UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Máster en Ingeniería de Computadores y Redes

Trabajo Fin de Máster

Diseño de protocolo de redes Mesh basado en LoRa

Autor: *: Daniel Alejandro Mejías Rojas*

Director(es): *Pietro Manzoni*

Septiembre 2021

Agradecimiento

Aunque este trabajo fue elaborado por mis conocimientos y esfuerzo puestos en él, poder llegar a este punto de mi vida, si recibí bastante apoyo. Cumplo una meta mas en mi vida y debo agradecer a muchas personas que me enseñaron a creer en mis sueños, me impulsaron a seguir creciendo y me ayudaron ellos momentos mas difíciles. Poder estar lejos de mi hogar y mi país no es fácil, ya que dejo atrás a todos mis familiares y amigos, pero, aunque se encuentren en la distancia ellos siguen apoyándome para seguir adelante. Mis padres, me enseñaron a tener dedicación y paciencia, a ir disfrutando la vida y construyendo el futuro que deseo. Me apoyaron en todos mis estudios realizados y aunque para esta meta se encuentra distantes, me siguen dando ese apoyo incondicional como pueden. De igual manera a mi hermano, que me empujo a expandir mis horizontes, a conocer nuevos lugares y nuevas personas. Le agradezco por siempre presionarme para que sea una excelente persona y pueda valerme de mi mismo disfrutando cada aventura. A mis amigos que, estando lejanos, en los momentos más tristes me ayudan a levantarme, a reír y a distraerme. Grandes amigos que me han acompañado y los considero como hermano. Agradezco también a los profesores que tuvieron la paciencia y dedicación de enseñarnos, de tomarse su tiempo para que comprendiéramos los temas y para que despertaran esa mente curiosa a la innovación y por último a la Universidad Politécnica de Valencia, que estando desde tan lejos me dio una gran oportunidad, me abrió sus puertas al conocimiento y me permitió conocer un gran país como lo es España. A todos les debo un agradecimiento por ayudarme a ser más fuerte, a ser una mejor persona, a ser un mejor profesional, a seguir cumpliendo mis metas y disfrutando la vida.

Resumen

En este trabajo se abordará el diseño de un programa de protocolo de comunicación utilizando la tecnología LoRa. Se dispondrá una red sensores que se conectaran en forma de una topología Mesh. Para este sistema se diseñará un protocolo de envío y recepción de mensajes proponiendo una estructura de paquete simple y se diseñará un protocolo de enrutamiento epidémico, donde la comunicación debe llegar a todos los dispositivos en su rango de alcance. De igual forma al ser LoRa una tecnología deseñada para IoT, se tratarán de cumplir los requisitos para ser efectivos en áreas que requieran sensores y ser de bajo consumo de potencia.

Palabras claves: LoRa, redes Mesh, redes de bajo consumo, IoT, Pycom, enrutamiento epidémico,

Tabla de Contenido

Índices generales, tablas e imágenes

1. Introducción

Poder comunicarse con otras personas ya no es una meta imposible. Ahora, la nueva conquista que busca el desarrollo de las comunicaciones es comunicarse con cosas. Con esta idea en mente surge IoT. Esto con el objetivo de mejorar las actividades que realizamos a diario y mejorar nuestra calidad facilitando algunas tareas. Poder controlar varios objetos a distancia en la palma de la mano con simples instrucciones ya se ha convertido en un hecho.

El internet de las cosas (IoT) viene a permitir conectar cosas cotidianas al internet. Esto permite crear una red digital entre nuestros dispositivos, en los cuales se puede configurar para que trabaje de manera autónoma, sin supervisión humana. Ahora, el desarrollo que busca expandirse en esta tecnología es en la manera de comunicarse entre los dispositivos. IoT abrió la puerta a buscar nuevas maneras de comunicación que se adapten a los deseos de desarrollo.

Uno de los planteamientos que se realiza al día de hoy es la utilización de dispositivos de largo alcance y bajo consumo. Esto con mente en la idea de la sensorización, donde los dispositivos se encuentren a grandes distancias y puedan mantener una comunicación óptima y mantener una conectividad segura pese a no encontrarse estático en un lugar. Ahora, si un dispositivo no se encuentra estático, debe depender de una alimentación propia, usando baterías. Por eso otro de los objetivos que se busca en IoT es el diseño de tecnología que logre soportar bajo consumo de energía en su funcionamiento.

LoRa es una de las tecnologías de IoT que se está implementando para comunicación de dispositivos de largo alcance y bajo consumo. LoRa es una tecnología inalámbrica al igual que WiFi, Bluetooth, LTE, SigFox o Zigbee. Sin embargo, esta tecnología ha logrado traer mejoras en su manera de transmisión, haciéndola una de las grandes opciones para diseños en IoT y para su utilización en Smart Cities.

El fin de este trabajo es adentrarse en el diseño de un método de comunicación con la tecnología LoRa, donde se abordará la forma de implementar esta tecnología mediante programación en Python y conseguir los mejores resultados en una red de dispositivos y con un bajo consumo de energía.

Motivación

Al realizar un trabajo uno tiene que disfrutar lo que realiza. Puede ser un trabajo de exigencia o de riesgo pero si es algo que le apasiona o lo disfruta es simple motivación para seguir cada día y sentirse autorrealizado. Elegir este tema para trabajo final de máster fue pensando en lo que me motiva trabajar algún día. Mi área de estudio principal fue la rama de las telecomunicaciones y entré en ella porque disfrutaba a la par el área de la electrónica y el área de informática. Con las telecomunicaciones conseguí ese punto medio entre las dos áreas. Con este trabajo puedo aprender de tecnologías de comunicación en IoT, una de las áreas que se encuentra en crecimiento en las telecomunicaciones. Aprender la tecnología LoRa y utilizar dispositivos de comunicación es una oportunidad que vi de opción de crecimiento y realizar este trabajo cumplió el objetivo de disfrutar del área en que me estoy especializando el día de hoy.

Objetivos

El objetivo del trabajo se encuentra realizar un protocolo de Lora para ser utilizados dispositivos Lopy4 de la marca Pycom que permite crear una red Mesh de dispositivos y permite ser usado en bajo consumo.

- Diseñar un programa que permita la comunicación entre varios dispositivos de LoRa.
- Implementar una estructura simple para paquetización de mensajes en LoRa.
- Diseñar un sistema de enrutamiento epidémico para lograr una conexión de red Mesh.
- Implementar un código que permita reducir el consumo de energía del sistema adaptándose a las exigencias del sistema.

Impacto Esperado

El diseño de este protocolo de comunicación va a permitir crear una comunicación entre varios dispositivos a una tasa baja de consumo de potencia. Con esta idea en mente se puede diseñar programas para sensores que realizan mediciones de temperatura o presión atmosférica. Los cuales no necesariamente deben estar ubicados en un sitio estático. El protocolo se plantea diseñar para encontrarse en un estado móvil, donde en cualquier sitio donde se encuentre tenga la posibilidad de comunicarse con la red principal. Por eso, con el uso de la tecnología LoRa se puede

utilizar para comunicaciones de largo alcance y donde todos los nodos participantes se convierten en canales de transmisión, mejorando así la red de comunicación. Partiendo de esta idea, el sistema puede ser utilizado para geolocalización. Puede ser implementado para personas que sufran enfermedades como el Alzheimer y se extravíen o para uso de investigación de la fauna, donde se realiza localización de animales, en donde mayor es la cantidad de dispositivo mejor es el comportamiento de la red.

Metodología

Para la elaboración de este trabajo se partirá del uso de la experimentación. Se iniciará en el diseño de un código de programación en lenguaje Python y se implementará en dispositivos compatibles a esta para la realización de pruebas. Para la recolección de datos el sistema cuenta con una terminal donde se ajustará el código para obtener resultados en ciertas partes del programa. De igual manera se utilizará equipo especializados para la medición de consumo de potencia en los dispositivos.

