trama. Este valor de tiempo se especifica en el estándar como el tiempo máximo que un nodo puede espera por una trama de confirmación (*ACK*).

Para el caso donde no se requiere un mensaje de *ACK*, entonces, aún cuando la transmisión fallara, el nodo transmisor tomaría el envío como correcto, ya que no tiene forma de saber que falló. Cabe resaltar que se eligió el lapso de tiempo antes mencionado para examinar los detalles más importantes, además de comprobar que se cumplieran los eventos de retransmisión y de descarto de paquetes.

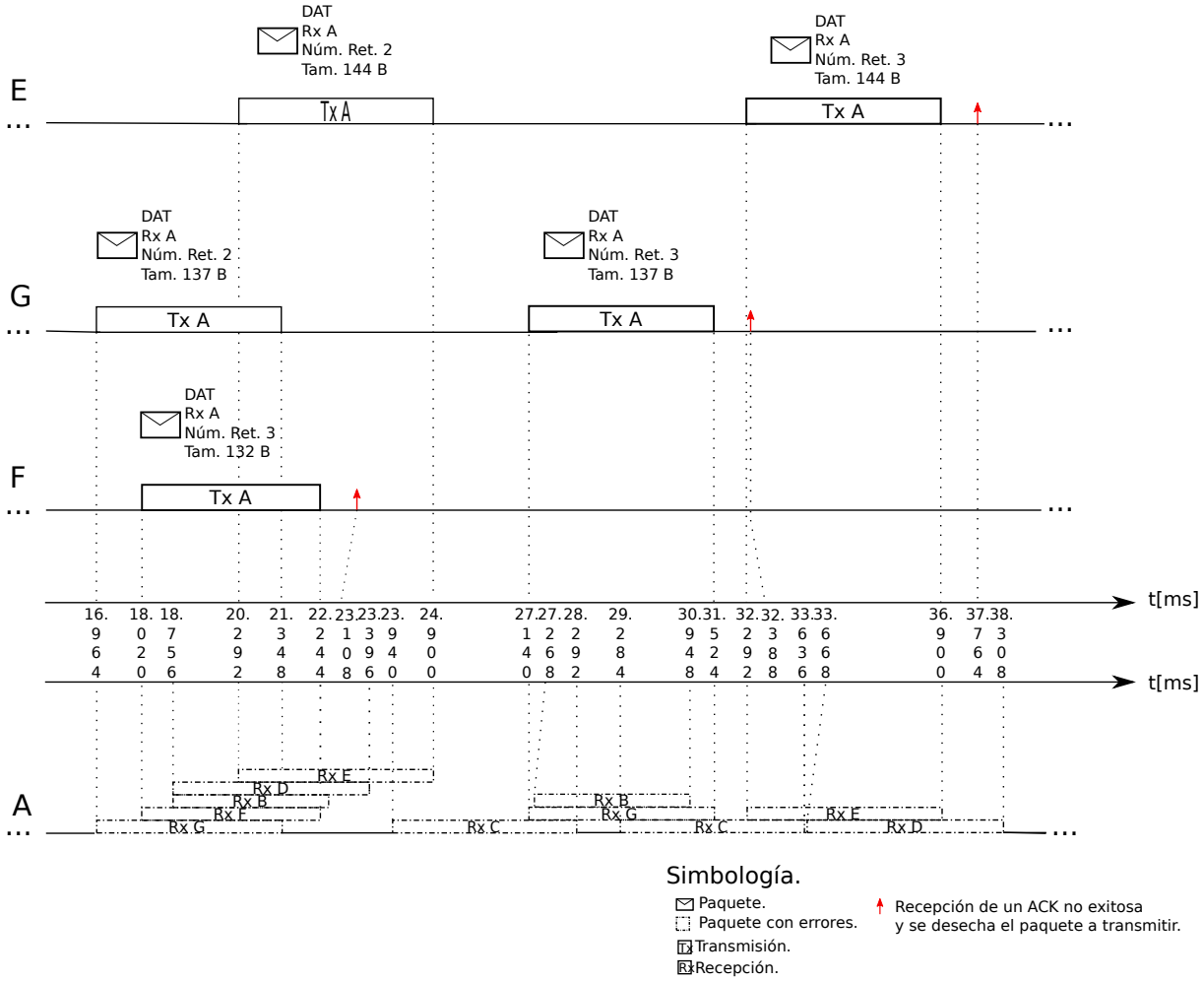



Figura 5.9: Representación de los datos obtenidos en el archivo de texto de la simulación con una topología tipo estrella (primera parte).

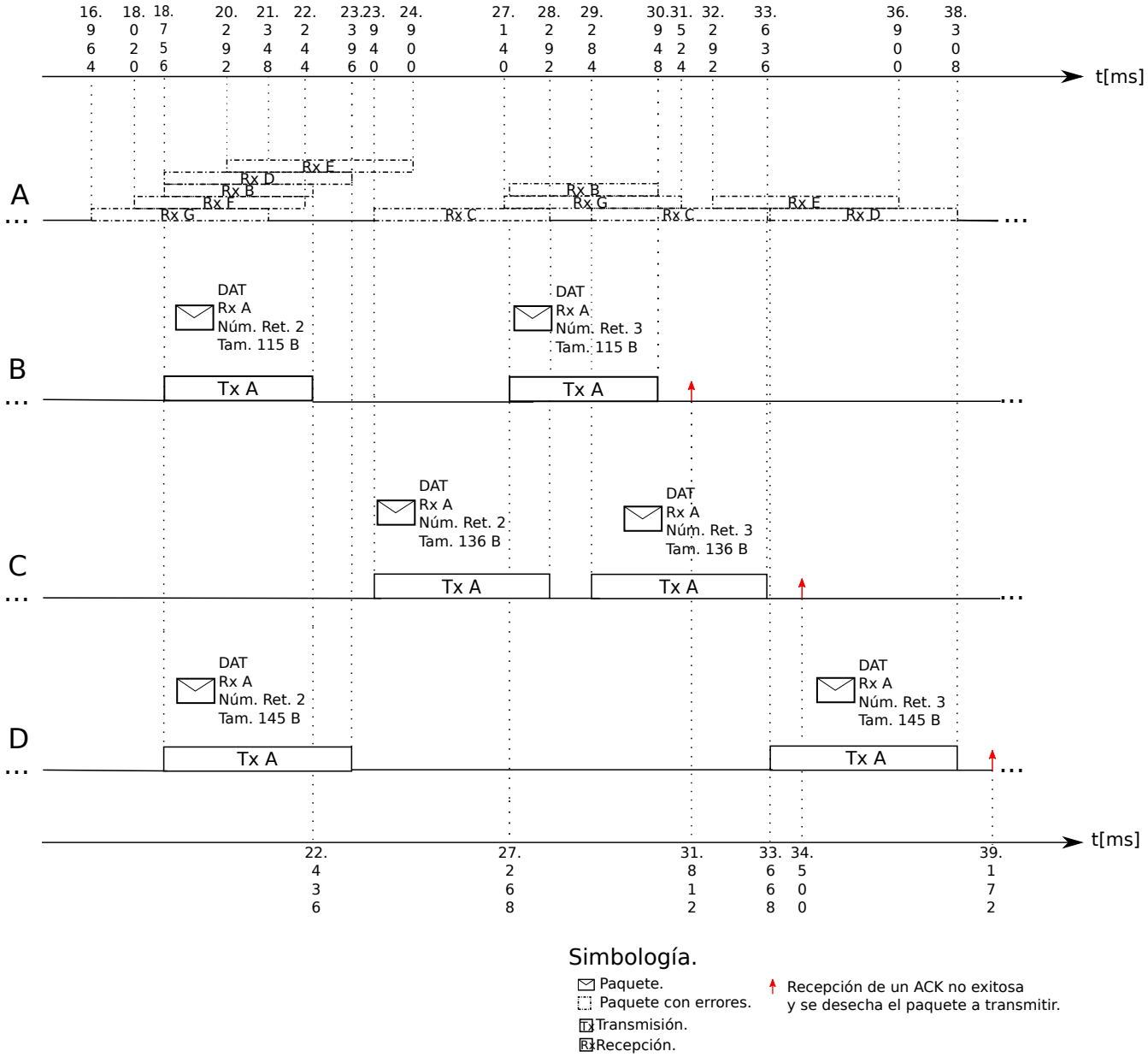



Figura 5.10: Representación de los datos obtenidos en el archivo de texto de la simulación con una topología tipo estrella (segunda parte).

### 5.2.2. Datos de la terminal

Con la información que se extrae de la terminal se crearon los esquemas de tiempo de las figuras 5.12 y 5.13. En estos esquemas se observan los mismos datos que se muestran en los esquemas creados con los datos del archivo de texto, pero también se muestran eventos como la recepción fallida de una trama *ACK*, y los mencionados en la prueba con dos nodos, los cuales complementan el análisis.

En la figura 5.11 se puede observar un ejemplo de los datos que se obtuvieron en la terminal, durante esta simulación.

```
Accion: Recepcion                Estatus: Recepcion fallida
Nodo Tx: A                       Tipo de paquete: ACK
Nodo Rx: F                       Numero de secuencia:0
Tiempo: 0.023108
======================================================================
Pulsa una tecla para continuar...
======================================================================
Accion: Transmision              Tamano en Bytes del paquete: 132
Tipo de paquete: DAT             Requiere ACK: True
Nodo Tx: F                       Numero de secuencia:0
Numero de retransmision: 4       Estatus: Se tira
EB:6                             Paquetes Tirados:1
Repeticiones del EB maximo:1     Paquetes Tx:4
Nodo Rx: A                       Tiempo: 0.023108
======================================================================
Pulsa una tecla para continuar...
======================================================================
Accion: Recepcion                Estatus: Recepcion fallida
Nodo Tx: A                       Tipo de paquete: ACK
Nodo Rx: B                       Numero de secuencia:0
Tiempo: 0.0233
======================================================================
Pulsa una tecla para continuar...
======================================================================
Accion: EB                       Estatus: Comienza
Nodo: B                          Tiempo EB:0.00384
EB:6                             Repeticiones del EB maximo:1
Tiempo: 0.0233
======================================================================
Pulsa una tecla para continuar...
======================================================================
Accion: Transmision              Tamano en Bytes del paquete: 145
Tipo de paquete: DAT             Requiere ACK: True
Nodo Tx: D                       Numero de secuencia:0
Numero de retransmision: 2       Estatus: Terminado
EB:5                             Paquetes Tirados:0
Repeticiones del EB maximo:0     Paquetes Tx:3
Nodo Rx: A                       Tiempo: 0.023396
```

Figura 5.11: Ejemplo de datos en la terminal de la simulación con una topología tipo estrella.

En las figuras 5.12 y 5.13 se puede analizar los procesos que realiza un nodo cuando solicita una trama *ACK* y no la recibe. Por ejemplo, para la línea de tiempo del nodo *E*, en la cual se visualiza que el nodo concluye con el tiempo de *exponential backoff* en el instante $20.164\,\text{ms}$, para después continuar con el evento *clear channel assessment*, que dura $128\,\mu\text{s}$, y al comprobar que el canal se encuentra desocupado, dicho nodo transmite en $20.292\,\text{ms}$. Cuando el dispositivo finaliza esta acción, espera como máximo $864\,\mu\text{s}$ para recibir el *acknowledge*, pero como la recepción falla, entonces el nodo *E* reinicia el proceso de transmisión en $25.764\,\text{ms}$, aumentando en uno el número de retransmisiones con las que cuenta el paquete.

Cada vez que falla el proceso de recepción de un *acknowledge*, esta acción se indica en ambos esquemas con una flecha de color negro, mientras que se usa la flecha de coloro rojo para indicar que un paquete ha superado el número máximo de retransmisiones.

En este experimento también se puede comprobar la existencia de trasmisiones simultáneas que colisionan. Un ejemplo de esto se puede observar en el instante $18.020\,\text{ms}$, en el cual el nodo *A* se encuentra recibiendo un paquete por parte del nodo *G*, pero en ese momento el nodo *F* ha concluido con el evento *CCA*, que ha detectado el canal libre y, por lo tanto transmite su trama ocasionando que tanto el paquete del nodo *G* como el del nodo *F* colisionen. Este es un ejemplo del problema de terminal oculta, del cual se habló en el capítulo 3.

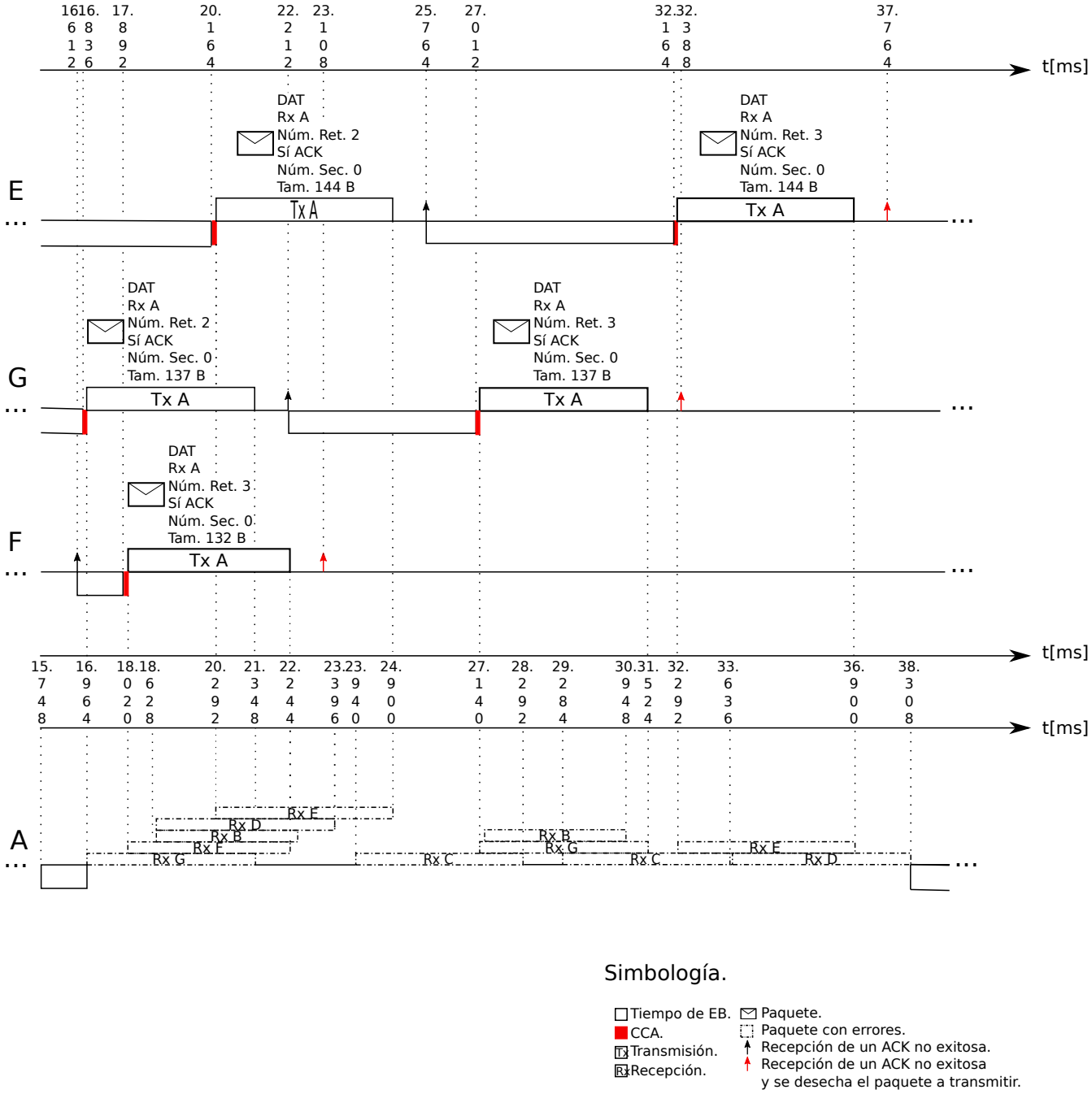



Figura 5.12: Representación de los datos obtenidos en la terminal de la simulación con una topología tipo estrella (primera parte).

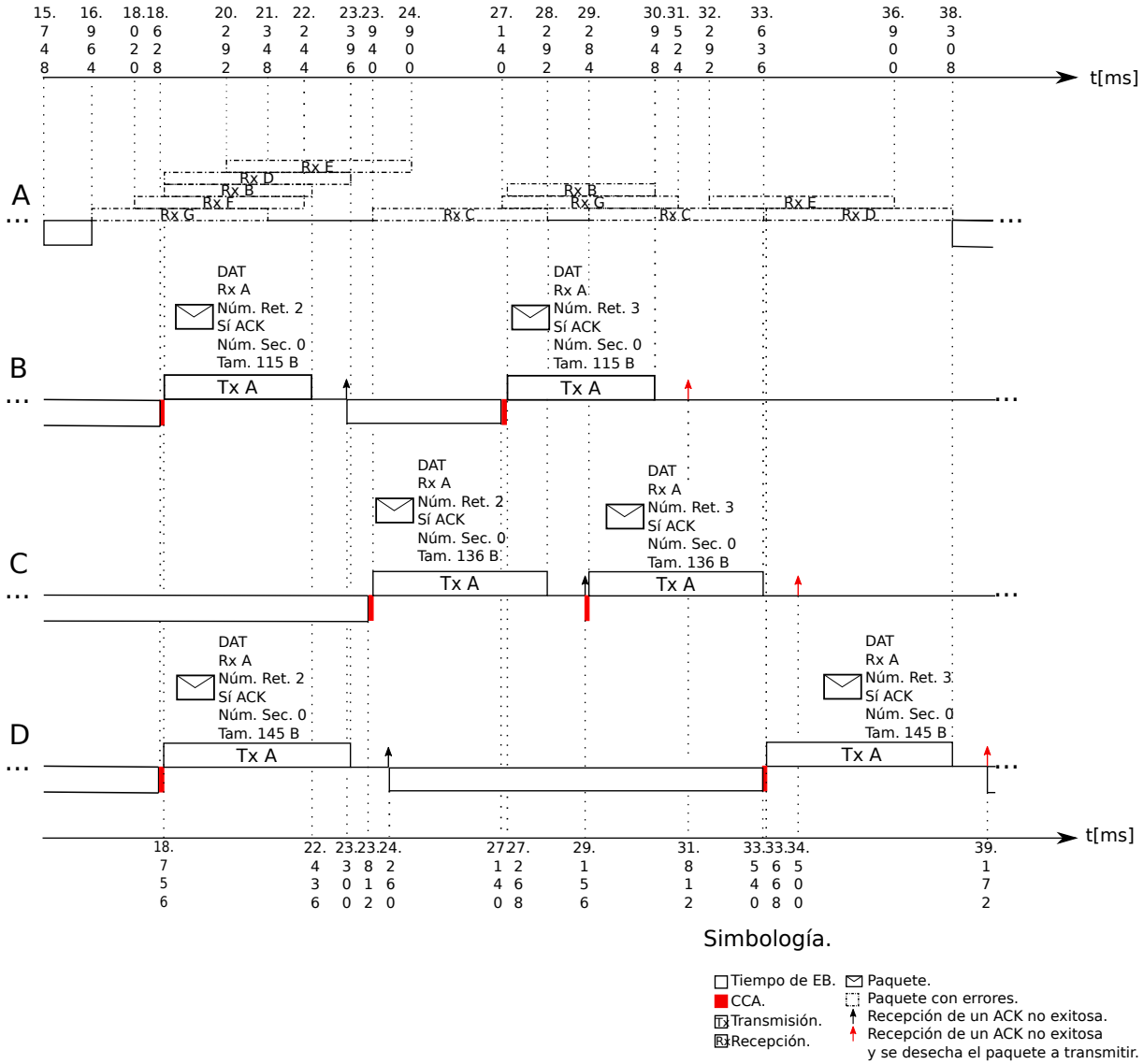



Figura 5.13: Representación de los datos obtenidos en la terminal de la simulación con una topología tipo estrella (segunda parte).

### 5.2.3. Animación

A través de los datos extraídos del archivo mostrado en la figura 5.8, se obtuvo el *frame* de la imagen 5.14 que muestra el momento en el que el nodo *A* recibe tramas de datos por parte de todos los nodos a excepción del nodo *C*. Específicamente, este evento tuvo lugar en el periodo de tiempo que abarca desde los $20.292\,\mathrm{ms}$ hasta los $21.348\,\mathrm{ms}$.

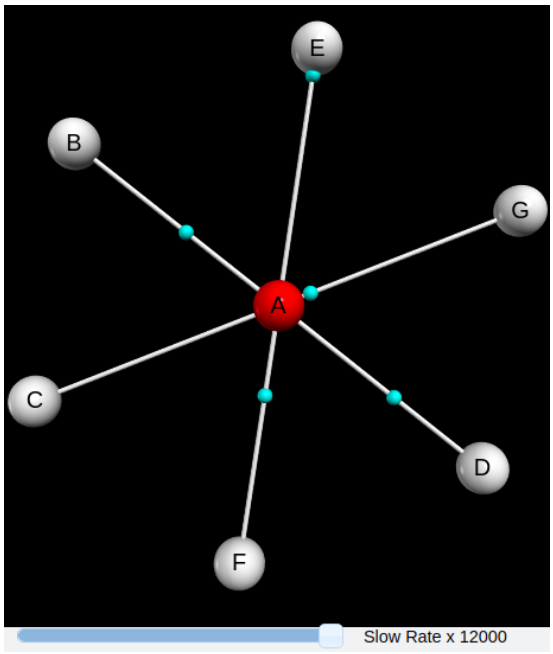



Figura 5.14: Animación de la simulación con una topología tipo estrella.

## 5.3. Prueba con una topología tipo árbol.

La última prueba consta de una red con topología tipo árbol, conformada por doce nodos, de los cuales tres son dispositivos *FDD*, ocho son dispositivos *RFD* y solo uno es el coordinador PAN, como se muestra en la figura 5.15. El nodo *A* se encuentra ubicado en $(0, 0)$, el nodo *B* está posicionado en $(-1, 0.5)$, el nodo *C* se localiza en $(-0.5, 2)$, el nodo *D* se coloco en $(2, -0.5)$, el nodo *E* se ubica en $(-1.5, -1.5)$, el nodo *F* está localizado en $(-2.5, -2)$, el nodo *G* se posicionó en $(-2, -2.5)$, el nodo *H* se encuentra ubicado en $(-1, 2.5)$, el nodo *I* está posicionado en $(0.5, 2.5)$, el nodo *J* se localiza en $(2.5, 0.5)$, el nodo *K* se coloco en $(3, -1)$ y el nodo *L* se ubica en $(2.5, -1.5)$.

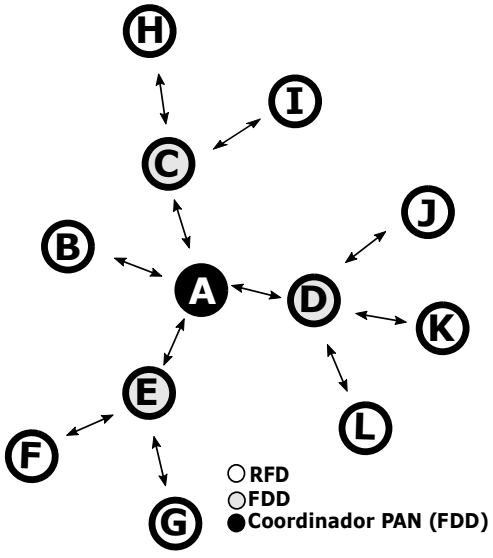



Figura 5.15: Topología tipo árbol.

### 5.3.1. Datos del archivo de texto

La información recabada en el archivo de texto se muestra en la figura 5.16. A través de estos datos se generaron los esquemas de tiempo mostradas en las figuras 5.17, 5.18 y 5.19.

Para este caso los esquemas de tiempo se dividieron en tres partes para facilitar el análisis de los procesos que se presentaron entre los dispositivos.

```
15 t    B    A    DAT    0    -      97    Comienza    0.000132
16 t    D    J    DAT    0    -      115   Comienza    0.000132
17 t    G    E    DAT    0    -      145   Comienza    0.000132
18 t    J    D    DAT    0    -      129   Comienza    0.000132
19 r    B    A    DAT    -    None   97    Comienza    0.000132
20 r    G    E    DAT    -    None   145   Comienza    0.000132
21 r    I    C    DAT    -    None   93    Comienza    0.000452
22 t    I    C    DAT    0    -      93    Comienza    0.000452
23 r    H    C    DAT    -    None   132   Comienza    0.000772
24 r    F    E    DAT    -    None   144   Comienza    0.000772
25 t    F    E    DAT    0    -      144   Comienza    0.000772
26 t    H    C    DAT    0    -      132   Comienza    0.000772
27 r    B    A    DAT    -    False  97    Termina     0.003236
28 t    B    A    DAT    0    -      97    Termina     0.003236
29 t    I    C    DAT    0    -      93    Termina     0.003428
30 t    A    B    ACK    0    -      11    Comienza    0.003428
31 r    A    B    ACK    -    True   11    Comienza    0.003428
32 dr   I    C    DAT    -    True   93    Termina     0.003428
33 t    A    B    ACK    0    -      11    Termina     0.00378
34 r    A    B    ACK    -    True   11    Termina     0.00378
35 t    D    J    DAT    0    -      115   Termina     0.003812
36 t    J    D    DAT    0    -      129   Termina     0.00426
37 r    A    B    DAT    -    None   129   Comienza    0.004612
38 t    A    B    DAT    0    -      129   Comienza    0.004612
39 dr   G    E    DAT    -    True   145   Termina     0.004772
40 t    G    E    DAT    0    -      145   Termina     0.004772
41 dr   H    C    DAT    -    True   132   Termina     0.004996
42 t    H    C    DAT    0    -      132   Termina     0.004996
43 dr   F    E    DAT    -    True   144   Termina     0.00538
44 t    F    E    DAT    0    -      144   Termina     0.00538
45 t    L    D    DAT    0    -      127   Comienza    0.005412
46 r    L    D    DAT    -    None   127   Comienza    0.005412
47 t    J    D    DAT    1    -      129   Comienza    0.005572
48 r    J    D    DAT    -    None   129   Comienza    0.005572
49 t    K    D    DAT    0    -      114   Comienza    0.005732
50 r    K    D    DAT    -    None   114   Comienza    0.005732
51 r    A    B    DAT    -    False  129   Termina     0.00874
```

Figura 5.16: Ejemplo de datos en el archivo de texto de la simulación con una topología tipo árbol.

En el esquema de la figura 5.17 se muestran los procesos que se presentaron en el periodo de tiempo que abarca desde el inicio de la simulación hasta los $10.340\,\mathrm{ms}$. Se puede observar que interactuaron los nodos *G*, *E* y *F*, se transmitieron un total de $4$ paquetes, de los cuales, solo $2$ fueron recibidos correctamente y los restantes colisionaron, el nodo *G* transmitió $2$ tramas al nodo *E*, el nodo *F* transmitió $1$ trama al nodo *E* y el nodo *E* transmitió $1$ trama al nodo *G*. En esta parte de los resultados no se presentaron retransmisiones.

En este experimento también se puede comprobar la presencia de trasmisiones simultáneas que terminan en colisión. Por ejemplo, el evento entre los $0.132\,\mathrm{ms}$ y los $5.380\,\mathrm{ms}$ en la línea de tiempo del nodo *E*, en donde se muestra la colisión de las tramas transmitidas por los nodos *F* y *G*.

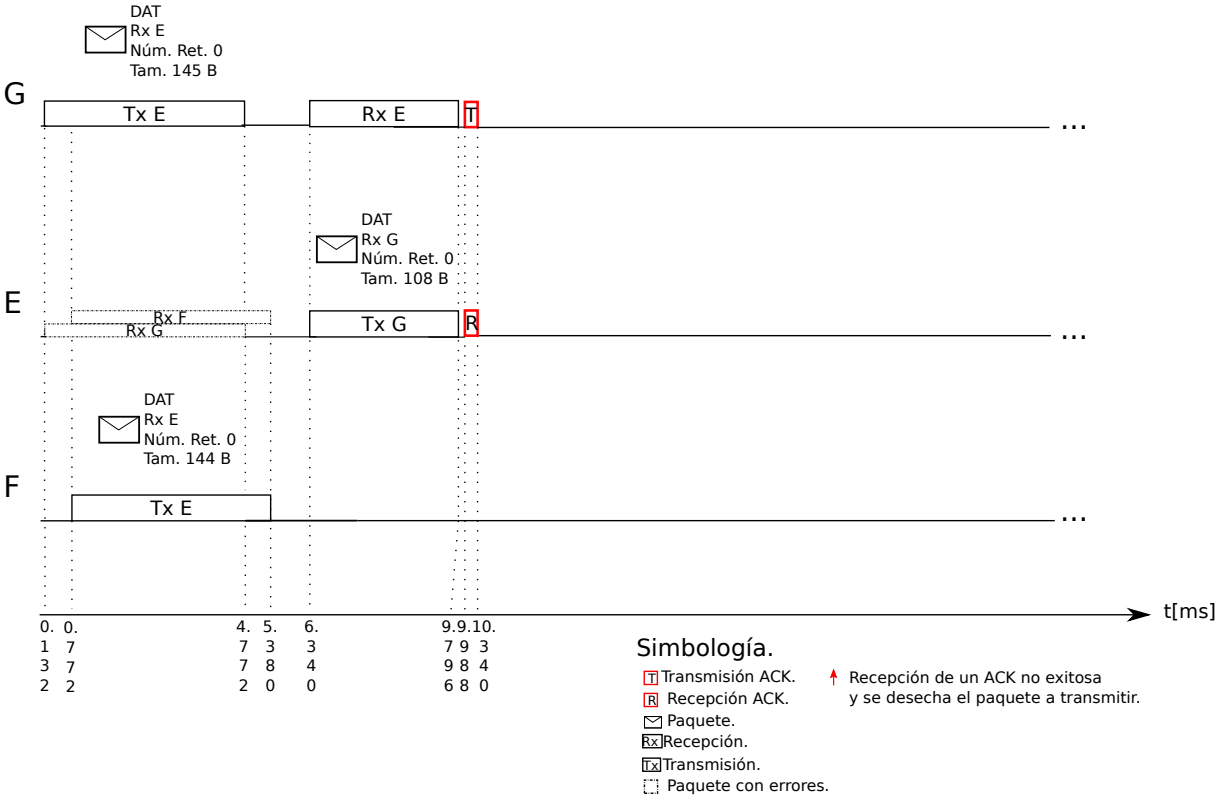



Figura 5.17: Representación de los datos obtenidos en el archivo de texto de la simulación con una topología tipo árbol (primera parte).