Estructura

Para la estructura del trabajo se realizará una breve explicación de los conceptos teóricos que engloba este trabajo, con la idea de plantear las características de la tecnología que se planea utilizar y explicar la topología de la red que se desea implementar. Luego se realizará un planteamiento y descripción de los sistemas y equipos que serán necesarios para realizar las pruebas del proyecto. Se propone un plan de trabajo para el desarrollo del código, exponiendo las fases que se van a cubrir hasta conseguir la estructura del código final. Seguidamente, se realiza una explicación de la estructura del código, cumpliendo cada fase para luego seguir con la implementación del código en los dispositivos y pruebas del sistema diseñado. Finalmente se realiza un análisis general de los resultados y se concluye en posibles mejoras del sistema para futuro.

2. Estado del arte

Internet de las Cosas (IoT)

Internet de las Cosas es la capacidad de realizar conexiones en una red con objetos físicos que son de usos cotidianos y los cuales son adaptados para realizar procesos de cómputo. Esto se puede conseguir con la utilización de sensores, software y usos de otros dispositivos tecnológicos que permitan conectar e intercambiar datos con otros dispositivos y sistemas a través de Internet. Esto permite crear sistemas digitales. (1) (2)

LoRa

El significado de LoRa (*Long Range)* se refiere al Internet de las Cosas de Largo Alcance. LoRa es una técnica de modulación de espectro ensanchado que se utiliza para crear enlaces de comunicación de largo alcance manteniendo un uso de baja potencia que son características usadas similarmente a las de la modulación por desplazamiento de frecuencia. Esta clase de modulación empleada se deriva de una tecnología de chirp de espectro ensanchado o amplio espectro (*spread spectrum*) la cual fue patentada por Semtech y actualmente es administrada por la "LoRa Alliance" una organización integrada por IBM, Cisco y Alibaba para la colaboración en el desarrollo de esta tecnología. (1)

La tecnología LoRa es una plataforma inalámbrica que permite conectividad de largo alcance y bajo consumo y por eso se convierte en una de las tecnologías ideales para el Internet de las Cosas y ser la mayor utilizadas en las Ciudades Inteligentes (*Smart Cities)*. Lora se ha convertido en una tecnología de utilidad en un gran número de mercados de distintos sectores (como agricultura, ganadería, servicios, seguridad pública, fabricación, etc.). Esto gracias a las características principales que posee LoRa: (3) (4)

- Alta resistencia a interferencias,
- Alta sensibilidad (-168 dB), basada en modulación Chirp,
- Bajo consumo de energía, pueden consumir muy poca energía que puede durar hasta 10 años con batería,
- Largo alcance entre 10 a 20 km conexiones entre nodos sensores,

- Bajos datos transferencia (hasta 255 bytes),
- Conexión punto a punto
- Seguridad con cifrado de datos AES-128 extremo a extremo integrado, el cual hace que interceptar la información sea prácticamente imposible.
- Tecnológica escalable ya que permite la conexión de miles de sensores con miles de nodos.

LoRa está diseñado para manejar una velocidad de datos variables, permitiendo tolerar factores de dispersión ortogonales. Esto permite al sistema poder controlar la velocidad de datos por el rango o potencia, permitiendo buscar la configuración óptima que se adapte al rendimiento de la red en un ancho de banda constante. Los anchos de bandas ISM compatibles para LoRa son: (5)

- 915 MHz en América
- 868 MHz en Europa
- 433MHz en Asia

	Europa	América
Banda de frecuencia	867-869 MHz	902-928 MHz
Canales	10	64 + 8 + 8
Canal banda ancha ascendente	125/250 kHz	125/500 kHz
Canal banda ancha descendente	125 kHz	500 kHz
TX encendido	+14 dBm	+20 dBm típ (+30 dBm permitidos)
TX desconectar	+14 dBm	+27 dBm
SF Up	7-12	7-10
Velocidad de datos	250 bps - 50 kbps	980 bps - 21.9 kbps

Link Budget Up	155 dB	154 dB
Link Budget Dn	155 dB	157 dB

Tabla 1 comparación de las especificaciones LoRa para Europa y Estados Unidos, dos regiones donde se utilizan bandas ISM ampliamente. (Fuente de la imagen: LoRa Alliance)[1]

Topología Mesh

La topología Mesh o malla, es una topología que permite la conexión directa entre un dispositivos y los demás de la red. Esto lo hace una red ideal a tolerancia a fallas, ya que permite tener varios caminos a los diferentes nodos. Al fallar uno de los nodos o enlaces, el sistema automáticamente puede escoger otro camino hasta llegar a su destino. Para este tipo de topología, realizarlo de forma física genera grandes costos ya que es necesario una gran cantidad de cable, por eso es un sistema mejor implementado en redes inalámbricas, ya que el sistema depende de la transmisión en radiofrecuencia y no se deben utilizar cables. De igual manera los dispositivos inalámbricos deben tener la capacidad de soportar varios canales de transmisión lo que genera un costo mayor. (6) (7)

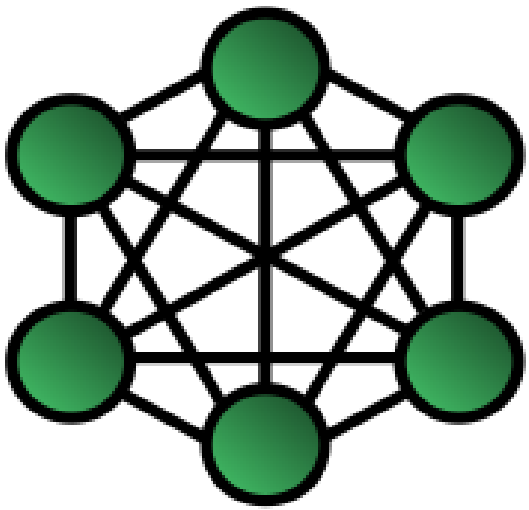

Ilustración 1 conexión en topología Mesh

[1] https://lora-alliance.org/

Ventajas:

- Con este tipo de red se pueden gestionar grandes cantidades de tráfico, gracias a la capacidad de conectar múltiples dispositivos que pueden transmitir simultáneamente
- La falla de un dispositivo no genera pérdida de la conexión de la red o en la transmisión de datos.
- Incorporar nuevos dispositivos no altera la transmisión de datos de los demás dispositivos.
- Esta red es escalable ya que permite agregar tantos dispositivos como se desee.

Desventajas:

- Es una de las redes más costosas a la hora de ser implementada, esto debido a la cantidad de dispositivos que son necesarios para mantener las conexiones.
- Construir y mantener la red es difícil y requiere mucho tiempo.

Redes Mesh y uso de tecnología LoRa

La red mesh es la red usada frecuentemente para la tecnología LoRa. Esto debido a su versatilidad y facilidad de implementación. Para la implementación de redes Mesh con la tecnología LoRa es ideal su uso para largo alcance, ya que permite preservar la vida útil de las baterías. En LoRa los nodos no se asocian específicamente a una pasarela o puerta de enlace específico, como lo hacen otras tecnologías (redes móviles 3G/4G). Cada uno lo transmite a múltiples pasarelas y para ser lograr una conexión a un servidor de red en la nube debe existir la conexión con otras tecnologías (móvil, Ethernet, satélite, o Wi-Fi). El servidor de red será el encargado de aplicar políticas de seguridad para administrar la complejidad de la red. (6) (7)

Objetivos y problemas al diseñar una red

- Enrutamiento: al realizar un envío de un paquete es necesario determinar el camino que se debe recorrer hasta el dispositivo final. Como los emisores no conocen la ubicación del destino es inexacto el camino a recorrer. Por eso, al no tener una ruta específica, cada dispositivo al alcance serán receptores de mensajes para así intentar alcanzar al dispositivo final. (8) (9)

- Asignación de recursos: al no considerar un sistema con un enrutamiento estándar, se debe contar con diferentes copias del mensaje en tránsito simultáneamente. Por eso se debe verificar que un mensaje no agote los recursos de la red y evitar los posibles bucles que se puedan presentar. (8)

- Rendimiento: Latencia promedio del mensaje, cantidad promedio de ancho de banda de comunicación y almacenamiento del sistema consumida en la entrega de un mensaje, cantidad de energía consumida en la entrega de un mensaje

- Fiabilidad: Para un sistema mesh la necesidad de reconocer una entrega exitosa de un mensaje no es necesariamente importante, sin embargo, se puede colocar recursos para comprobar la recepción de un mensaje. (8)

- Seguridad: un mensaje recorre una ruta arbitraria hasta llegar a su destino. Dependiendo del receptor, pueden necesitar garantizar la fiabilidad del mensaje debido a la sensibilidad que posee el mensaje y los requisitos de la aplicación. Para esto puede ser de utilidad el uso de criptografía para el cifrado del mensaje y otra manera es rastrear la ruta completa que recorre el mensaje hasta llegar a su destino. Así se puede conocer todos los hosts y saber si fue expuesto en alguno o se consigue un host que no es de confianza. Con esta información se puede hacer filtros para delimitar cuáles nodos no deben ser considerados. (8) (6)

Protocolo de Enrutamiento

El protocolo estándar a considerar para una red Mesh es el protocolo de enrutamiento epidémico. En este, se admite la entrega eventual de mensajes a destinos arbitrarios con suposiciones mínimas con respecto a la topología subyacente y la conectividad de la red subyacente. En este caso solo se necesitaría una conectividad periódica por los pares para garantizar la entrega final del mensaje. (8) (9) (6)

El protocolo de enrutamiento epidémico se basa en la distribución del mensaje de manera transitiva a través de redes ad hoc hasta su destino final. Cada host mantiene un búfer que consta de mensajes que ha originado, así como mensajes que está

almacenando en búfer en nombre de otros hosts. Para mayor eficiencia, una tabla hash indexa esta lista de mensajes, codificada por un identificador único asociado con cada mensaje. Cada host almacena un vector de bits, llamado vector de resumen, que indica qué entradas en sus tablas hash locales están configuradas. Algunos de estos puntos no se entrarán en muchos detalles ya que no se exploró durante este proyecto. (8)

Propuesta

El diseño a implementar en este trabajo es utilizar la tecnología LoRa en la banda de frecuencia de 868 MHz, permitida en la Unión Europea, para realizar conexiones con topología de red Mesh o Malla. Para lograr esta topología es necesario plantearse el uso de un enrutamiento Epidémico. Para lograr este enrutamiento se plantea diseñar un protocolo que realice la transmisión epidémica entre los nodos y asociará en los mensajes un identificador único para soportar el enrutamiento y tolerancia a fallas. Además, se incluirá un recuento de saltos para realización de pruebas y de manera opcional, solicitud de confirmación de recepción de mensajes. Como cada dispositivo debe poseer un identificador único (ID), este se generará de una fracción de la MAC de cada dispositivo. Como diseño para la implementación en IoT, se toma en considerar abordar el diseño en conseguir bajo consumo de potencia en el protocolo de comunicación, utilizando la capacidad de los dispositivos en entrar en estados de suspensión.

3. Análisis del problema

Diseñar un protocolo que permita una topología de red Mesh mediante tecnología de comunicación LoRa, esto mediante el uso de varios dispositivos de la marca pycom.

Realizar una comunicación efectiva de pares de dispositivos sin necesidad de considerar una ruta óptica, sólo es necesario cumplir la recepción del mensaje hasta el dispositivo final.

- **Enrutamiento en topología Mesh**

Para la realización de la topología malla con dispositivos LoRa se debe diseñar el protocolo de enrutamiento. Se debe plantear el modo de comunicación entre los dispositivos y la manera de realizar envío entre varios nodos. Partiendo de la utilización del enrutamiento epidémico el sistema debe tener la capacidad de comunicarse con todos los dispositivos para así llegar a su destino final.

- **Bucles de enrutamientos en el sistema**

Al realizar un envío de un mensaje a todos los nodos disponibles, estos verificarán el destino del mensaje y realizarán un reenvío de éste. Si el sistema no tiene capacidad de saber los mensajes elaborados o recibidos, al realizar el reenvío, muchos nodos pueden recibir el mensaje y procesarlo nuevamente, causando así un bucle de reenvíos y creando una inundación de mensajes en la red. El sistema debe tener la capacidad de discernir en los mensajes que se han recibido anteriormente, de esta manera se pueden evitar los posibles bucles presentes en el sistema.

- **Identificadores de dispositivos y mensajes**

Al realizar una conexión de red de dispositivos, cada uno de ellos debe saber cómo diferenciarse. Para eso es necesario la asignación de un identificador único. Este identificador como estará asociado en la estructura del mensaje debe poseer un tamaño apropiado para hacer ligero el tamaño del paquete. De igual manera que los dispositivos. Cada mensaje debe ser único, así el sistema puede diferenciar cada uno de los mensajes en tráfico. Para eso es ideal plantear un identificador para los mensajes, que permite la verificación en los dispositivos, donde se puede guardar en un buffer para los ID de los mensajes que son recibidos en el dispositivo.

- **Elaborar un código de verificación de mensaje**

Una de las condiciones que se puede establecer al realizar un envío de mensaje es la elaboración de envío de respuesta por parte del receptor. Esto se puede realizar de manera opcional para pruebas de efectividad de envío y latencia de la red.

El diseño de la verificación del mensaje puede ser uno a uno, donde cada paquete que un host envía debe esperar un paquete de respuesta. Esta manera, aunque es efectiva ya que aseguras que cada paquete es recibido, causa un mayor uso de los recursos de los host ya que se generan más paquetes que se encontraran en tránsito en la red. Una manera de reducir la cantidad de estos paquetes es realizar la respuesta del dispositivo para cada cierta cantidad de mensajes recibidos.

- **Diseño de filtro de dispositivos para pruebas y seguridad del sistema**

Un host debe tener la capacidad de limitar la recepción de mensajes de dispositivos no deseados. Ya que pueden existir host maliciosos o host que se encuentre comprometido. Para la implementación del protocolo es bueno realizar una consideración en la utilización de filtros para dispositivos específicos. Con este filtro se podría evitar recibir mensajes de un dispositivo no deseado y podría ser utilizado para realización de pruebas.

- **Reducir el consumo de potencia con estado de hibernación**

Al realizar la transmisión y recepción de mensajes se debe considerar el consumo energético que ocasiona. Muchos de estos dispositivos generan un mayor consumo al realizar una transmisión, aunque estas transmisiones son periódicas. Sin embargo, la recepción, aunque no genera tanto consumo, es estática ya que el dispositivo se encuentra en modo de escucha, donde espera recibir cualquier mensaje.

4. Solución propuesta

Con el planteamiento del problema ya expuesto, ahora se partirá en la manera de afrontar dicho problema para llegar a la solución. Para llegar a la solución del problema, se debe dividir el problema en fase e ir construyendo la estructura del programa desde la parte más básica. Para comenzar se comenzará estudiando la manera de realizar una comunicación simple de mensaje entre dos dispositivos y de esta base ir trabajando en las características que sean necesarias para cumplir con el objetivo. Se diseñará un plan de trabajo según estas características hasta llegar a la fase de prueba y se explicará el diseño de la solución por fases.

Plan de Trabajo

- Fases de desarrollo del código
    - Estructura de paquete: Inicialmente se debe plantear cuales son los datos que va a conformar el paquete. Es necesario considerar plantear un paquete que se adapte a las características de envío y no supere los límites del dispositivo.
    - Funciones de empaquetado y desempaquetado: se debe diseñar las funciones que serán invocados a la hora de realizar un paquete. A la hora de realizar un envío se debe suministrar la información para empaquetar y a la hora de recibir se debe lograr desempaquetar la información.
    - Estructura de envío y recepción de un paquete: De igual manera se debe diseñar las funciones que se invocaran en cada dispositivo para la realización de envío de mensajes y recepción de mensajes.
    - Asignación de ID para cada paquete a enviar: Cada mensaje o paquete que se envíe debe ser único. Se debe crear un sistema de etiquetado que permita asignar un identificador único a cada paquete.
    - Diseño de buffer para almacenaje de mensajes a enviar: Al recibir un grupo de mensajes los cuales pasarán a un estado de reenvío, el dispositivo debe tener la capacidad de ir recibiendo y encolando cada mensaje. Esto para no saturar el sistema de envío del dispositivo.
- Fase de realización de Prueba

- Código para el filtrado de dispositivos: Para la ejecución de pruebas se debe diseñar un sistema de filtrado de dispositivo. Esto permite crear conexiones de topología de forma de prueba ya que no se cuenta con la distancia necesaria para realizarlo en tiempo real.