La figura 5.18 muestran los procesos que se presentaron en el periodo de tiempo que abarca desde el inicio de la simulación hasta los $23.428\,\mathrm{ms}$. Se puede observar que interactuaron los nodos *K*, *J*, *D* y *L*. Se transmitió un total de $5$ paquetes, de los cuales, ninguno fue recibido correctamente y todos colisionaron, existieron un total de $4$ retransmisiones, de las cuales solo una fue exitosa y las demás colisionaron, el nodo *K* transmitió $2$ tramas al nodo *D*, el nodo *J* transmitió $1$ trama al nodo *D* y retransmitió el mismo paquete en $2$ ocasiones. El nodo *D* transmitió $1$ trama al nodo *J* y el nodo *L* transmitió $1$ trama al nodo *D* y retransmitió el mismo paquete $2$ veces.

En este esquema también se puede comprobar la presencia de trasmisiones simultáneas que terminan en colisión y dicho evento se ve ejemplificado entre los $5.412\,\mathrm{ms}$ y los $9.700\,\mathrm{ms}$ en la línea de tiempo del nodo *D*. También en este tiempo se muestra la colisión entre los paquetes de los nodos *K*, *J* y *L*.

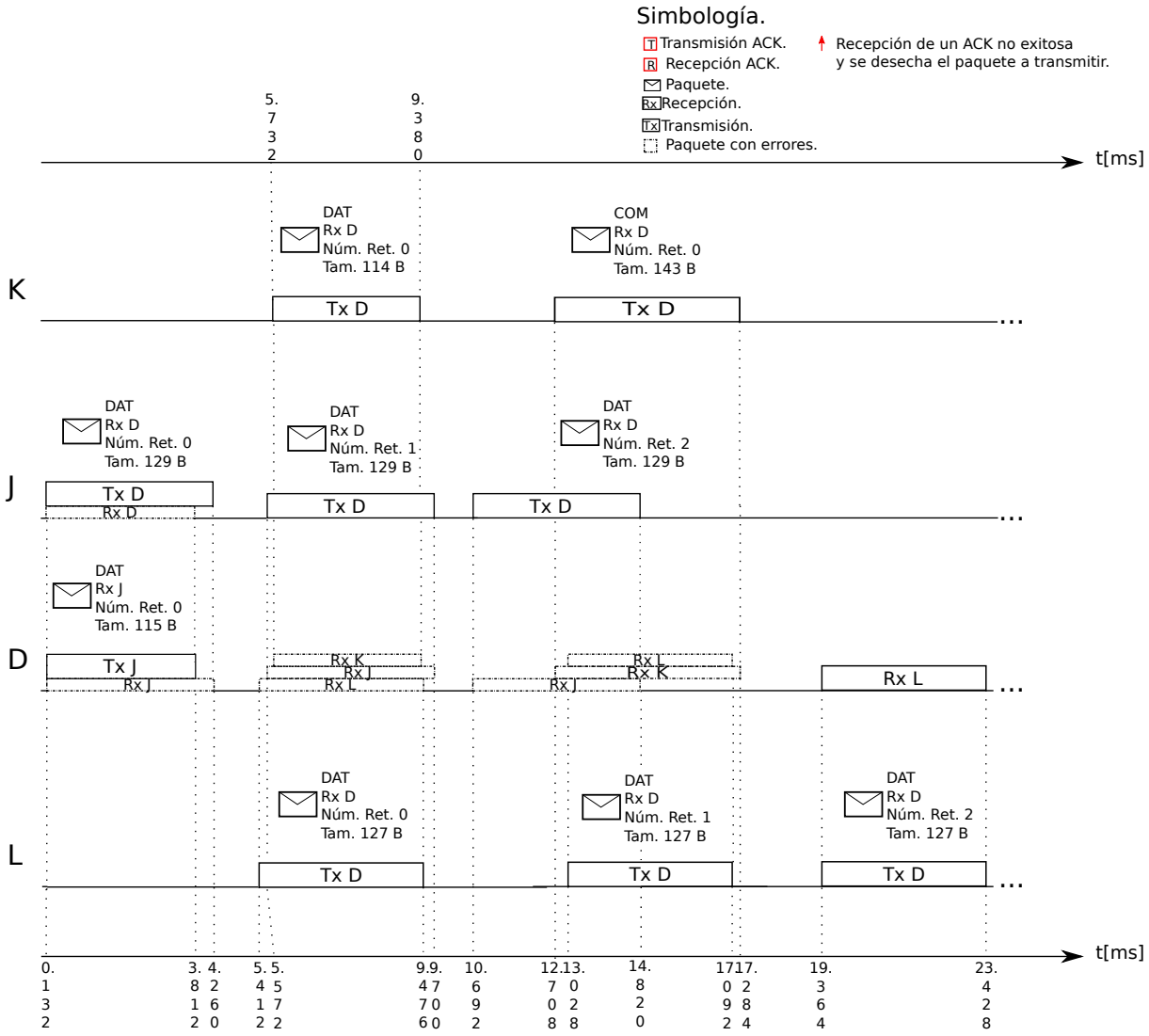



Figura 5.18: Representación de los datos obtenidos en el archivo de texto de la simulación con una topología tipo árbol (segunda parte).

En el esquema de la figura 5.19 se pueden observar los procesos que se presentaron en el periodo de tiempo que abarca desde el inicio de la simulación hasta los $18.020\,\mathrm{ms}$, se puede observar que interactuaron los nodos *A*, *B*, *C*, *I* y *H*. Se transmitieron $7$ paquetes, de los cuales, $2$ fueron recibidos correctamente y todos los demás llegaron al receptor con errores, existieron un total de $2$ retransmisiones, de las cuales solo una fue exitosa y las demás colisionaron, el nodo *B* transmitió $1$ trama al nodo *A* y retransmitió el mismo paquete en $2$ ocasiones. El nodo *A* transmitió $3$ tramas al nodo *B*. El nodo *C* transmitió $1$ trama al nodo *A*, el nodo *I* transmitió $1$ trama al nodo *C* y el nodo *H* transmitió $1$ trama al nodo *C*.

En este esquema también se puede verificar las trasmisiones simultáneas que terminan en colisión, un ejemplo se presenta entre los $0.452\,\mathrm{ms}$ y los $4.996\,\mathrm{ms}$ en la línea de tiempo del nodo *C*. En este mismo tiempo se muestra la colisión entre las tramas de los nodos *I* y *H*.

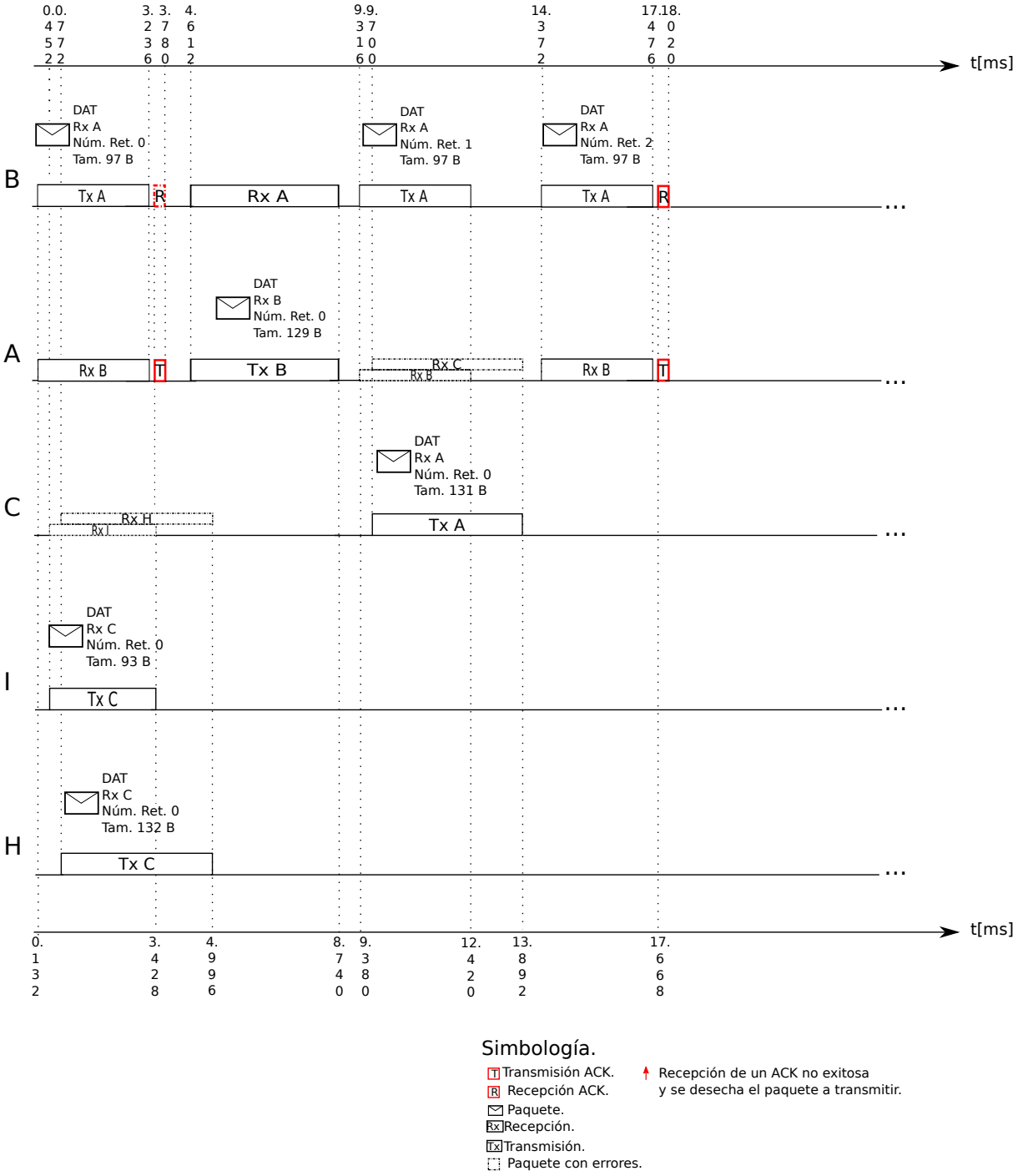



Figura 5.19: Representación de los datos obtenidos en el archivo de texto de la simulación con una topología tipo árbol (tercera parte).

### 5.3.2. Datos de la terminal

Con los datos que se muestran en la terminal se crearon los esquemas de tiempo de las figuras 5.21, 5.22 y 5.23. Estos esquemas muestran los mismos datos que los esquemas creados con la información del archivo de texto, pero también se muestran eventos como los ya mencionados en las dos pruebas anteriores, los cuales complementan el análisis. En la figura 5.20 se puede observar un ejemplo de los datos que se obtuvieron en la terminal, durante la simulación.

```
Accion: LIFS                    Estatus: Termina
Nodo: D                         Tiempo: 0.004452
================================================================
Pulsa una tecla para continuar...
================================================================
Accion: EB                      Estatus: Termina
Nodo: A                         Tiempo EB:0.00064
EB:3                            Repeticiones del EB maximo:0
Tiempo: 0.004484
================================================================
Pulsa una tecla para continuar...
================================================================
Accion: CCA                     Estatus: Comienza
Nodo: A                         Estado del canal: ---
Tiempo: 0.004484
================================================================
Pulsa una tecla para continuar...
================================================================
Accion: CCA                     Estatus: Termina
Nodo: A                         Estado del canal: Libre
Tiempo: 0.004612
================================================================
Pulsa una tecla para continuar...
================================================================
Accion: EB                      Estatus: Se pausa
Nodo: B                         Tiempo EB:0.00096
EB:4                            Repeticiones del EB maximo:0
Tiempo: 0.004612
================================================================
Pulsa una tecla para continuar...
================================================================
Accion: Recepcion               Estatus: Comienza
Tipo de paquete: DAT            Requiera ACK: False
Nodo Rx: B                      Contiene errores: None
Nodo Tx del paquete: A          Colisiono: ---
Numero de secuencia:0           Paquetes Rx:0
Nodos Tx: ['A']
Tiempo: 0.004612
```

Figura 5.20: Ejemplo de datos en la terminal de la simulación con una topología tipo árbol.

En la figura 5.21 se presenta el acontecimiento en el que un paquete no requiere *ACK*, por lo tanto, aunque la transmisión falle, el nodo fuente la considera como correcta, ya que no tiene forma de saber que el proceso falló. Un ejemplo de esto se puede apreciar en la línea de tiempo del nodo *F*, ya que dicho nodo le transmite una trama de datos al nodo *E*, pero como el paquete no requiere *ACK*, pese a que colisiona con un paquete del nodo *G*, el nodo *F* continua con el proceso de LIFS en lugar de tratar de retransmitir el paquete.