- Diseño de diferentes topologías de red para prueba: Se plantean distintas formas topológicas que permitan probar el protocolo diseñado.

- Prueba y medición de consumo: Al realizar un protocolo destinado al uso de IoT, se debe considerar el consumo que pueda generar este protocolo a la hora de su implementación. Se deben realizar mediciones del consumo energético a la hora de la ejecución del protocolo.

Arquitectura del Sistema

Para el diseño del protocolo se debe partir de la comunicación simple entre dos dispositivos. Para esta comunicación entre ambos dispositivos, se debe realizar un envío de mensajes cada intervalo de tiempo. El sistema debe ser capaz de realizar el envío en la misma frecuencia de comunicación con el uso del estándar de LoRa y cada dispositivo debe tener la capacidad de comprender el mensaje que recibe. Para esto se debe poseer la misma estructura de mensajes para manejar un estándar entre los dispositivos.

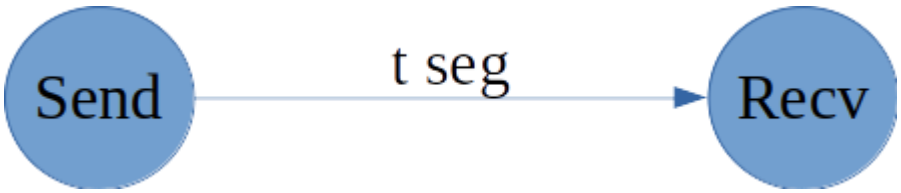

Ilustración 2 conexión simple entre dos nodos

Todos los dispositivos que pertenezcan a la red deben tener la capacidad de enviar y recibir, ya que la transmisión se debe realizar en ambos sentidos. Un dispositivo tiene como función de emisor como de receptor. Esto permite tener la capacidad de realizar respuestas de mensaje si fuera necesario.

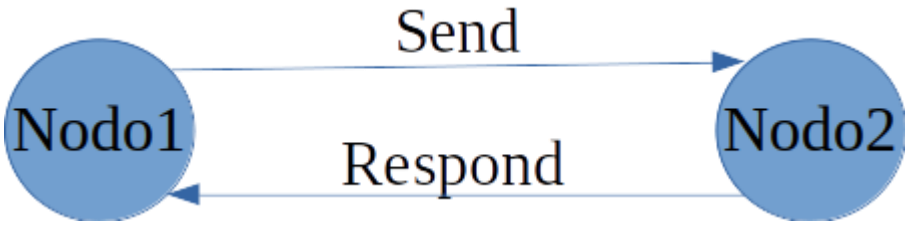

Ilustración 3 Envió y respuesta entre dos nodos

Al utilizar medios de difusión por radiofrecuencia, permite mantener conectividad con varios dispositivos a la vez. Mediante el uso de la tecnología LoRa se debe poder realizar comunicación con todos los dispositivos en su rango de frecuencia. Al realizar comunicación se corre el peligro que un nodo reenvíe un mensaje al nodo origen. Para esto el sistema se diseñará para tolerar los bucles en la red, mediante uso de lista de mensajes. Se llevará un registro de los mensajes que transiten por los dispositivos para así evitar ser recibido nuevamente.

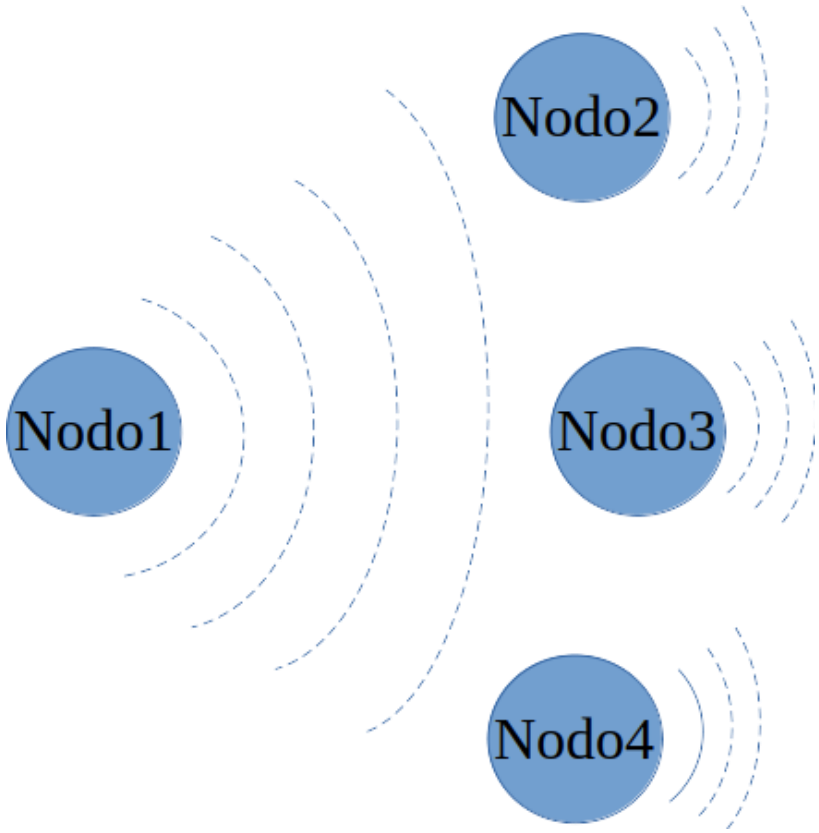

Ilustración 4 Comunicación por difusión de mensaje

Al realizar una red Mesh con nodos de la tecnología LoRa, el sistema debe tener la capacidad de enrutar un paquete hasta llegar a su destino. Si un nodo no tiene conexión directa al destino final del mensaje, éste realizará el envío a otros nodos que posiblemente se encuentren en su rango de transmisión.

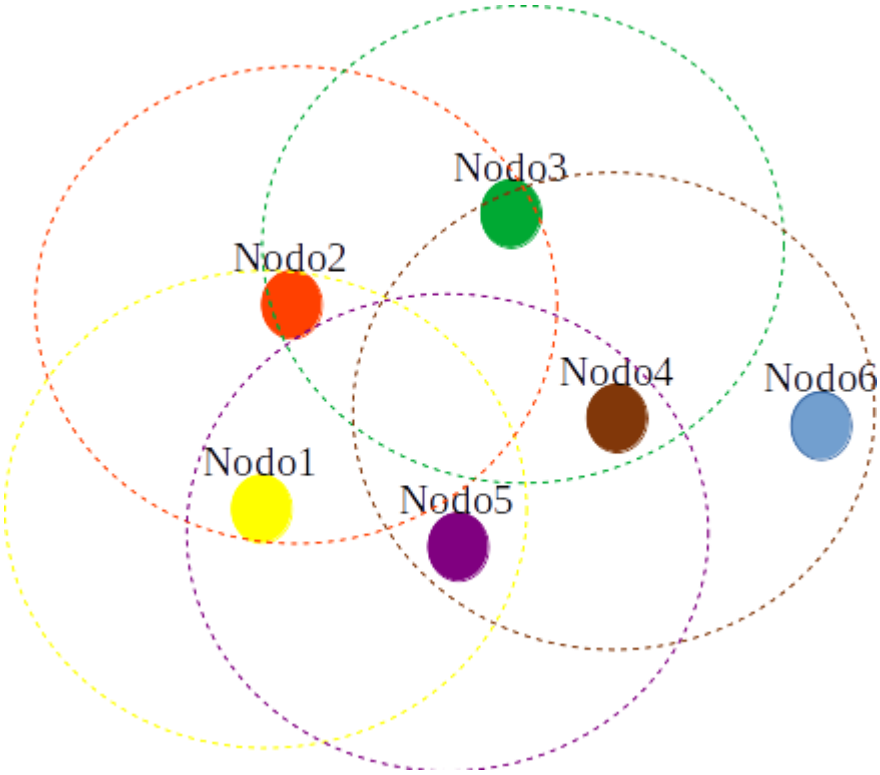

Ilustración 5 conexión de red Mesh entre nodos

Al ahora de escoger una ruta el sistema no gasta recursos para escoger ruta. Los paquetes serán enviados por diferentes rutas para así tener mayor éxito de llegar al destino. Esto consigue tolerancia a fallos en alguna ruta. De igual manera mientras el sistema no se encuentra en uso se colocará un tiempo de suspensión. Se sincronizará de tal manera que el sistema al recibir un mensaje pueda suspender sus funciones hasta el período de recibir el próximo mensaje.

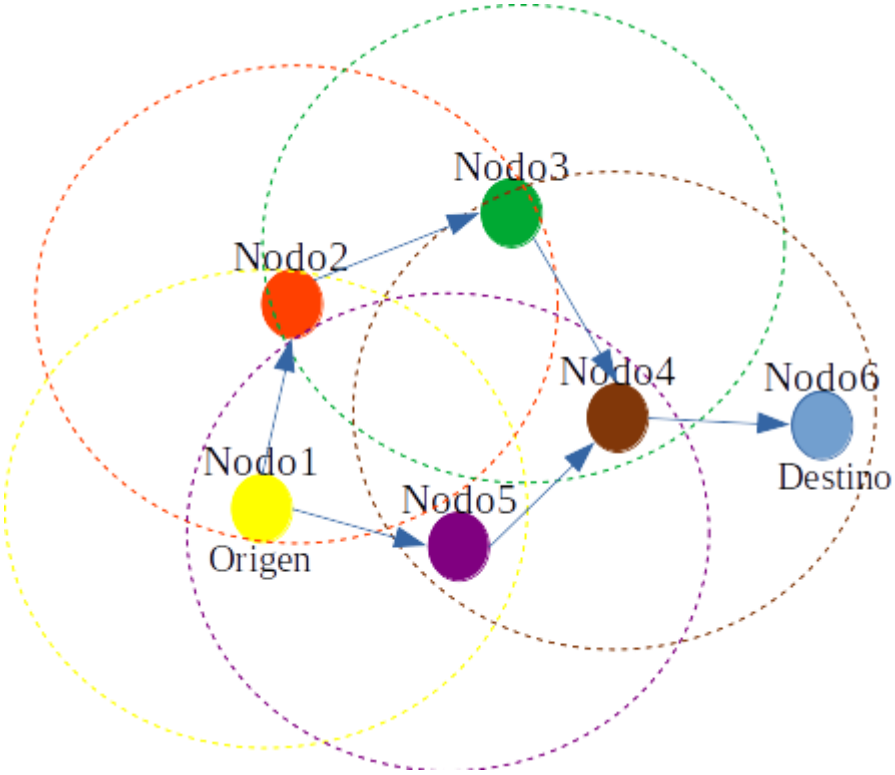

Ilustración 6 Enrutamiento epidémico entre los nodos

Diseño Detallado

Para el diseño del protocolo de comunicación entre dispositivos se realizará una división de por fases. Con la estructura de fase que se realizara a continuación se va realizando el código que será implementado y probado para cumplir los objetivos. Todo el diseño se realizara en lenguaje Python, ya que este es el soportado por los sistemas Pycom. La estructura de código se encontrara en el Anexo 1 y Anexo 2, y externamente se cargara en un repositorio de Github (https://github.com/damejias23/TFM_Mesh_Lora).

Fase I: Comunicaciones entre dos dispositivos (P2P)

Para iniciar una comunicación entre varios dispositivos se debe plantear inicialmente una comunicación básica entre dos dispositivos. Para eso en la primera fase, es importante realizar una comunicación de una sola vía entre dos dispositivos, donde se define la estructura básica del paquete y un tamaño inicial y se define la estructura de comunicación con la tecnología Lora.

La comunicación entre dispositivo se elabora inicialmente en una vía donde un dispositivo puede enviar un mensaje a otro dispositivo y poder leer el mensaje sin problema.

Fase II: Comunicaciones entre varios dispositivos

En la segunda fase al tener declarada la comunicación entre dos dispositivos, se debe incorporar la comunicación entre varios dispositivos. En esta comunicación se realizará de forma epidémica, donde un dispositivo envía un mensaje y será recibidos por igual a todos que se encuentre en su rango de alcance. Para conocer el destino al que se realiza los mensajes que envía un dispositivo, es necesario asignar un identificador único a cada dispositivo. Cada mensaje que se envíe debe especificar el ID que realiza el envío y el ID del destinatario del mensaje.

Fase III: Identificación entre los mensajes

Al realizar una comunicación epidémica entre varios dispositivos o nodos, se puede encontrar con el inconveniente de que el nodo final esté fuera del alcance. Para esto se realiza un enrutamiento, donde si un dispositivo no tiene el alcance a su destino, realizará el envío a todos sus nodos en su rango y a su vez estos intentarán reenviar el mensaje al nodo final o a otros nodos que puedan alcanzarlo.

Para evitar que un nodo reciba un mensaje desde varias rutas es necesario asignar un identificador único de mensaje. Esto permite verificar los mensajes que han pasado por el nodo y descartar los mensajes repetidos con igual identificador. En los nodos al generarse un mensaje se debe generar un número de ID aleatorio que no será igual a ninguno de los ID enviados por otros nodos, esto se logra colocando parte de la ID del nodo en el formato del ID del mensaje. Con esto se logra conocer el ID del nodo que realizó el envío.

Fase IV: Ciclo de respuesta de mensaje

Al realizar un envío de mensaje desconocemos si este mensaje llegó a su destino o si el mensaje sufrió una pérdida en el envío. Esto nos permite reducir costos en los recursos ya que el sistema solo debe enviar y no utilizar memoria para esperar una respuesta. Sin embargo, una comunicación no está completamente cubierta a pérdidas de paquete, por esto se puede utilizar medidas de comprobación para verificar si el dispositivo se encuentra en la recepción de los mensajes.

Para realizar la comprobación de mensaje se plantea realizarlo en la recepción de una cantidad específica de mensajes. De esta manera se logra disminuir el uso de recursos al realizar un envío. Realizar un envío y respuesta de cada mensaje causa un aumento considerable de los recursos y del canal, ya que la red debe soportar una capacidad de tráfico igual a la suma de los mensajes de envío y los mensajes de respuesta. Para reducir el consumo de la respuesta de los mensajes, se puede realizar una respuesta al recibir una cierta cantidad de mensajes de un nodo.

En esta fase se realizará el diseño del código que se encargará de la respuesta de los mensajes de los nodos. Para este caso este código se podrá activar o desactivar. De esta manera el sistema puede trabajar si se desea esperar una respuesta de los mensajes o sin respuesta de los mensajes.

Fase V: Ciclo de suspensión del equipo

El protocolo de comunicación propuesto, plantea una comunicación periódica, donde cada cierto tiempo el sistema realizará un envío de mensaje. El tiempo transcurrido al recibir otro mensaje, el sistema se mantiene activo esperando una recepción. Para esto se propone implementar una configuración que permite al dispositivo suspender su estado de actividad hasta recibir su próximo mensaje. Esto supone que los sistemas deben estar sincronizados, donde el tiempo de envío de mensajes y suspensión de mensajes debe ser el mismo. Un sistema se mantendrá activo para transmitir mientras contenga mensajes los cuales debe enviar. A la hora de no tener ningún mensaje que enviar, el sistema pasará a un estado de suspensión hasta que finalice el tiempo de inactividad.