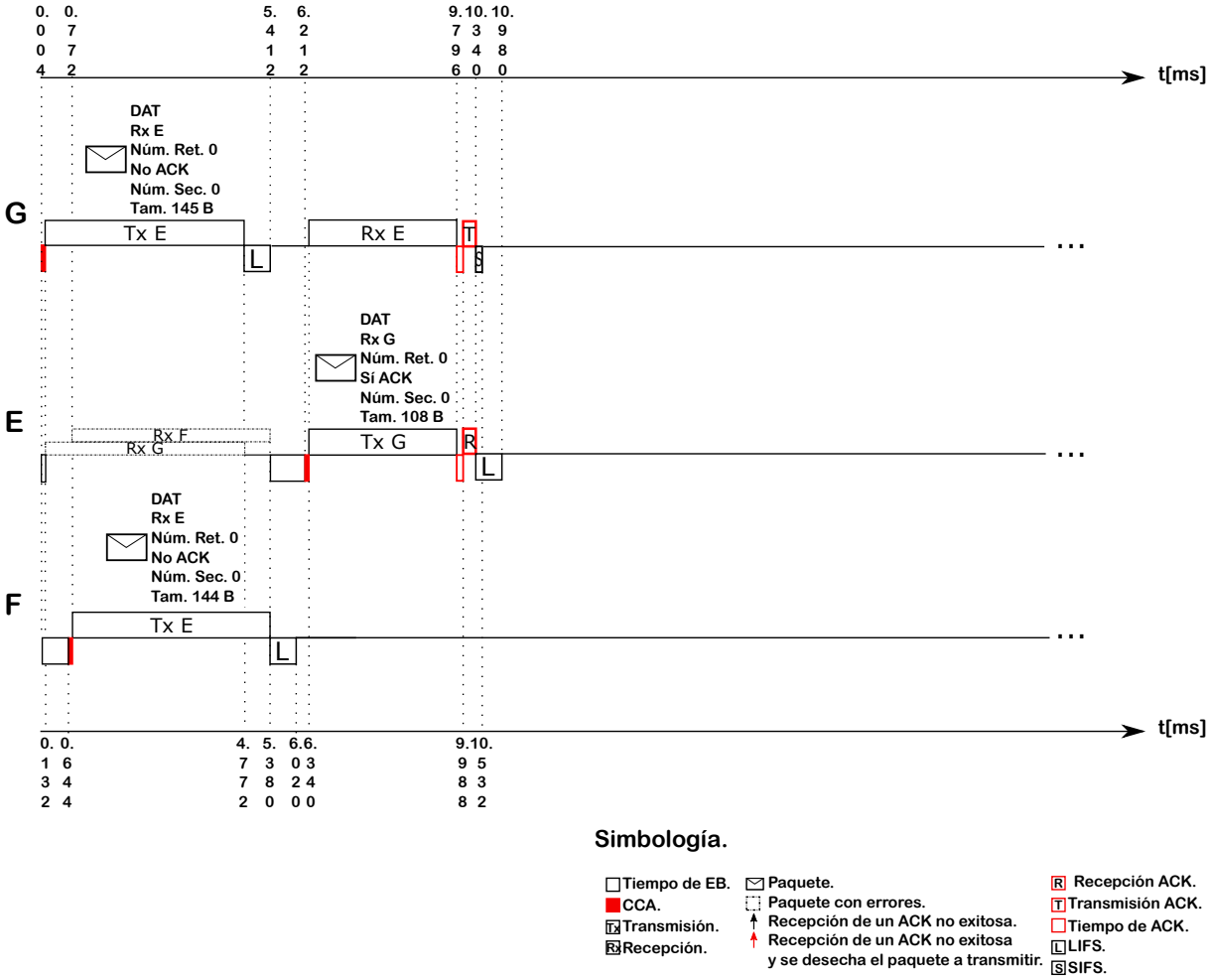
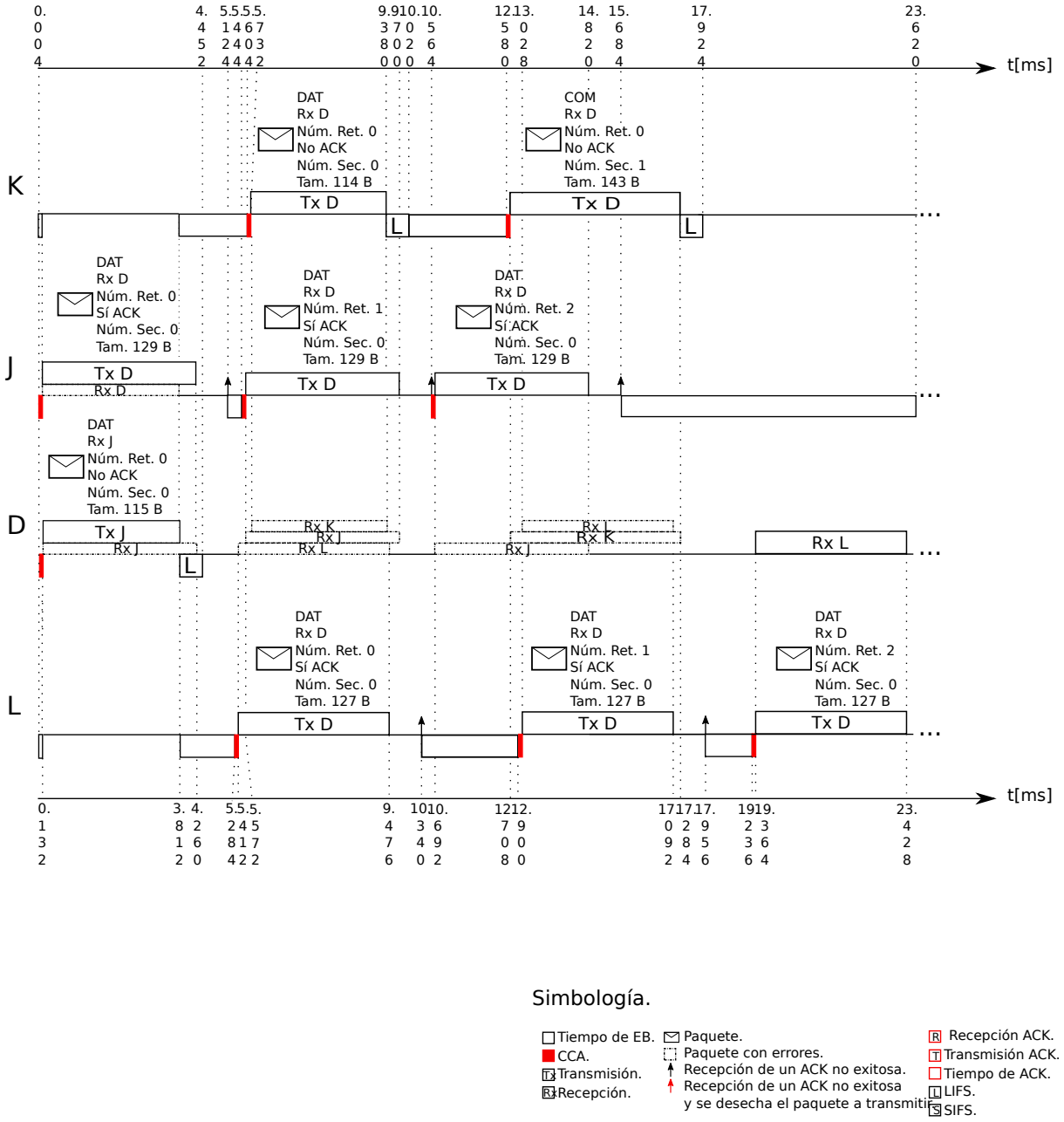



Figura 5.21: Representación de los datos obtenidos en la terminal de la simulación con una topología tipo árbol (primera parte).

En la figura 5.22 se puede observar que los nodos *D* y *J* esperaron el mismo tiempo de *exponential backoff*, que fue de $0\,\mathrm{s}$, y al realizar el proceso de *CCA*, ambos dispositivos detectaron el canal libre y transmitieron al mismo tiempo. La trama del nodo *D* iba dirigida al nodo *J* y viceversa, al presentarse este suceso, ninguno de los dos nodos recibió la trama y el único que realizó la acción de retransmisión fue el nodo *J*, debido a que el paquete procedente de este dispositivo requería un acuse de recepción, caso que no se presenta con el nodo *D*, el cual después de transmitir continuo con el proceso *LIFS*.

También se puede apreciar la recepción correcta de un paquete que se ha retransmitido, suceso que ocurre en la línea de tiempo del nodo *L*, ya que después de esperar la recepción de una trama *ACK* hasta $17.956\,\mathrm{ms}$, este dispositivo reinicia el proceso de transmisión, esperando un nuevo tiempo de *exponential backoff*, realizando el proceso de *clear channel assessment* y al detectar el canal desocupado retransmite por segunda ocasión una trama de datos al nodo *D* en $19.364\,\mathrm{ms}$, la cual es recibida correctamente y sin errores en $23.428\,\mathrm{ms}$.




Simbología.




Figura 5.22: Representación de los datos obtenidos en la terminal de la simulación con una topología tipo árbol (segunda parte).

En la figura 5.23 se logra observar que una trama puede ser recibida por el nodo destino con errores, sin que dicho paquete haya sufrido una colisión, y esto es debido a la probabilidad de error que se propuso en los parámetros iniciales de la simulación, este caso se presenta en la línea de tiempo del nodo *B*, el cual en $3.780\,\mathrm{ms}$ termina de recibir una trama *ACK* por parte del nodo *A*, sin embargo como dicho paquete trae errores, el nodo *B* la desecha y considera que el proceso de transmisión falla y reinicia el proceso en el instante $4.100\,\mathrm{ms}$.

También se puede apreciar que después de que un nodo transmite una trama corta, este en vez de esperar un *LIFS*, espera un *SIFS*, como se observa en la línea de tiempo del nodo *A* en $3.780\,\mathrm{ms}$, donde termina de transmitir un paquete *ACK* y de inmediato inicia el proceso *SIFS*.

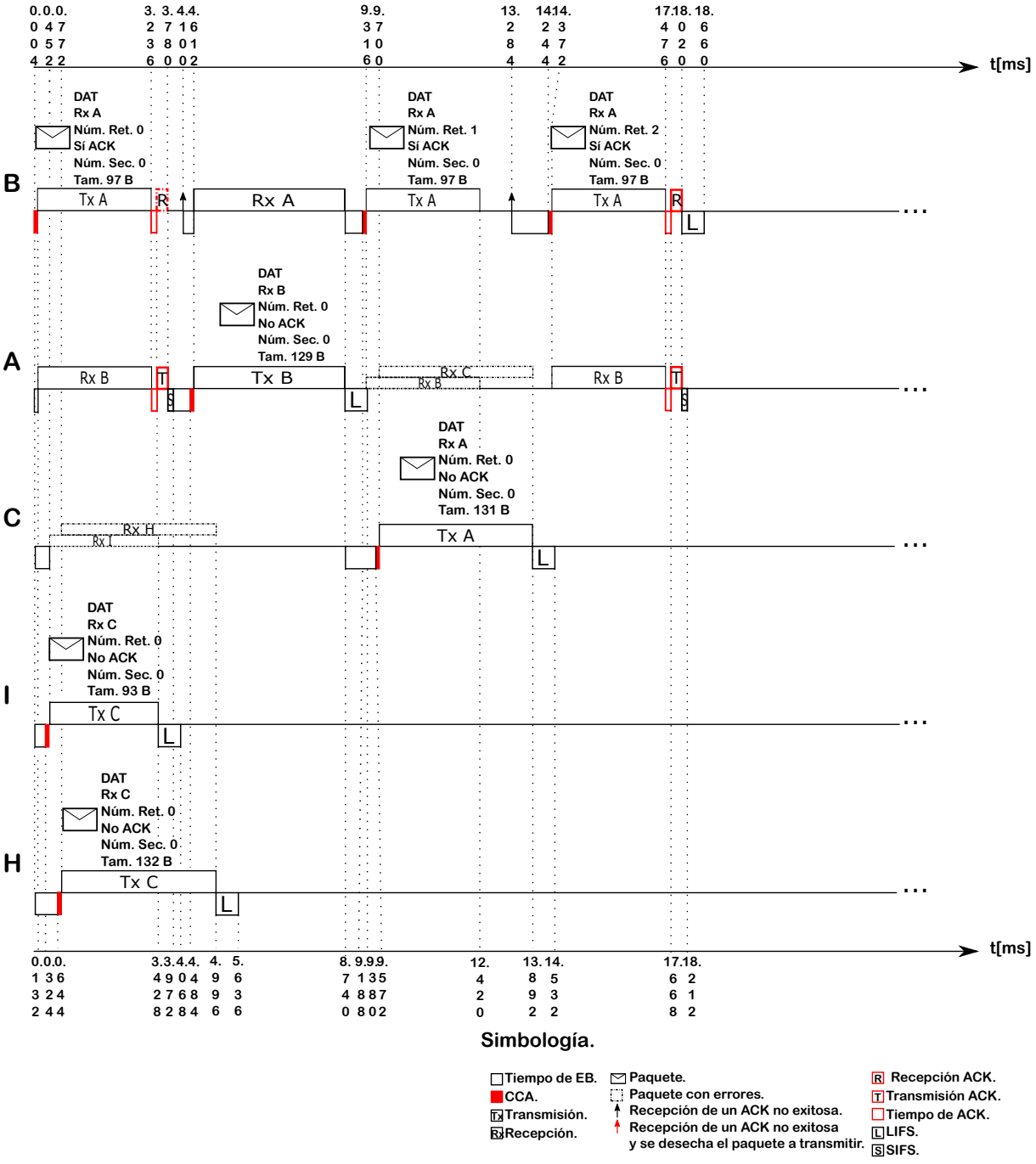



Figura 5.23: Representación de los datos obtenidos en la terminal de la simulación con una topología tipo árbol (tercera parte).

### 5.3.3. Animación

Al realizar la animación de los resultados obtenidos del archivo de texto, el cual se muestra en la figura 5.16, se obtuvo el *frame* de la figura 5.24, en el que se muestra cuando el nodo *A* se encuentra transmitiendo una trama *ACK* al nodo *B*. El nodo *C* se encuentra recibiendo una trama de datos por parte del nodo *H* y los nodos *F* y *G* transmiten al mismo tiempo al nodo *E*. Específicamente, este evento tuvo lugar en el periodo de tiempo que abarca desde los $3.428\,\text{ms}$ hasta los $3.780\,\text{ms}$.

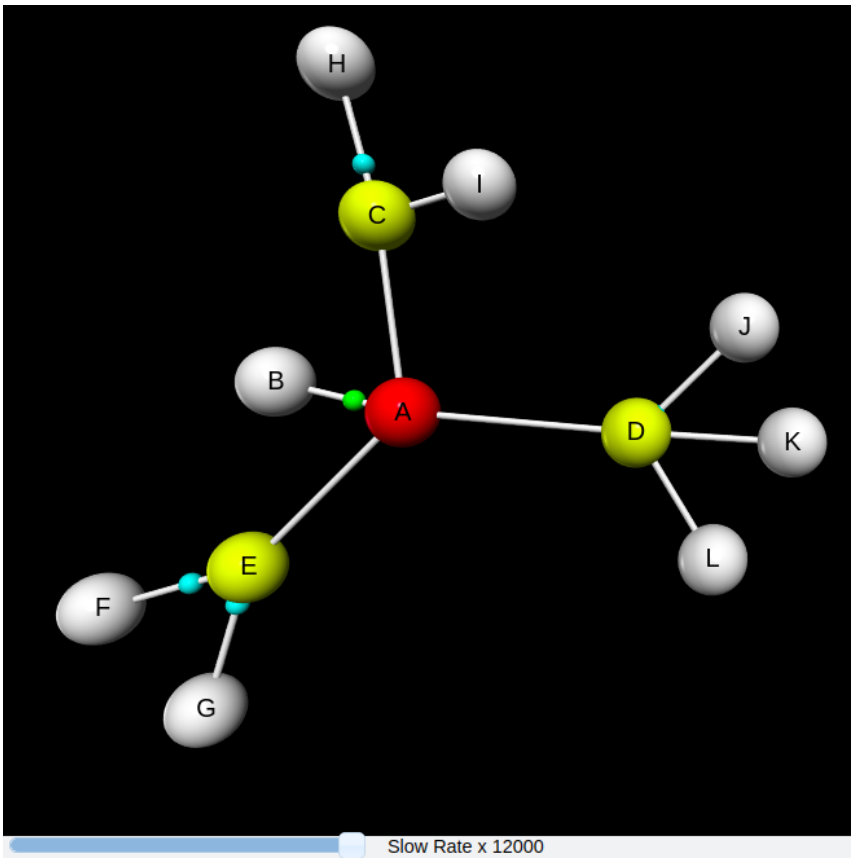



Figura 5.24: Animación de la simulación con una topología tipo árbol.

## 5.4. Consumo de memoria RAM

Se realizó una prueba con el fin de analizar la cantidad de recursos que utiliza el simulador, con respecto al uso de memoria RAM. Para ello, se dejo fijo los siguientes parámetros:

- Tiempo de simulación de 100 segundos.
- Se desactivaron las funciones de recolección de datos.
- El tiempo promedio de creación de paquetes se estableció en 1 segundo.
- La probabilidad de crear un paquete de comandos cada vez que se crea uno de datos se fijo en 50 %.
- La probabilidad de que un paquete contenga errores se estableció en 10 %.
- La probabilidad de que un paquete requiera acuse de recibido se fijó en 80 %.