Fase VI: Diseño de Pruebas

Para realizar las pruebas se debe realizar en un espacio reducido, lo que imposibilita colocar dispositivos fuera del rango de alcance para simulaciones. Para poder lograr con éxito estas pruebas se plantea diseñar un código que permita limitar los dispositivos por el ID. De esta manera se puede construir topologías de conexiones mediante restricciones en las ID entre los dispositivos. Esto solo se implementará de uso para la realización de pruebas.

El código de limitación de ID se puede desarrollar de tal manera que puede permitir ser usado para la seguridad de la red. En una red pueden existir dispositivos maliciosos o con la seguridad comprometida. Por eso es ideal que los dispositivos

cuenten con la posibilidad de filtrar dispositivos, ya sea denegar la comunicación con estos o permitir sólo la comunicación con ciertos dispositivos especificados.

Tecnología Utilizada

Para el desarrollo del protocolo de comunicación es necesario utilizar un dispositivo que cumpla como mínimo, un soporte de comunicación por Lora y recursos básicos de procesamiento y memoria. Principalmente se desea realizar un programa básico que no exige utilizar muchos recursos de cómputos para así diseñar un programa que realice el menor consumo de potencia en los dispositivos. Estos dispositivos a su vez, poseen la característica de generar un bajo consumo de potencia. A continuación explicaremos el hardware utilizado para el desarrollo de las pruebas y el software incorporado para su desarrollo:

Hardware

Pysense: es una placa con sensores que tiene la capacidad de ser utilizado con cualquier módulo de redes múltiples de Pycom. Se adapta a todas las placas Pycom y contiene una serie de sensores, como luz ambiental, presión y humedad. Incluye un conector de 6 pines para sensores Pynode.[2]

Características:

- Sensor de luz ambiental
- Sensor de presión barométrica
- Sensor de humedad
- Acelerómetro de 3 ejes y 12 bits
- Sensor de temperatura
- Puerto USB con acceso serial
- Cargador de batería LiPo
- Compatibilidad con tarjetas microSD
- Operación de energía ultrabaja (~ 1uA en sueño profundo)

Tamaño:

- Peso 0,036 kilogramos

[2] https://pycom.io/product/pysense-2-0-x/

- Dimensiones 5,5 × 3,5 × 1 cm
- Dimensiones: 55 x 35 x 10 mm

Ilustración 7 Dispositivo Pysense (Fuente de origen: Pycom)[3]

LoPy4: Es una placa portadora cuádruple programable por Micropython. Funciona con LoRa, Sigfox, WiFi y Bluetooth, lo que la convierte en una plataforma de IoT de nivel empresarial excepcional.[4]

Característica

- Potente CPU, BLE y radio WiFi de última generación
- También se puede duplicar como puerta de enlace Nano LoRa

[3] https://pycom.io/
[4] https://pycom.io/product/lopy4/

- MicroPython habilitado
- Cabe en una placa de pruebas estándar (con encabezados)
- Uso de energía ultrabajo: una fracción en comparación con otros microcontroladores conectados

Especificación LoRa

- Transceptor Semtech LoRa SX1276
- Pila LoRaWAN
- Dispositivos de clase A y C

Frecuencias operativas de LoRa

- 868 MHz (Europa) a 14 dBm como máximo
- 915 MHz (Norteamérica y Sudamérica, Australia y Nueva Zelanda) a 20 dBm como máximo
- 433 MHz (Europa) a 10 dBm como máximo
- 470 - 510 MHz (China) a 14 dBm como máximo

Especificación de la gama LoRa

- Rango de nodo: hasta 40 km
- Nano-gateway: hasta 22 km
- Capacidad de la nano-puerta de enlace: hasta 100 nodos

Interfaces

- 2 x UART, SPI, 2 x I2C, I2S, tarjeta micro SD
- Canales analógicos: ADC de 8_12 bits
- Temporizadores: 4_16 bit con PWM y captura de entrada
- DMA en todos los periféricos
- GPIO: hasta 24

Memoria

- RAM: 4 MB
- Flash externo: 8 MB

- Aceleración de punto flotante de hardware
- Python multiproceso

Poder

- Entrada: 3,3 V 5,5 V
- Salida 3v3 capaz de generar hasta 400 mA
- WiFi: 12 mA en modo activo, 5uA en espera
- LoRa : 15 mA en modo activo, 1 uA en espera
- Sigfox (Europa): 12 mA en modo Rx, 42 mA en modo Tx y 0,5 uA en modo de espera
- Sigfox (Australia, Nueva Zelanda y Sudamérica): 12 mA en modo Rx, 120 mA en modo Tx y 0,5 uA en modo de espera

Tamaño

- 55 mm x 20 mm x 3,5 mm (sin encabezados)
- Peso: 7g

Ilustración 8 Dispositivo Lopy4 (Fuente de origen: Pycom)[5]

[5] https://pycom.io/

Firmware

El firmware es la aplicación de bajo nivel de los dispositivos. Se encarga del control lógico de circuito en cada hardware. Para el diseño de este proyecto se realizó la actualización del firmware de los dispositivo LoPy4 con la versión Pycom MicroPython 1.20.0.rc13 [v1.9.4-94bb382] con fecha de publicación del 22-08-2019. Esta es la última actualización lanzada hasta la fecha de la publicación de este trabajo. Esto se debe a mejoras y correcciones de errores en los dispositivos y esto a veces significa actualizar el firmware de un hardware.

Software

Para el diseño del protocolo de comunicación es necesario utilizar un software que permita cargar las instrucciones que se desea emplear. Para esto se necesita un programa que sea compatible con la comunicación del dispositivo y que pueda monitorizar la información procesada. Para los dispositivos Pycom, la marca recomienda utilizar aplicaciones como Atom o Visual Code que cuentan con las funciones adecuadas para la conexión con sus dispositivos. Para la ejecución de este proyecto se utilizó la aplicación Atom, ya que cuenta con herramientas versátiles para el desarrollo del código en Python y con los paquetes necesarios para la programación y monitoreo de los dispositivos.

Atom es un editor de código fuente el cual cuenta con una gran variedad de complementos para realizar programación en diferentes lenguajes. Por esto Atom se convierte en una de los programas más completos e ideales para programadores. Soporta cualquier ordenador con sistemas Windows, Linux o Mac con macOS. Además, es un complemento perfecto para la misión principal de GitHub de crear un mejor software trabajando juntos.

Para la comunicación entre la aplicación de Atom y los dispositivos de Pycom es necesario instalar un paquete. En este proyecto se utilizó el paquete Pymakr Atom versión 2.1.13.

Pymakr le permite comunicarse con su placa Pycom usando la línea de comandos REPL incorporada y con esto se abrirá de forma predeterminada un panel para líneas de comando que permitirá ejecutar comandos en el dispositivo.

Desarrollo de la solución propuesta

Comunicación simple entre nodos y estructura de mensaje

En el diseño de comunicación es necesario designar la estructura inicial las variables y elementos necesarios para la comunicación en LoRa. Al realizar la llamada correspondiente de estos elementos es necesario definir la estructura inicial de los mensajes, para así diseñar un sistema que permita determinar el tamaño de paquete a recibir y diferenciar cada elemento del paquete. El sistema debe ser capaz de encapsular los datos correspondientes a enviar y al ser recibido por el nodo final desencapsular las variables manteniendo la estructura inicial.

Inicialmente para la comunicación en LoRa en los dispositivos de Pycom, es necesario incluir las librerías que permitirá llamar a las funciones de comunicación entre procesos y permite otras funciones que se utilizarán a lo largo del trabajo. Muchas de estas librerías se encuentran alojadas en el repositorio de memoria de los dispositivos. Para su uso solo se debe importar en las instrucciones de código que se desea implementar. Si alguna de las funciones de código no se encuentra en la memoria del dispositivo o se desea crear una con características específicas, se debe crear un repositorio de liberia que tendrá el nombre de ***lib.*** En esta carpeta se incluirán todas las funciones que se crearán para la comunicación, proceso de los mensajes, entre otras funciones que se irán explicando a lo largo del trabajo. Las librerías que se utilizaran y se encuentran incluido en el repositorio del dispositivo son:

- Con la librería ***Socket*** permite enviar mensajes a través de una red. Esta librería es básica para cualquier tipo de comunicación y proporciona una forma de comunicación entre procesos. Al formar parte de esta tecnología será necesario utilizar las funciones de Socket para establecer una comunicación.