Lo anterior para una red con 2, 5, 10, 50 y 100 nodos. Para realizar la medición del consumo de memoria se utilizó el *script ps_mem*, el cual está disponible en Internet y es una gran herramienta diseñada en Python [29]. La gráfica con los resultados obtenidos se puede observar en la figura 5.25.

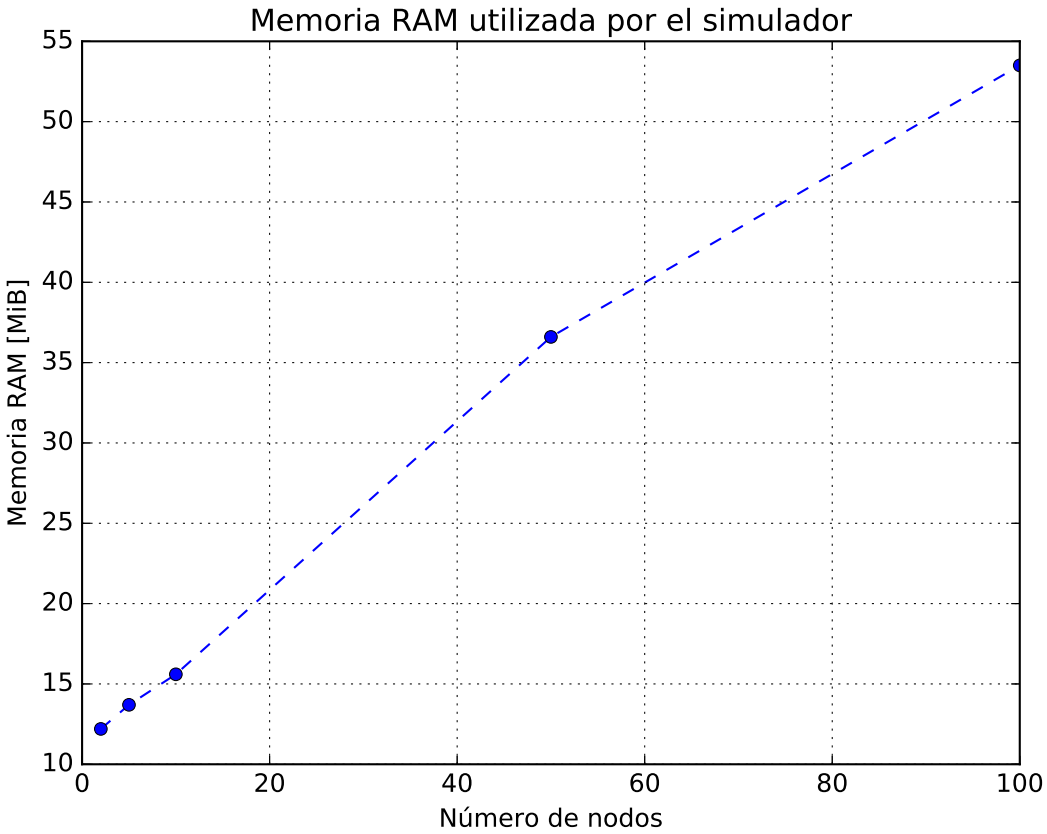



Figura 5.25: Cantidad de memoria RAM consumida por el simulador.

Los resultados de la figura 5.25 muestran que entre mayor sea la cantidad de nodos, mayor es la cantidad de memoria que consume el simulador. Dicho comportamiento se podría ver justificado debido a que Python es un lenguaje interpretado, esto significa que requiere de un interprete para ser ejecutado y por lo tanto añade una capa más de software. Sin embargo, los resultados muestran que la tendencia está por debajo de una tendencia lineal con respecto a la cantidad de nodos en la red.

# Capítulo 6

# Conclusiones

## 6.1. Conclusiones generales

El simulador diseñado en este trabajo, como se muestra en la sección 5.4, tiene un gasto de memoria por debajo de lo lineal con respecto al número de nodos. Sin embargo, tiene fortalezas como el ejecutarse en cualquier plataforma o sistema operativo que soporte Python sin la necesidad de compilarse o adaptarse a las bibliotecas propias del sistema operativo o arquitectura, ya que al ser implementado en el lenguaje Python nativo, lo hace extremadamente portable. Especialmente, hoy en día casi cualquier sistema operativo cuenta con Python instalado, además tiene ventajas como alta flexibilidad, ya que al estar programado en dicho lenguaje puede incorporar cualquier biblioteca y expandir sus funcionalidades de una manera muy simple. Más aún, el lenguaje Python suele ser fácil de aprender, leer y escribir, además de que es uno de los lenguajes de programación más utilizados a nivel mundial y cuenta con una gran comunidad activa, múltiples herramientas y bibliotecas, lo que permite abarcar muchas áreas de las ciencias e ingenierías.

Conforme a los resultados presentados en este trabajo, no se realizaron pruebas de uso del CPU, tiempo de procesamiento, escalabilidad, entre otras, con el fin de comparar el rendimiento del simulador diseñado en esta tesis con otros disponibles en el mercado, debido a que no serían concluyentes los resultados al contrastar un proyecto cuyo enfoque es la capa MAC del estándar IEEE 802.15.4 con otros que contienen todas las capas del modelo OSI. De igual forma, no se realizaron experimentos con factores como tasa de perdida de paquetes, retardo promedio, encaminamientos, etcétera, por la ausencia de las capas que se encargan de dichos parámetros. Sin embargo, para probar esta capa, se utilizaron funciones probabilísticas como entradas a la capa MAC provenientes de la capa física.

La principal aportación de esta tesis es la implementación de la capa MAC del estándar IEEE 802.15.4 en un simulador con ayuda de la biblioteca SimPy, además de comprobar su correcto funcionamiento a través de los escenarios vistos en el capitulo 5.

Durante el trabajo de tesis, se pudo observar que diseñar e implementar un simulador no es tarea fácil, ya que se tienen que considerar todos los escenarios que se puedan presentar durante la simulación y proponer soluciones que permitan el correcto funcionamiento de los procesos. Además de que siempre se tiene que considerar que una simulación es una aproximación de la realidad y por ende, la primera versión de un simulador nunca va a ser la definitiva y siempre va a estar sujeta a cambios como cualquier software en la actualidad. Finalmente, se espera que este trabajo sirva como base para otras tesis donde otros alumnos incluyan otros protocolos o estándares de comunicación.

## 6.2. Verificación de la hipótesis

Retomando la hipótesis que se presenta en la sección 1.2:

> "*El uso de la biblioteca Simpy como base de un DES permite ventajas en la construcción de un simulador orientado a WSN como flexibilidad, repetibilidad,*

*reusabilidad con una curva de aprendizaje menor a la de otros lenguajes de programación.*"

Al revisar la hipótesis, se puede concluir que *SimPy* es una herramienta muy útil para realizar simuladores de eventos discretos, ya que al ser una biblioteca de *Python*, esta puede complementarse con otros paquetes para mejorar los resultados, en el caso de este trabajo, se utilizaron bibliotecas como *vpython, random, collections, networkx,* entre otras, las cuales ayudaron a aportar una mayor flexibilidad para mejorar, facilitar y aumentar las funcionalidades del simulador. Un ejemplo de esto es el diseño e implementación de una animación. Sin embargo, *SimPy* podría aportar mayor flexibilidad si permitiera tener un dominio completo de los procesos, ya que en algunas situaciones se requiere otorgarles prioridad conforme al orden de ejecución, esta acción facilitaría la programación.

En los experimentos realizados en esta tesis, se prueba la repetibilidad de los resultados, ya que el simulador se ejecutó en diversas ocasiones, con los mismos parámetros iniciales, y siempre se obtuvieron las mismas salidas, esta característica se puede ver complementada con la ayuda de la biblioteca *random*, la cual permite reproducir los resultados cada vez que se ponga en funcionamiento el simulador, de igual manera, se puede comprobar la reusabilidad, debido a que se realizaron 3 experimentos para comprobar el correcto funcionamiento del simulador, con diferentes parámetros iniciales, y en todos estos se observó un correcto funcionamiento.

Por último, la curva de aprendizaje para poder utilizar y manipular este simulador, es menor a la de otros simuladores, debido a que está implementado en el lenguaje de programación *Python*, el cual es simplificado, flexible, portable, fácil de aprender, escribir, leer y cuenta con una gran comunidad.

## 6.3. Trabajo futuro

Como trabajo futuro, por parte de otros estudiantes interesados en el proyecto, se plantea la implementación de capa física con el fin de evaluar con más detalle los efectos de interferencia, multitrayectoria, pérdida de potencia, desvanecimientos profundos, entre otros. También se compararán las tres pruebas mostradas en el capítulo 5 con algún otro simulador como lo es NS3 para ver a más detalles los tiempos de procesamiento y los resultados de ambos simuladores.

# Apéndice A

# Clase Nodo

```
import random
import simpy
import time
from decimal import Decimal
import collections
from Pack import Pack
from printClass import printClass
from Canal import Canal

#Parametros establecidos en 802.15.4

class Node:

  def __init__(self, env, idPAN, coordinator, name, probabilities, time):
    self.Rsps = float(62500) #Modulation speed
    self.sym = 4 #A symbol has 4 bits
    self.Rbps = self.Rsps * self.sym ##Transmission speed
    self.tsym = 1/self.Rsps #Symbol time
    self.timeMaxToSleep = time #Generate a packet approximately every x seconds
    self.name = name #Node identifier
    self.env = env  #The simulation environment
    self.canal =None #transmission medium
    self.idPAN = idPAN #Identifier of the source network
    self.coordinator = coordinator #Boolean variable to know if a node is
    coordinator
    self.EBTime = 0 #Exponential Backoff time
    self.macMinBE = 3 #EB minimum value
    self.macMaxBE = 5 #EB maximum value
    self.BEexp = self.macMinBE #Exponential Backoff
    self.unitEBPeriod = 20*self.tsym #Exponential backoff period time
    self.macMaxCSMAEB =  4 #Maximum attempts to transmit a packet, after the value
    of macMaxBE
    self.auxBEexp = 0 #Auxiliary variable to count the number of repetitions of
    macMaxBE
    self.paqTirados = 0 #Number of packages discarded
    self.paqsent = 0 #Number of packages sent
    self.paqReceived = [] #List of received packages
    self.ACKReceived = None #Auxiliary variable to store an ACK
    self.paqReceivedCorrectly = 0 #Number of packages received correctly
    self.vecinos = None #Node neighbor list
    self.queue = []#List of created packages
    self.aTurnaroundTime = 12 * self.tsym #Time a node waits to send an ACK
    self.timeToWaitAnACK = 54 * self.tsym #Time a node waits to receive an ACK
    self.pack = None #Auxiliary variable to store the packet to be transmitted
    self.packAppend = None #Auxiliary variable to store the created package
```

```
43      self.command = None #Command list
        self.txProcess = None #Auxiliary variable where the transmission process is
        stored
45      self.aMinSIFSPeriod = 12*self.tsym #Time between short frames
        self.aMaxSIFSFrameSize = 18 #Maximum number of bytes in a short frame
47      self.aMinLIFSPeriod = 40*self.tsym #Time between long frames
        self.CCAtime = 8*self.tsym #CCA time
49      self.probTC = probabilities[0] #Probability of creating a command package every
        time a data package is created
        self.flagRx = False #Reception flag
51      self.flagTx = False #Transmission flag
        self.packACK = None #Auxiliary variable to generate ACK packages
53      self.packACKTx = None #Auxiliary variable to store the ACK packets to be
        transmitted
        self.rxProcess = None #Auxiliary variable where the reception process is stored
55      self.packRx = {} #Dictionary of packages received at the same time
        self.flagTO = False #Flag indicating a collision
57      self.flagIFS = False #Flag indicating whether the IFS timeout has occurred
        self.error = 0.000001 #Error constant
59      self.printObject = None #Flag to enable or disable console printing
        self.probError = probabilities[1] #Probability that a package contains errors,
        without considering collisions
61      self.ACKrequestProcess = None #Auxiliary variable where the ACK reception
        process is stored
        self.flagBE = False #Flag to drop a packet that reached macMaxCSMAEB
63      self.txProcessACK = None #Auxiliary variable where the ACK transmission process
        is stored
        self.EBprocess = None #Auxiliary variable where the Exponential Bacoff process
        is stored
65      self.IFSprocess = None #Auxiliary variable where the IFS process is stored
        self.flagACKTx = False #Flag to indicate an ACK was transmitted
67      self.flagACKRx = False #Flag to indicate an ACK was received
        self.flagRxComplete = False #Flag to indicate that a package was received
69      self.processAuxACKRx = None #Auxiliary variable where the auxiliary process of
        receiving an ACK is stored
        self.probACK = probabilities[2] #Probability that a package requires ACK
71
    def get_index(self, probability):
73      c_probability = 0
        sum_probability = []
75      for p in probability:
          c_probability += p
77        sum_probability.append(c_probability)
        r = random.random()
79      for index, sp in enumerate(sum_probability):
          if r <= sp:
81          return index
        return len(probability)-1
83
    def probability(self, P):
85      response = [True, False]
        probability = [P, 1-P]
87      resp_index = self.get_index(probability)
        return response[resp_index]
89
    def exponentialBackOff(self):
91      if self.BEexp < self.macMaxBE + 1:
          return random.randint(0, pow(2, self.BEexp))
93      else:
          self.auxBEexp += 1
95        if self.auxBEexp >= self.macMaxCSMAEB + 1:
```

```
        self.paqTirados += 1
97      self.BEexp = self.macMinBE
        self.auxBEexp = 0
99      self.flagBE = True
        return random.randint(0, pow(2, self.BEexp))
101   return random.randint(0, pow(2, self.macMaxBE))

103 def getEB (self):
    self.EBTime = self.exponentialBackOff() * self.unitEBPeriod
105
    def unsuccessfulTx (self, pack):
107   self.BEexp += 1
      self.getEB()
109   if self.auxBEexp <= self.macMaxCSMAEB + 1 and pack.auxFrameRetries < pack.
      aMaxFrameRetries +1 and self.flagBE == False:
        self.queue.insert(0, pack)
111   else:
        self.printObject.printTx(pack, self.env.now, 'Se tira', self.paqsent, self.
      paqTirados, '10', '5')
113     self.printObject.printTxt(pack, self.env.now, 'dt', pack.node.name, pack.
      source, pack.totalLen, '-', '-', self.pack.auxFrameRetries)
        self.successfulTx()
115     self.flagBE = False

117 def successfulTx(self):
      self.auxBEexp = 0
119   self.BEexp = self.macMinBE
      self.getEB()
121
    def makePack (self, typeF, secuence):
123   node = random.choice(self.vecinos)
      typeF = typeF
125   self.packAppend = Pack(typeF, secuence, node, self.name, self.idPAN)
      self.packAppend.ACKrequest = self.probability(self.probACK)
127   self.packAppend.setupFrame(0, self.command)
      self.printObject.printPack(self.packAppend, self.env.now)
129   self.printObject.printTxt(self.packAppend, self.env.now, 'c', self.packAppend.
      node.name, self.packAppend.source, self.packAppend.totalLen, '-', '-', '-')
      self.queue.append(self.packAppend)
131
    def putQueue (self):
133   yield self.env.timeout(self.tsym/self.sym)
      secuence = 0
135   while True:
        self.makePack ("DAT", secuence)
137     secuence += 1
        if self.probability(self.probTC):
139       self.makePack ("COM", secuence)
          secuence += 1
141     yield self.env.timeout(random.expovariate(1.0/self.timeMaxToSleep))

143 def getAllindexOfPacks(self, list, nameDest):
      return filter(lambda a: list[a].node.name == nameDest, range(0,len(list)))
145
    def getAllindexOfPacksRx(self, list):
147   return filter(lambda a: list[a] == True, range(0,len(list)))

149 def getAllindexOfNodes(self, list, source):
      return filter(lambda a: list[a].name == source, range(0,len(list)))
151
    def ackTx(self, pack, time):
```

```
153     try:
          typeF = "ACK"
155       nodeIndex = self.getAllindexOfNodes(self.vecinos, pack.source)
          node = self.vecinos[nodeIndex[0]]
157       self.packACK = Pack(typeF, pack.secuence, node, self.name, self.idPAN)
          self.packACK.ACKtime = time
159       timeAux = self.env.now - self.packACK.ACKtime
          if timeAux <= self.unitEBPeriod:
161         if  timeAux <= self.aTurnaroundTime:
              self.printObject.printTxACK(self.packACK, self.env.now, 'Esperando tiempo
    de ACK', 'No ha caducado', self.paqsent, self.paqTirados, timeAux)
163           yield self.env.timeout(self.aTurnaroundTime - timeAux - self.error)
            if self.flagTx == True or self.flagRx == True:
165           yield self.env.timeout(self.error)
              while True:
167             if self.flagTx == True or self.flagRx == True:
                  yield self.env.timeout(self.tsym/self.sym)
169             else:
                  timeAux = self.env.now - self.packACK.ACKtime
171               if timeAux > self.unitEBPeriod:
                    self.printObject.printTxACK(self.pack, self.env.now - self.error, '
    ---', 'Ha caducado', self.paqsent, self.paqTirados, '---')
173                 else:
                      self.packACK.setupFrame(self.env.now, self.command)
175                   self.queue.insert(0, self.packACK)
                      self.pack = self.queue.pop(0)
177                   self.txProcess = self.env.process(self.tx())
                    self.flagACKTx = False
179                 self.txProcessACK = None
                    break
181         else:
              self.flagACKTx = False
183           self.txProcessACK = None
              self.packACK.setupFrame(self.env.now + self.error, self.command)
185           self.queue.insert(0, self.packACK)
              self.pack = self.queue.pop(0)
187           self.txProcess = self.env.process(self.tx())
        else:
189       self.printObject.printTxACK(self.pack, self.env.now, '---', 'Ha caducado',
    self.paqsent, self.paqTirados, '---')
          self.flagACKTx = False
191       self.txProcessACK = None
      except simpy.Interrupt as i:
193     self.printObject.printTxACK(self.pack, self.env.now, '---', 'Ha caducado',
    self.paqsent, self.paqTirados, '---')
        self.flagACKTx = False
195     self.txProcessACK = None

197   def FCScalculate(self, pack):
        if self.probability(self.probError):
199       pack.FCSbool = True
        else:
201       pack.FCSbool = False

203   def rx(self):
        while True:
205       if self.flagTx == True:
            yield self.env.timeout(self.tsym/self.sym)
207         continue
          else:
209         packSource = self.getAllindexOfPacks(self.canal.packT, self.name)
```

```
        if packSource:
211       for j in range(0,len(packSource)):
            i = packSource[0]
213         pack = self.canal.packT.pop(i)
            if pack.txTime + (self.tsym/self.sym) + self.error > self.env.now:
215           self.packRx[pack] = 0
              lista = [o.source for o in self.packRx.keys()]
217           if pack.typeF == 'ACK':
                self.printObject.printRxACK(pack, self.env.now, 'Comienza', pack.
    source, pack.node.name, '---')
219             self.printObject.printTxt(pack, self.env.now, 'r', pack.node.name,
    pack.source, pack.totalLen, 'Comienza', pack.FCSbool, '-')
              else:
221             self.printObject.printRx(pack, self.env.now, lista, 'Comienza', '---
    ', 'None')
                self.printObject.printTxt(pack, self.env.now, 'r', pack.node.name,
    pack.source, pack.totalLen, 'Comienza', pack.FCSbool, '-')
223           if self.flagRx == True or len(packSource) > 1 or self.flagTO == True:
                self.printObject.printCol(pack.source, self.env.now, self.name)
225             self.flagTO = True
              packSource = self.getAllindexOfPacks(self.canal.packT, self.name)
227           self.flagRx = True
            else:
229           continue
        if len(self.packRx) != 0:
231       if self.flagRxComplete:
            self.flagRxComplete = False
233       else:
            yield self.env.timeout(self.tsym/self.sym)
235       self.packRx = {key: self.packRx[key] + self.tsym/self.sym for key in self.
    packRx}
          fullyRxPack = {key: self.packRx[key] + self.error >= (key.totalLen*2*self.
    tsym  - self.tsym/self.sym) for key in self.packRx}
237       indexOffullyRxPack = self.getAllindexOfPacksRx(fullyRxPack.values())
          if indexOffullyRxPack :
239         self.flagRxComplete = True
            yield self.env.timeout(self.tsym/self.sym - self.error)
241         for i in indexOffullyRxPack:
              if self.flagTO == False and fullyRxPack.keys()[i].FCSbool == None:
243             self.FCScalculate(fullyRxPack.keys()[i])
              else:
245             fullyRxPack.keys()[i].FCSbool = True
            if len(self.packRx) == 1 and self.flagTO == False and fullyRxPack.keys()
    [indexOffullyRxPack[0]].FCSbool == False:
247           pack = fullyRxPack.keys()[indexOffullyRxPack[0]]
              self.paqReceived.append(pack)
249           self.paqReceivedCorrectly += 1
              self.flagRx = False
251           self.packRx.pop(pack)
              lista = [o.source for o in self.packRx.keys()]
253           if pack.typeF == 'ACK':
                self.printObject.printRxACK(pack, self.env.now + self.error, '
    Termina', pack.source, pack.node.name, pack.FCSbool)
255             self.printObject.printTxt(pack, self.env.now + self.error, 'r', pack
    .node.name, pack.source, pack.totalLen, 'Termina', pack.FCSbool, '-')
              else:
257             self.printObject.printRx(pack, self.env.now + self.error, lista, '
    Termina', 'False', pack.FCSbool)
                self.printObject.printTxt(pack, self.env.now + self.error, 'r', pack
    .node.name, pack.source, pack.totalLen, 'Termina', pack.FCSbool, '-')
259           if pack.typeF == "ACK" and pack.secuence == self.pack.secuence:
```

```
                yield self.env.timeout(self.error)
261             if self.ACKrequestProcess.processed:
                  self.printObject.printRxACK(pack, self.env.now, 'Ha caducado',
    pack.source, pack.node.name, '---')
263             else:
                  self.ACKrequestProcess.interrupt()
265               pack.ACKanswer = True
              elif (pack.typeF == "DAT" or pack.typeF == "COM") and pack.ACKrequest
    == True:
267             self.flagACKTx = True
                yield self.env.timeout(self.error)
269             time = self.env.now
                while True:
271               if self.flagTx == True or self.flagRx == True:
                    yield self.env.timeout(self.tsym/self.sym)
273               else:
                    self.txProcessACK = self.env.process(self.ackTx(pack, time))
275                 break
              else:
277             yield self.env.timeout(self.error)
                continue
279         else:
              yield self.env.timeout(self.error)
281           if len(self.packRx) == 1 and self.flagTO == False and fullyRxPack.keys
    ()[indexOffullyRxPack[0]].FCSbool == True:
                pack = fullyRxPack.keys()[indexOffullyRxPack[0]]
283             self.packRx.pop(pack)
                lista = [o.source for o in self.packRx.keys()]
285             if pack.typeF == 'ACK':
                  self.printObject.printRxACK(pack, self.env.now, 'Termina', pack.
    source, pack.node.name, pack.FCSbool)
287               self.printObject.printTxt(pack, self.env.now, 'r', pack.node.name,
     pack.source, pack.totalLen, 'Termina', pack.FCSbool, '-')
                else:
289               self.printObject.printRx(pack, self.env.now, lista, 'Termina', '
    False', pack.FCSbool)
                  self.printObject.printTxt(pack, self.env.now, 'r', pack.node.name,
     pack.source, pack.totalLen, 'Termina', pack.FCSbool, '-')
291           else:
                for i in indexOffullyRxPack:
293               pack = fullyRxPack.keys()[i]
                  self.packRx.pop(pack)
295               lista = [o.source for o in self.packRx.keys()]
                  if pack.typeF == 'ACK':
297                 self.printObject.printRxACK(pack, self.env.now, 'Termino y
    colisiono', pack.source, pack.node.name, 'True')
                    self.printObject.printTxt(pack, self.env.now, 'dr', pack.node.
    name, pack.source, pack.totalLen, 'Termina', pack.FCSbool, '-')
299                 else:
                      self.printObject.printRx(pack, self.env.now, lista, 'Termina', '
    True', pack.FCSbool)
301                   self.printObject.printTxt(pack, self.env.now, 'dr', pack.node.
    name, pack.source, pack.totalLen, 'Termina', pack.FCSbool, '-')
              if len(self.packRx) == 0 :
303             self.flagRx = False
                self.flagTO = False
305       else:
            self.flagRxComplete = False
307         yield self.env.timeout(self.tsym/self.sym)

309   def IFS (self):
```

```
    if self.pack.totalLen < self.aMaxSIFSFrameSize:
311   self.printObject.printIFS(self.name, self.env.now, 'Comienza', 'SIFS')
      yield self.env.timeout(self.aMinSIFSPeriod)
313   self.printObject.printIFS(self.name, self.env.now, 'Termina', 'SIFS')
    else:
315   self.printObject.printIFS(self.name, self.env.now, 'Comienza', 'LIFS')
      yield self.env.timeout(self.aMinLIFSPeriod)
317   self.printObject.printIFS(self.name, self.env.now, 'Termina', 'LIFS')

319 def auxIFS(self):
    self.IFSprocess = self.env.process(self.IFS())
321 yield self.IFSprocess
    self.flagIFS = False
323 if self.pack.typeF != "ACK":
      self.env.process(self.CSMA_CA_UNS())
325
    def ackRx(self):
327   while True:
        try:
329       time = self.env.now
          if (time - self.pack.txTimeStop) + self.error < self.timeToWaitAnACK and
    self.pack.auxFrameRetries <= self.pack.aMaxFrameRetries:
331         yield self.env.timeout(self.tsym/self.sym)
          else:
333         self.pack.retransmitionPack()
            if self.pack.auxFrameRetries > self.pack.aMaxFrameRetries:
335           self.paqTirados += 1
              self.processAuxACKRx.interrupt('MaxRetries rebasado')
337           self.printObject.printRxACK(self.pack, self.env.now, "Recepcion fallida"
    , self.pack.node.name, self.pack.source, '---')
              self.printObject.printTx(self.pack, self.env.now, 'Se tira', self.
    paqsent, self.paqTirados, self.BEexp, self.auxBEexp)
339           self.printObject.printTxt(self.pack, self.env.now, 'dt', self.pack.node.
    name, self.pack.source, self.pack.totalLen, '-', '-', self.pack.auxFrameRetries)
              self.successfulTx()
341           break
            self.unsuccessfulTx (self.pack)
343         self.processAuxACKRx.interrupt('ACKNotRx')
            self.printObject.printRxACK(self.pack, self.env.now, "Recepcion fallida",
    self.pack.node.name, self.pack.source, '---')
345       break
        except simpy.Interrupt as i:
347       break

349 def auxACKRx(self):
    try:
351   self.flagACKRx = True
      self.ACKrequestProcess = self.env.process(self.ackRx())
353   yield self.ACKrequestProcess
      self.flagACKRx = False
355   if self.pack.typeF != "ACK":
        self.successfulTx()
357   self.flagIFS = True
      self.env.process(self.auxIFS())
359 except simpy.Interrupt as i:
      self.flagTx = False
361   self.flagACKRx = False
      self.env.process(self.CSMA_CA_UNS())
363
    def tx(self):
365   try:
```

```
        self.flagTx = True
367     with self.canal.wirelessMedium.request() as req:
          yield req
369       self.canal.nodesTx.append(self.name)
          yield self.env.process(self.canal.intoCanal(self.pack, self. vecinos))
371       if self.pack.typeF == 'ACK':
            self.printObject.printTxACK(self.pack, self.env.now, 'Comienza', 'No ha
    caducado', self.paqsent, self.paqTirados, self.pack.txTime - self.pack.ACKtime)
373           self.printObject.printTxt(self.pack, self.env.now, 't', self.pack.node.
    name, self.pack.source, self.pack.totalLen, 'Comienza', '-', self.pack.
    auxFrameRetries)
            yield self.env.timeout(self.pack.totalLen*2*self.tsym)
375       else:
            self.printObject.printTx(self.pack, self.env.now, 'Comienza', self.paqsent
    , self.paqTirados, self.BEexp, self.auxBEexp)
377           self.printObject.printTxt(self.pack, self.env.now, 't', self.pack.node.
    name, self.pack.source, self.pack.totalLen, 'Comienza', '-', self.pack.
    auxFrameRetries)
            yield self.env.timeout(self.pack.totalLen*2*self.tsym)
379       self.canal.wirelessMedium.release(req)
          self.flagTx = False
381       self.txProcess= None
          self.pack.txTimeStop = self.env.now
383       self.paqsent += 1
          self.canal.nodesTx.remove(self.name)
385       if self.pack.typeF == 'ACK':
            self.printObject.printTxACK(self.pack, self.env.now, 'Terminado', '---',
    self.paqsent, self.paqTirados, self.pack.txTime - self.pack.ACKtime)
387           self.printObject.printTxt(self.pack, self.env.now, 't', self.pack.node.
    name, self.pack.source, self.pack.totalLen, 'Termina', '-', self.pack.
    auxFrameRetries)
          else:
389           self.printObject.printTx(self.pack, self.env.now, 'Terminado', self.
    paqsent, self.paqTirados, self.BEexp, self.auxBEexp)
              self.printObject.printTxt(self.pack, self.env.now, 't', self.pack.node.
    name, self.pack.source, self.pack.totalLen, 'Termina', '-', self.pack.
    auxFrameRetries)
391       if self.pack.ACKrequest:
            self.processAuxACKRx = self.env.process(self.auxACKRx())
393       else:
            if self.pack.typeF != "ACK":
395           self.successfulTx()
            self.flagIFS = True
397         self.env.process(self.auxIFS())
      except simpy.Interrupt as i:
399     self.flagTx = False
        self.txProcess= None
401     self.env.process(self.CSMA_CA_UNS())

403 def repeatedElements(self, nodeObjectL, nodeNameL):
      listaAux = nodeNameL[:]
405   lista = [o.name for o in nodeObjectL]
      listaAux.extend(lista)
407   resultado = [item for item, count in collections.Counter(listaAux).items() if
      count > 1]
      return resultado
409
    def CCA (self):
411   vecinosTx = self.repeatedElements(self.vecinos, self.canal.nodesTx)
      if vecinosTx:
413     return False
```

```
    return True
415
  def EBFunction(self):
417   time = 0
    while True:
419     vecinosTx = self.repeatedElements(self.vecinos, self.canal.nodesTx)
      if len(vecinosTx) == 0 and self.flagTx == False and self.flagIFS == False and
    self.flagACKRx == False and self.flagACKTx == False:
421       if time + self.error >= self.EBTime:
          break
423       yield self.env.timeout(self.tsym/self.sym)
        time = time + (self.tsym/self.sym)
425     elif self.flagACKTx == True:
        self.printObject.printEB(self.name, self.env.now, 'Se pausa', self.EBTime,
    self.BEexp, self.auxBEexp)
427       if self.txProcessACK == None:
          yield self.env.timeout(self.error)
429       yield self.txProcessACK
        self.printObject.printEB(self.name, self.env.now, 'Se reanuda', self.EBTime,
     self.BEexp, self.auxBEexp)
431     elif self.flagIFS == True:
        self.printObject.printEB(self.name, self.env.now, 'Se pausa', self.EBTime,
    self.BEexp, self.auxBEexp)
433       yield self.IFSprocess
        self.printObject.printEB(self.name, self.env.now, 'Se reanuda', self.EBTime,
     self.BEexp, self.auxBEexp)
435     elif self.flagACKRx == True:
        self.printObject.printEB(self.name, self.env.now, 'Se pausa', self.EBTime,
    self.BEexp, self.auxBEexp)
437       yield self.ACKrequestProcess
        self.printObject.printEB(self.name, self.env.now, 'Se reanuda', self.EBTime,
     self.BEexp, self.auxBEexp)
439     else:
        if self.flagTx == True:
441         vecinoTx = self
        else:
443         vecinoTx = self.vecinos[self.getAllindexOfNodes(self.vecinos, vecinosTx
    [0])[0]]
        self.printObject.printEB(self.name, self.env.now, 'Se pausa', self.EBTime,
    self.BEexp, self.auxBEexp)
445       if vecinoTx.txProcess == None:
          yield vecinoTx.txProcessACK
447       elif vecinoTx.txProcessACK == None:
          yield vecinoTx.txProcess
449       else:
          yield vecinoTx.txProcess & vecinoTx.txProcessACK
451       self.printObject.printEB(self.name, self.env.now, 'Se reanuda', self.EBTime,
     self.BEexp, self.auxBEexp)

453   def CSMA_CA_UNS(self):
    if self.queue and self.flagTx == False and self.flagIFS == False:
455     self.printObject.printEB(self.name, self.env.now, 'Comienza', self.EBTime,
    self.BEexp, self.auxBEexp)
      self.EBprocess = self.env.process(self.EBFunction())
457     yield self.EBprocess
      self.printObject.printEB(self.name, self.env.now, 'Termina', self.EBTime, self
    .BEexp, self.auxBEexp)
459     self.printObject.printCCA(self.name, self.env.now, 'Comienza', '---')
      yield self.env.timeout(self.CCAtime - self.error)
461     canalState = self.CCA()
      pack = self.queue.pop(0)
```

```
463         pack.setupFrame(self.env.now, self.command)
            if canalState and self.flagTx == False and self.flagIFS == False:
465           self.pack = pack
              self.printObject.printCCA(self.name, self.env.now + self.error, 'Termina', '
    Libre')
467           self.txProcess = self.env.process(self.tx())
              yield self.txProcess
469         else:
              yield self.env.timeout(self.error)
471           self.printObject.printCCA(self.name, self.env.now, 'Termina', 'Ocupado')
              self.unsuccessfulTx (pack)
473           self.env.process(self.CSMA_CA_UNS())
        else:
475       yield self.env.timeout(self.tsym/self.sym)
          self.env.process(self.CSMA_CA_UNS())
477
    def setupNode (self, vecinos, command, canal, printObject):
479   yield self.env.timeout(0)
      self.vecinos = vecinos
481   self.command = command
      self.canal = canal
483   self.printObject = printObject
      self.getEB()
485   yield self.env.timeout(random.expovariate(1.0/self.timeMaxToSleep))
      self.env.process(self.putQueue())
487   self.env.process(self.CSMA_CA_UNS())
      self.env.process(self.rx())
```