- En ***Network*** se importará la función ***LoRa*** la cual permitirá que los dispositivos activen la comunicación por LoRa. La librería Network

permite proporcionar acceso a controladores de red y configuración de enrutamiento. Con la función LoRa se invocará los controladores de para esta tecnología en los dispositivos y se determinará los estándares y frecuencia en los cuales funcionará.

- Con la librería ***Time*** se podrá establecer comportamientos basados en el manejo del tiempo. Se puede representar el tiempo en código, como objetos, números y cadenas. Además, permite realizar esperas de ejecución en un tiempo el cual se dará uso en la aplicación y realizar mediciones de eficiencia en el tiempo.

- La librería ***Pycom*** permite ejecutar controladores para los dispositivos de la marca Pycom donde se incluyen los modelos Pysense, Pytrack y Pyscan. Además, contiene librería y ejemplos de diferentes códigos para poder implementar en los dispositivos.

- La librería ***Struct*** permite manejar datos binarios almacenados en archivos, bases de datos o desde conexiones de red. Con las funciones de struct se pueden compactar cadenas de formatos para una conversión en una estructura binaria.

- Con la librería ***Ubinascii*** se podrá implementar conversiones entre datos binarios y varias codificaciones de estos en forma ASCII.

- La librería **Machine** permite utilizar características de bajo nivel de los dispositivos. Con esta librería se pueden realizar modificación de la memoria flash del dispositivo Pysense, como borrar o restaurar memoria y también tiene la capacidad de apagar, reiniciar o suspender el equipo por un tiempo determinado.

Antes de establecer las funciones que se encargaran de ejecutar los controladores e invocar el canal de comunicación es necesario definir la estructura del mensaje. Lo ideal es plantear una estructura simple de paquete, ya que esta ocupa una cierta cantidad de bytes en los dispositivos y la relación del tamaño puede afectar la potencia necesaria para su transmisión. Para esto se establecerán los suficientes datos para lograr una comunicación en LoRa donde el mensaje llegue al destino final.

Para una estructura básica de un paquete es necesario almacenar el ID de los dispositivos que realizan el envío y que recibirá el mensaje. El ID de los dispositivos es un ID estático para cada uno y su asignación estará asociado a la máscara de MAC de

cada dispositivo. La MAC es un estándar de identificación para los dispositivos de red y para cada dispositivo, equipo o tarjeta de red, este identificador es único. Este identificador se expresa en hexadecimal. Sin embargo, como la MAC es un identificador de un tamaño de 48 bits solo se utilizará una parte de esta para reducir el tamaño del identificador. Para este trabajo se utilizó los primeros tres bits de la MAC leídos de derecha a izquierda y su valor se transforma a decimal. En la estructura del paquete se asigna un byte a cada ID tanto para el dispositivo de envío como para el dispositivo destino.

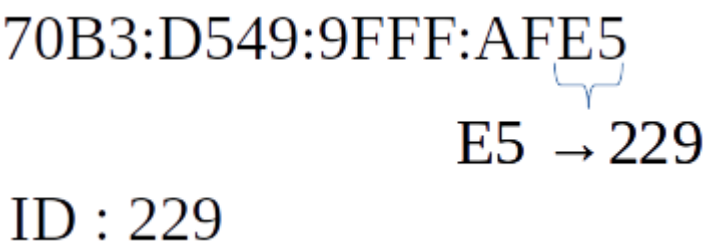

Ilustración 9 Formato del ID para los dispositivos

El ID de origen es un identificador que especifica el dispositivo de donde está saliendo el mensaje. Este identificador debe ser modificado y actualizado cada vez que entre a un dispositivo el cual realiza un envío, ya que, al recibir un paquete, este se convierte en un nuevo emisor de paquete. Esto permite conocer cuál dispositivo está realizando el reenvío. La información del identificador del nodo origen será almacenada en una parte del Identificador del mensaje.

Para obtener el ID de los dispositivos se diseñó una función llamada ***Def_ID*** que de valor de entrada se debe colocar la variable que contiene los datos del controlador de LoRa y mediante la librería de ***ubinascii*** se solicita el valor de la MAC del dispositivo. La función devuelve los primeros tres bits de la MAC leídos de derecha a izquierda en su valor se transforma a decimal. La estructura de la invocación de la función queda de la siguiente forma:

```
Def_ID(lora)
```

El siguiente dato importante que será agregado en la estructura del paquete es el ID de mensajes. Cada mensaje que realice un dispositivo para ser enviado debe ser único. Esto permite que los dispositivos reconozcan si un mensaje es nuevo o se ha

recibido anteriormente y así no causar bucles en el sistema. El identificador de mensaje se formará de dos partes, una parte será un número decimal con tres dígitos el cual se generará de forma aleatoria y la segunda parte será el ID del dispositivo. Con esto se logra que en cada ID de mensaje se encuentre el ID del dispositivo. El tamaño de memoria que será necesario para este identificador es de 4 bytes.

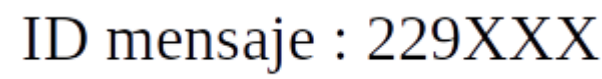

Ilustración 10 Estructura de ID de los mensajes

El ID del mensaje se creará mediante una función denominada ***UID_message*** que de valor de entrada recibirá el ID del dispositivo. Con el ID y con una función generadora de número aleatoria llamada ***Random,*** se crea un ID de mensaje y se retorna para ser anexado al paquete.

```
UID_message(id_device)
```

Como en el ID del mensaje se anexa el ID del dispositivo de origen, si fuera necesario se puede recuperar el ID del origen. Para esto se diseñó una función que recibe de valor el ID del mensaje y retorna el ID del dispositivo origen.

```
get_SendID(UID)
```

Al realizar los envío de paquete, se debe evaluar la cantidad de rutas que realiza hasta llegar a su destino. Así se puede conocer si la ruta escogida fue la más corta o la más larga. Para esto se agregara en el paquete un contador de saltos, que permitirá conocer cuántos saltos realizó el paquete hasta llegar a su destino. Este contador de salto también puede permitir descartar un mensaje. Si un paquete supera una cierta cantidad de salto, este paquete puede ser descartado para liberar la red, ya que supera la cantidad de salto para llegar a su destino. Esta cantidad de salto debe estar

relacionado a la cantidad de nodos en la red, ya que es la máxima cantidad de salto que puede hacer en la red.

Estos cuatros datos conforman el encabezado del paquete, los cuales cada dispositivo que recibe un paquete debe leer estos datos para conocer el destino y conocer si el paquete ha sido recibido anteriormente. Al tener el encabezado del paquete, solo queda agregar el mensaje a transmitir. Para fines prácticos de este proyecto se supondrá que la información que se recibe es la localización de cada dispositivo.

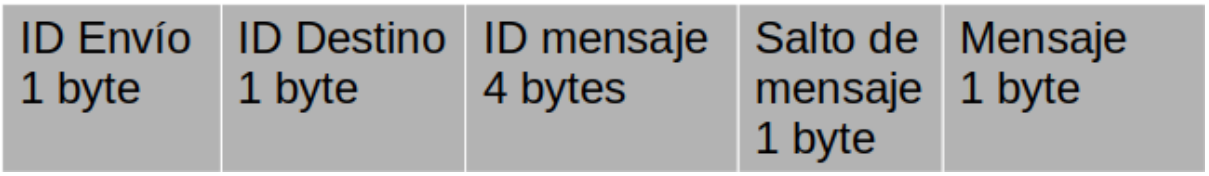

Ilustración 11 Estrucutra de Paquete de mensaje

Al realizar una respuesta de mensaje, esta se realizará de manera más simple. No es necesario anexar un mensaje en el paquete. Para esto solo se utilizara el encabezado. Al realizar un paquete de un tamaño reducido, el sistema por su tamaño lo considerará como mensaje de respuesta y con el ID del dispositivo se conoce su destino final.