# Apéndice B

# Clase Nodo PAN

```
import random
import simpy
import time
from decimal import Decimal
import collections
from Pack import Pack
from Node import Node
from Canal import Canal
from printClass import printClass

class NodoPAN(Node):
  def __init__(self, env, idPAN, coordinator, name, grafo, printFunctions,
    probabilities, time):
    Node.__init__(self, env, idPAN, coordinator, name, probabilities, time) #The PAN
     Node inherits from the Node class
    self.commandCoor = [2, 3, 7] #Commands that a coordinator node can use
    self.commandNC = [1, 3, 4, 5, 6, 8, 9] #Commands that a device node can use
    self.grafo = grafo #Experimental graph
    self.nodes = [] #List of node objects
    self.printConsole = printFunctions[0] #Boolean variable to enable console
    printing
    self.printTxt = printFunctions [1] #Boolean variable to enable animation
    self.probabilities = probabilities #Probabilities list
    self.timeBetweenPack =time #Generate a packet approximately every x seconds

  def makeNetwork(self):
    self.canal = Canal(self.env, len(self.grafo))
    for i in self.grafo.keys():
      if i == self.name:
        continue
      elif len(self.grafo[i]) > 1:
        node = Node(self.env, self.idPAN, True, i, self.probabilities, self.
    timeBetweenPack)
        self.nodes.append(node)
      else:
        node = Node(self.env, self.idPAN, False, i, self.probabilities, self.
    timeBetweenPack)
        self.nodes.append(node)
    for i in self.nodes:
      auxVecinos = []
      if i.coordinator == True:
        for j in self.grafo[i.name]:
          auxVecinos.append(self.nodes[i.getAllindexOfNodes(self.nodes, j)[0]])
        printObject = printClass(self.printConsole, self.printTxt)
```

```
40          yield self.env.process(i.setupNode(auxVecinos, self.commandCoor, self.canal,
     printObject))
        else:
42          for j in self.grafo[i.name]:
              auxVecinos.append(self.nodes[i.getAllindexOfNodes(self.nodes, j)[0]])
44          printObject = printClass(self.printConsole, self.printTxt)
            yield self.env.process(i.setupNode(auxVecinos, self.commandNC, self.canal,
    printObject))
```

# Apéndice C

# Clase Paquete

```
import random

class Pack:
  def __init__(self, typeF, secuence, node, source, idPAN):
    self.lenHeader = 0#Byte size of the header
    self.lenPay = 0#Byte size of the payload
    self.lenFCS =0 #Byte size of the FCS
    self.auxFrameRetries = 0 #Number of forwardings per package
    self.typeF = typeF #Frame type
    self.aMaxFrameRetries = 3 #Maximum number of retransmissions
    self.secuence = secuence #Sequence number
    self.txTimeStop = 0 #Time the packet stopped transmitting
    self.node = node #Recipient node object
    self.source = source #Name node source
    self.idPAN = idPAN #Identifier of the source network
    self.lenPHY = 6 #PHY layer data bytes
    self.ACKtime = 0 #Time an ACK frame spends on the source device
    self.totalLen = 0 #Total length of the frame
    self.ACKrequest = False #Boolean ACK request variable
    self.ACKanswer = False #Boolean ACK commit variable
    self.flag = False #Flag to know if the package values have been initialized
    self.FCSbool = None #Flag to know if a package has errors

  def retransmitionPack(self):
    self.auxFrameRetries += 1

  def makeHeader(self):
    if self.typeF == "ACK" :
      self.lenHeader = 2+1
    elif self.typeF == "BEA":
      self.lenHeader = 2+1+10
    else:
      self.lenHeader = 2+1+20

  def beaconFrame(self):
    self.lenPay = 2 + random.randrange(60, 120)

  def dataFrame(self):
    self.lenPay = random.randrange(61, 122)

  def ackFrame(self):
    self.lenPay = 0

  def commandFrame(self, command):
```

```
45      commandTypes = ["Association request", "Association response", "Disassociation
        notification",
        "Data Request", "PAN ID conflict notification", "Orphan notification", "
        Coordinator realigment",
47      "Beacon Request", "GTS request"]
        self.lenPay = 1 + random.randrange(60, 121)
49
    def makePayload(self, command):
51      if self.typeF == "ACK":
          self.ackFrame()
53
        elif self.typeF == "BEA":
55        self.beaconFrame()

57      elif self.typeF == "COM":
          self.commandFrame(random.choice(command))
59
        else:
61        self.dataFrame()

63  def makeFCS(self):
      self.lenFCS = 2
65
    def setupFrame(self, txTime, command):
67    if self.flag == False:
        self.makeHeader()
69      self.makePayload(command)
        self.makeFCS()
71      self.totalLen = self.lenHeader + self.lenPay + self.lenFCS + self.lenPHY
        self.flag = True
73    self.FCSbool = None
      self.txTime = txTime
```

# Apéndice D

# Clase Canal

```
import random
import simpy
import time
from decimal import Decimal
from Pack import Pack

class Canal:

  def __init__(self, env, num_medium):
    self.env = env #Environment
    self.packT = [] #List of packets transmitting
    self.wirelessMedium = simpy.Resource(env, num_medium) #Resource
    self.nodesTx = [] #List of nodes transmitting
    self.error = 0.000001 #Error constant

  def intoCanal(self, pack, vecinos):
    self.packT.append(pack)
    yield self.env.timeout(self.error)
    vecinosTx = pack.node.repeatedElements(vecinos, self.nodesTx)
    if vecinosTx:
      pack.FCSbool = True
```

# Apéndice E

# Clase Impresión

```
class printClass:
  def __init__(self, pConsole, pTxt):
    self.printConsole = pConsole #Flag to enable or disable console printing
    self.printTxtBool = pTxt #Flag to enable or disable txt printing
  #Funciones para imprimir en consola

  def printPack(self, pack, time):
    if self.printConsole == True:
      print('================================================================')
      print('Accion: {action}\t Requiere ACK: {ACK}'.format(action='Creacion de
    paquete', ACK=pack.ACKrequest))
      print('Tipo de paquete: {pack}\t\t Nodo Rx: {nodeRx}'.format(pack=pack.typeF,
    nodeRx=pack.node.name))
      print('Nodo Tx: {node}\t\t\t Numero de secuencia:{numberSec}'.format(node=pack
    .source, numberSec=pack.secuence))
      print('Tiempo: {time}'.format(time=time))
      print('================================================================')
      #raw_input("Pulsa una tecla para continuar...")

  def printTx(self, pack, time, status, paqsent, paqTirados, BEexp, auxBEexp):
    if self.printConsole == True:
      print('================================================================')
      print('Accion: {action}\t\t Tamano en Bytes del paquete: {packLen}'.format(
    action='Transmision', packLen=pack.totalLen))
      print('Tipo de paquete: {pack}\t\t Requiere ACK: {ACK}'.format(pack=pack.typeF
    , ACK=pack.ACKrequest))
      print('Nodo Tx: {node}\t\t\t Numero de secuencia:{numberSec}'.format(node=pack
    .source, numberSec=pack.secuence))
      print('Numero de retransmision: {numberRet}\t Estatus: {status}'.format(
    numberRet=pack.auxFrameRetries, status=status))
      print('EB:{EB}\t\t\t Paquetes Tirados:{packFailed}'.format(EB=BEexp,
    packFailed=paqTirados))
      print('Repeticiones del EB maximo:{NB}\t Paquetes Tx:{packTx} '.format(NB=
    auxBEexp, packTx=paqsent))
      print('Nodo Rx: {nodeRx}\t\t\t Tiempo: {time}'.format(nodeRx=pack.node.name,
    time=time))
      print('================================================================')
      #raw_input("Pulsa una tecla para continuar...")

  def printTxACK(self, pack, time, status, caducidad, paqsent, paqTirados, ACKtime):
    if self.printConsole == True:
      print('================================================================')
      print('Accion: {action}\t\t Paquetes Tx:{packTx}'.format(action='Transmision',
     packTx=paqsent))
```

```
      print('Nodo Tx: {node}\t\t\t Ha caducado: {Caducidad}'.format(node=pack.source
    , Caducidad=caducidad))
35    print('Tipo de paquete: {pack}\t\t Tiempo de ACK:{Wtime}'.format(pack=pack.
    typeF, Wtime=ACKtime))
      print('Tamano en Bytes del paquete: {packLen}\t Estatus: {status}'.format(
    packLen='11', status=status))
37    print('Numero de secuencia:{numberSec}\t\t Paquetes Tirados:{packFailed}'.
    format(numberSec=pack.secuence, packFailed=paqTirados))
      print('Nodo Rx:{Rx}\t\t\t Tiempo: {time}'.format(Rx=pack.node.name, time=time)
    )
39    print('================================================================')
      #raw_input("Pulsa una tecla para continuar...")
41
  def printRx(self, pack, time, lista, status, col, errores):
43  if self.printConsole == True:
      print('================================================================')
45    print('Accion: {action}\t\t Estatus: {status}'.format(action='Recepcion',
    status=status))
      print('Tipo de paquete: {pack}\t\t Requiera ACK: {ACK}'.format(pack=pack.typeF
    , ACK=pack.ACKrequest))
47    print('Nodo Rx: {node}\t\t\t Contiene errores: {FCS}'.format(node=pack.node.
    name, FCS=errores))
      print('Nodo Tx del paquete: {nodeRx}\t\t Colisiono: {col}'.format(nodeRx=pack.
    source, col=col))
49    print('Numero de secuencia:{numberSec}\t\t Paquetes Rx:{packRx}'.format(
    numberSec=pack.secuence, packRx=pack.node.paqReceivedCorrectly))
      print('Nodos Tx: {lista}'.format(lista=lista))
51    print('Tiempo: {time}'.format(time=time))
      print('================================================================')
53    #raw_input("Pulsa una tecla para continuar...")

55  def printRxACK(self, pack, time, status, Tx, Rx, errores):
    if self.printConsole == True:
57    print('================================================================')
      print('Accion: {action}\t\t Estatus: {status}'.format(action='Recepcion',
    status=status))
59    print('Nodo Tx: {nodeTx}\t\t\t Tipo de paquete: {pack}'.format(nodeTx= Tx,
    pack='ACK'))
      print('Nodo Rx: {node}\t\t\t Numero de secuencia:{numberSec}'.format(node= Rx,
     numberSec=pack.secuence))
61    print('Contiene errores: {FCS}\t\t Tiempo: {time}'.format(FCS=errores, time=
    time))
      print('================================================================')
63    #raw_input("Pulsa una tecla para continuar...")

65  def printCCA(self, node, time, status, canal):
    if self.printConsole == True:
67    print('================================================================')
      print('Accion: {action}\t\t\t Estatus: {status}'.format(action='CCA', status=
    status))
69    print('Nodo: {node}\t\t\t Estado del canal: {canal}'.format(node=node, canal
    =canal))
      print('Tiempo: {time}'.format(time=time))
71    print('================================================================')
      #raw_input("Pulsa una tecla para continuar...")
73
  def printEB(self, node, time, status, timeEB, BEexp, auxBEexp):
75  if self.printConsole == True:
      print('================================================================')
77    print('Accion: {action}\t\t\t Estatus: {status}'.format(action='EB', status=
    status))
```

```
      print('Nodo: {node}\t\t\t\t Tiempo EB:{timeEB}'.format(node=node, timeEB=
    timeEB))
      print('EB:{EB}\t\t\t\t Repeticiones del EB maximo:{NB}'.format(EB=BEexp, NB=
    auxBEexp))
      print('Tiempo: {time}'.format(time=time))
      print('=================================================================')
      #raw_input("Pulsa una tecla para continuar...")

  def printIFS(self, node, time, status, action):
    if self.printConsole == True:
      print('=================================================================')
      print('Accion: {action}\t\t\t Estatus: {status}'.format(action=action, status=
    status))
      print('Nodo: {node}\t\t\t\t Tiempo: {time}'.format(node=node, time=time))
      print('=================================================================')
      #raw_input("Pulsa una tecla para continuar...")

  def printCol(self, node, time, nodoRx):
    if self.printConsole == True:
      print('=================================================================')
      print('Accion: {status}\t\t Nodo Rx: {nodoRx}'.format(status='Colision',
    nodoRx=nodoRx))
      print('Nodo Tx: {node}\t\t\t Tiempo: {time}'.format(node=node, time=time))
      print('=================================================================')
      #raw_input("Pulsa una tecla para continuar...")

  #Funciones para imprimir en el documento

  def printTxt(self, pack, time, action, Rx, Tx, tamano, status, errores,
    retransmision):
    if self.printTxtBool == True:
      f = open ('Registro.txt', 'a')
      f.write('{accion}\t {nodoTx}\t {nodoRx}\t {pack}\t {retransmision}\t {errores
    }\t {tamano}\t {status}\t {tiempo}\n'.format(
          accion=action, nodoTx=Tx, nodoRx=Rx, pack=pack.typeF, tamano=tamano, status=
    status, tiempo=time, errores=errores, retransmision=retransmision))
      f.close()
```

# Apéndice F

# Simulador

```
import simpy
import random
from NodoPAN import NodoPAN

RANDOM_SEED = 45 #Random number to repeat the results
SIM_TIME = 0.5  #Simulation time
printFunctions = [False, True] #to enable or disable printing to console and text file, respectively
timeBetweenPack = 0.2 #Generate a packet approximately every x seconds
probCOMFrame = 0.2 #Probability of creating a command package every time a data package is created
probError = 0.1 #Probability that a package contains errors, without considering collisions
probACK = 0.5 #Probability that a package requires ACK
nodePANName = 'A'
#First test
grafo = {'A':['B'], 'B':['A']}
#Second test
#grafo = {'A':['B', 'C', 'D', 'E', 'F', 'G'], 'B':['A'], 'C':['A'], 'D':['A'], 'E':['A'], 'F':['A'], 'G':['A']}
#Third test
#grafo = {'A':['B', 'C', 'D', 'E'], 'B':['A'], 'C':['A', 'I', 'H'], 'D':['A', 'L', 'K', 'J'],
#    'E':['F', 'G', 'A'], 'F':['E'], 'G':['E'], 'H':['C'], 'I':[ 'C'], 'J':['D'], 'K':['D'],
#    'L':['D']}
#grafo = {'a':[1,2,3,4,5,6,7,8,9],'b':[3,4,5,6,7,8,9,11,12],'c':[5,6,7,8,9,13,14,15,16],'d':[2,3,4,5,6,7,8,9,17],'e':[3,4,5,6,7,8,9,18],'f':[4,5,6,7,8,9,19],'g':[1,2,11,12],'h':[3,4,13,14],'i':[5,6,15,16,19],'j':[7,8,9,17,18]}



print('\nStart the simulation')
random.seed(RANDOM_SEED)
probabilities = [probCOMFrame, probError, probACK] #Probabilities list
f = open ('Registro.txt', 'w')#The file where the information will be placed is created
f.close()
env = simpy.Environment() #The simulation environment is created
NodoPAN = NodoPAN(env, 1, True, nodePANName, grafo, printFunctions, probabilities, timeBetweenPack) #PAN node is created
NodoPAN.nodes.append(NodoPAN)
```

```
env.process(NodoPAN.makeNetwork())#The rest of the nodes are created together with
    the network
env.run(until=SIM_TIME)
print('\nFinish the simulation')
```

# Apéndice G

# Animación

```
import networkx as nx
import matplotlib.pyplot as plt
from vpython import *
import numpy as np
import time

class Interface:
  def __init__(self, archivo, graph, nodePAN):
    self.Rsps = float(62500) #Modulation speed
    self.archivo = archivo #Pointer to a file
    self.nodePos = None #Nodes' coordinates
    self.graph = graph #Experimental graph
    self.nodes = {} #Sphere dictionary {identifier:sphere object}
    self.nodePAN = nodePAN #PAN node name
    self.events = [] #List of events to simulate
    self.tsym = 1/self.Rsps #Symbol time
    self.titleS = 'Slow Rate x' #Scroll title
    self.wts = [] #Scroll values list
    self.slider = None #Scroll object

  def getPosition(self):
    G = nx.Graph()
    G.add_nodes_from(self.graph.keys())

    for i in self.graph.keys():
      for j in self.graph[i]:
        G.add_edge(i, j)
    self.nodePos = nx.spring_layout(G)

  def nodeCreation(self):
    for i in self.graph.keys():
      if self.nodePAN == i :
        self.nodes[i] = sphere(pos=vector(self.nodePos[i][0], self.nodePos[i][1], 0)
    ,radius=0.1, color=color.red)
      elif len(self.graph[i]) == 1 :
        self.nodes[i] = sphere(pos=vector(self.nodePos[i][0], self.nodePos[i][1], 0)
    ,radius=0.1, color=color.white)
      else:
        self.nodes[i] = sphere(pos=vector(self.nodePos[i][0], self.nodePos[i][1], 0)
    ,radius=0.1, color=color.yellow)
      label(pos=self.nodes[i].pos, text=i, height=20, font='sans', opacity = 0, box
   = False, color = color.black)

  def edgeCreation(self):
    for i in self.graph.keys():
```

```
42      for j in self.graph[i]:
          cylinder(pos = self.nodes[i].pos, axis = self.nodes[j].pos - self.nodes[i].
    pos, radius = 0.01, color = color.white)
44
  def packCreation(self, typeF, source, destination):
46    a = self.nodes[source].pos
      b = self.nodes[destination].pos - self.nodes[source].pos
48    if typeF == 'DAT':
        return sphere(pos = a, axis = b, radius=0.03, color=color.cyan)
50    elif typeF == 'COM':
        return sphere(pos = a, axis = b, radius=0.03, color=color.blue)
52    else:
        return sphere(pos = a, axis = b, radius=0.03, color=color.green)
54
  def getEvent(self):
56    i = 0
      for line in self.archivo:
58      event = []
        data = line.split()
60      if data[0] != 't' or data[7] == 'Termina':
          continue
62      else:
          package = self.packCreation(data[3], data[1], data[2])
64        event.append(package)
          event.append(float(data[6])*2*self.tsym)
66        event.append(float(data[8]))
          event.append(i)
68        event.append(event[0].axis + event[0].pos)
          self.events.append(event)
70        i += 1
          f = open ('Evento.txt', 'a')
72        f.write("".join(data))
          f.write("\n")
74        f.close()
      self.archivo.close()
76    #Creating a placebo event for the simulation to run correctly
      event = []
78    package = sphere(pos = vector(0,0,0), radius=0.03, visible = False)
      event.append(package)
80    event.append(1*2*self.tsym)
      event.append(float(data[8]) + 1)
82    event.append(i)
      event.append(vector(1,1,1))
84    self.events.append(event)

86  def set_background(self, sl):
      self.wts[0].text = '{}'.format(sl.value)
88


90  def start(self):
      startTime = time.time()
92    events = []
      event = self.events.pop(0)
94    flagStart = []
      flagStop = []
96    initialPos = []
      flagSim = True
98    while flagSim:
        t =time.time()
100     rate((4*self.Rsps/self.s.value))
        currentTime = time.time()
```

```
      while True:
        if currentTime - startTime >= (event[2] * self.s.value):
          events.insert(0, event)
          flagStart.append(True)
          flagStop.append(False)
          initialPos.append(vector(event[0].pos))
          if self.events:
            event = self.events.pop(0)
          else:
            break
        else:
          break
      for i in events:
        if flagStart[i[3]] == True and flagStop[i[3]] == False:
          if time.time() - startTime >= (i[1]+i[2])*self.s.value:
            i[0].visible = False
            events.remove(i)
            flagStop[i[3]] = True
            if len(events) == 0:
              flagStart.clear()
              flagStop.clear()
              self.eventsSim.clear()
              if len(self.events) == 0:
                flagSim = False
          else:
            i[0].pos = initialPos[i[3]] +  ((((time.time() - startTime) - (i[2]*self
    .s.value))/((i[1])*self.s.value)) * i[0].axis)


  def setUp(self):
    self.getPosition()
    self.nodeCreation()
    self.edgeCreation()
    self.getEvent()
    self.s = slider(length=300, left=10, min=100, max=12000, step=100, bind = self.
    set_background)
    scene.append_to_caption('    '+self.titleS+' ')
    self.wts.append(wtext(text='0.000'))
    scene.append_to_caption('\n\n')
    self.s.value = 4000
    self.wts[0].text = '4000'
    self.start()

def main():
  #First test
  #graph = {'A':['B'], 'B':['A']}
  #Second test
  #graph = {'A':['B', 'C', 'D', 'E', 'F', 'G'], 'B':['A'], 'C':['A'], 'D':['A'], 'E
    ':['A'], 'F':['A'], 'G':['A']}
  #Third test
  graph = {'A':['B', 'C', 'D', 'E'], 'B':['A'], 'C':['A', 'I', 'H'], 'D':['A', 'L',
    'K', 'J'],
     'E':['F', 'G', 'A'], 'F':['E'], 'G':['E'], 'H':['C'], 'I':[ 'C'], 'J':['D'], 'K
    ':['D'],
     'L':['D']}
  f = open ('Registro.txt', 'r')
  nodePANName = 'A'


  scene.width = 1200
  scene.height = 720
```

```
158   title = 'IEEE 802.15.4 MAC Simulator\n'
      scene.title = title
160   I = Interface(f, graph, nodePANName)
      I.setUp()
162

164 if __name__ == '__main__':
        main()
```

# Bibliografía


[1] A. S. Guerrero, "Simulación de eventos," tech. rep., Centro Cultural Itaca S. C.

[2] K. Devadiga, "Ieee 802.15. 4 and the internet of things," *Aalto University School of Science*, 2007.

[3] A. ur Rehman Khana, S. M. Bilalb, and M. Othmana, "A performance comparison of network simulators for wireless networks," 2013.

[4] S. Shuermans and C. Voskoglou, *The global developer population 2019. How many developers are there?* Hatton Gardens, London: SlashData, 7 2019.

[5] "The Network Simulator." `https://www.isi.edu/nsnam/ns/`, 2020. [Online; accessed 13-Junuary-2020].

[6] M. J. H. Mohammed Humayun Kabir, Syful Islam and S. Hossain, "Detail comparison of network simulators," *International Journal of Scientific y Engineering Research*, vol. 5, pp. 200–218, 10 2014.

[7] A. T, "A comparative study of various network simulation tools," *International Journal of Computer Science y Engineering Technology*, vol. 7, pp. 200–218, 08 2016.

[8] "ns-3 Network Simulator." `https://www.nsnam.org/`, 2020. [Online; accessed 15-Junuary-2020].

[9] "What is OMNeT++?." `https://omnetpp.org/intro/`, 2020. [Online; accessed 16-Junuary-2020].

[10] V. Mishra and S. Jangale, "Analysis and comparison of different network simulators," *International Journal of Application or Innovation in Engineering y Management*, 2014.

[11] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems y Workshops*, Simutools '08, (Brussels, BEL), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[12] "QualNet - Network Simulation." `https://www.scalable-networks.com/qualnet-network-simulation`, 2020. [Online; accessed 18-Junuary-2020].

[13] "OPNET ahora forma parte de Riverbed." `https://www.riverbed.com/mx/products/steelcentral/opnet.html?redirect=opnet`, 2020. [Online; accessed 19-Junuary-2020].

[14] E. Weingartner, H. vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," in *2009 IEEE International Conference on Communications*, pp. 1–5, June 2009.

[15] D. Andreu, "Implementation and performance evaluation of ieee 802.15. 4 protocol," 2011.

[16] R. Rai, *IEEE 802.15. 4 protocol implementation and measurement of current consumption*. PhD thesis, University of North Carolina at Charlotte, 2006.

[17] N. S. Bhat, "Design and implementation of ieee 802.15. 4 mac protocol on fpga," *arXiv preprint arXiv:1203.2167*, 2012.

[18] "Discrete-event simulation with SimPy." `https://stefan.sofa-rockers.org/downloads/simpy-ep14.pdf`, 2014. [Online; accessed 20-Junuary-2020].

[19] "Overview — SimPy 3.0.11 documentation." `https://simpy.readthedocs.io/en/latest/`, 2019. [Online; accessed 20-Junuary-2020].

[20] S. Farahani, *ZigBee Wireless Networks and Transceivers*, vol. 1 of *1*. 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA Linacre House, Jordan Hill, Oxford OX2 8DP, UK: Newnes, 1 ed., 2008.

[21] S. R. Caprile, *Equisbí: Desarrollo de aplicaciones con comunicación remota basadas en módulos ZigBee y 802.15.4*, vol. 1 of *1*. Buenos Aires, Gran Aldea: Editores-GAE, 1 ed., 8 2009.

[22] M. C. Acosta Ponce, "Estudio del éstandar ieee 802.15. 4 zigbee para comunicaciones inalámbricas de área personal de bajo consumo de energía y su comparación en el estándar ieee 802.15. 1 bluetooth," B.S. thesis, QUITO/EPN/2006, 2006.

[23] J. Zheng and M. J. Lee, "A comprehensive performance study of ieee 802.15. 4," *Sensor network operations*, vol. 4, pp. 218–237, 2006.

[24] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "Performance evaluation of the ieee 802.15. 4 mac for low-rate low-power wireless networks," in *IEEE International Conference on Performance, Computing, and Communications, 2004*, pp. 701–706, IEEE, 2004.

[25] C. Wang, T. Jiang, and Q. Zhang, *ZigBee® Network Protocols and Applications*. CRC Press, 2016.

[26] W. Yuan, J.-P. M. Linnartz, and I. G. Niemegeers, "Adaptive cca for ieee 802.15. 4 wireless sensor networks to mitigate interference," in *2010 IEEE Wireless Communication and Networking Conference*, pp. 1–5, IEEE, 2010.

[27] I. Group *et al.*, "Part 15.4: Low-rate wireless personal area networks (lr-wpans)," *IEEE, IEEE Standard for Local and metropolitan area networks IEEE Std*, vol. 802, pp. 4–2011, 2011.

[28] N. Barroca, L. M. Borges, F. J. Velez, and P. Chatzimisios, "Ieee 802.15. 4 mac layer performance enhancement by employing rts/cts combined with packet concatenation," in *2014 IEEE International Conference on Communications (ICC)*, pp. 466–471, IEEE, 2014.

[29] J. Carles, "Ps_mem para ver el consumo de memoria ram de un programa." urlhttps://geekland.eu/ps_mem-consumo-ram-programa/, 2017.