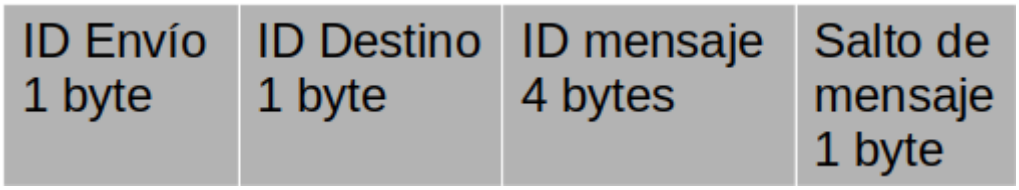

Ilustración 12 Estructura de Paquete de respuesta

Funciones de encapsulado y desencapsulado

Una vez diseñada la estructura del paquete se diseñan las funciones correspondientes que se encargan de encapsular la función en una estructura binaria de datos y al ser recibido por el nodo final, desencapsular esta información. El

encapsulado y desencapsulado se realizará usando la librería ***struct***. Con la funcion de struct, se define el tipo el formato de cada variable que se realizará el empaquetado y se agregan las variables a encapsular.

`b`	`signed char`	entero	1 byte
`B`	`unsigned char`	entero	1 byte
`i`	`int`	entero	4 bytes
`I`	`unsigned int`	entero	4 bytes

Ilustración 13 Tabla de valores de tipos de variables definidas

Se utilizará el signo “!” que representa el orden de bytes de la red que siempre es big-endian como se define en IETF RFC 1700 .

La estructura del paquete como se definió se forma por ID del dispositivo y ID del dispositivo de recepción el cual se asignará un carácter de un byte a cada uno, ID del mensaje el cual se asigna un entero de 4 bytes, contador de saltos que se asignará en un carácter de un byte y localización que se asignará en un carácter de un byte. Con esto la declaración e la variable estática de la estructura del paquete queda de la siguiente forma:

```
_LORA_PKG_FORMAT = "!BBIBB"
```

Para la estructura del paquete de respuesta se declara con el ID del dispositivo y ID del dispositivo de recepción el cual se asignará un carácter de un byte a cada uno, ID del mensaje el cual se asigna un entero de 4 bytes, contador de saltos que se asignará en un carácter de un byte. De esta forma la declaración de la variable estática de la estructura del paquete queda con el siguiente formato:

```
_LORA_PKG_ACK_FORMAT = "!BBIB"
```

Para la estructura de las funciones de encapsulado se debe colocar de entrada los datos que se van a empaquetar y la función retorna el paquete de caracteres binarios que contiene la información. En el caso contrario que se desea desencapsular, la función debe tener como datos de entrada el paquete y su tamaño, la cual procesa y retorna los valores de los datos enviados. La estructura de las funciones a invocar queda de la siguiente forma:

- Encapsulado de mensaje: `pack_Lora(DEVICE_ID,Device_next, UID_msg, TTL, location)`
- Encapsulado de mensaje respuesta: `pack_AckLora(DEVICE_ID, DEVICE_Resp, UID_msg, TTL)`
- Desencapsulado de mensaje: `unpack_Lora(recv_pkg, recv_pkg_len)`
- Desencapsulado de mensaje respuesta: `unpack_AckLora(recv_ack, recv_pkg_len)`

En el diseño del enrutamiento de los paquetes, los nodos intermedios deben verificar el destino del paquete. Para esto es necesario verificar el encabezado del paquete. Se diseñan dos funciones que se encargará de extraer el encabezado, verificarlo, realizar las modificaciones para su envío y nuevamente agregar el encabezado del paquete.

Se elabora una función la cual tiene como dato de entrada el paquete y su tamaño, y tiene como objetivo devolver, los datos del encabezado, que en este caso retorna los ID del dispositivo de origen y de destino, el ID del mensaje y el contador de salto del mensaje. La invocación de la función quedaría de esta manera:

```
get_IDdest(recv_pkg, recv_pkg_len)
```

Una vez que el nodo verifique la información del encabezado y catalogue como un paquete de reenvío, se debe realizar la modificación ID del nodo de envío, ya que este nodo se convierte ahora como nodo origen. La función contiene la capacidad de analizar si un paquete es de respuesta o no, ya que un paquete de envío a un paquete de respuesta posee diferentes tamaños y por ende, la función debe realizar diferentes operaciones a cada uno. Además, el contador de salto se suma en uno para llegar al próximo destino. La estructura de la invocación de la función queda de la siguiente manera:

```
set_IDorigen(pack, ID_sent)
```

Con el planteamiento básico de la estructura del mensaje, las librerías definidas y las variables declaradas se puede comenzar con el diseño del programa principal. Inicialmente se parte de un programa simple de envío y recepción entre dos nodos. Como el paquete se estará enviando periódicamente se debe establecer un proceso de ciclo, donde un nodo se encontrara enviando y el segundo nodo se encontrará escuchando. A su vez, se debe asignar un reloj para colocar un tiempo de espera al sistema. Con el planteamiento basico de la estructura el diseño del codigo quedaria de la siguiente forma:

Para envío:

```
while(True):
pck = pack_Lora(id_device, ID_send, UID, location, Px_Rx)
lora_sock.send(pck)
sleep(15)
```

Para recepción

```
while(True):
recv_pkg = lora_sock.recv(256)
if (len(recv_pkg) > len(_LORA_PKG_FORMAT) -1):
id_to_send, id_end, UID_msg, location, My_px = unpack_Lora(recv_pkg, recv_pkg_len)
```

Con esto se logrará el simple objetivo de conseguir enviar y recibir entre dos dispositivos en una sola dirección. Para esta configuración. En esta configuración se puede agregar más dispositivos, pero no existe una configuración de diferenciación entre paquetes y si un dispositivo se encuentra fuera de rango no existe la posibilidad de realizar reenvíos de paquetes.

Comunicación y enrutamiento entre varios dispositivos

Al tener una estructura básica de envío y recepción de mensajes, se puede trabajar ambos sistemas por separado. De esta manera se hace más optimo trabajar cada una en una función de forma distinta e independientes entre sí. Para poder enviar un paquete solo sería necesario invocar la función *Send*, la cual asignando un ID de destino, se realizará el envío de un mensaje periódicamente con los datos de localización. Para recibir un mensaje de un dispositivo se necesitará la función *Receive* que se encontrará siempre activo escuchando y recibiendo los paquetes enviados en la frecuencia especificada.

Cuando un dispositivo recibe un paquete, éste debe realizar la verificación si este paquete le fue asignado o es para ser reenviado. Un dispositivo puede recibir un paquete con destinos a otros dispositivos debido a la configuración de enrutamiento epidémico, donde un mensaje se envía a todos los dispositivos que se encuentran en su rango de alcance. Para esto es necesario verificar el destino final, para saber si el mensaje se debe recibir, descartar o reenviar. Para realizar la verificación solo es necesario comparar la ID final del mensaje con la ID del dispositivo receptor. Si las ID son iguales el mensaje llegó a su destino, si no coincide el mensaje se debe enviar o descartar.

```
            if(id_end == id_device):
                id_to_send, id_end, UID_msg, location, My_px  = unpack_Lora(recv_pkg,  recv_pkg_len)
                print("Mensaje  para  mi  de  %d"  % get_SendID(UID_msg))
```

Si el paquete que se recibe no posee la ID del dispositivo, este paquete será reenviado a otros dispositivos para intentar llegar a su posible destino o intentar que otro nodo llegue al destino. Cuando se realiza un reenvío de paquete se debe actualizar el ID del dispositivo de envío. Cada dispositivo que envíe debe colocar su ID con identificador de envío. Esto permitirá que los dispositivos receptores conozcan el nodo de quien realiza el envío. Igualmente, el contador de salto debe sumar un valor más a la cantidad de saltos que lleve, así se conocerá al final del destino la cantidad de saltos realizados desde el origen.

```
id_to_send, id_end, UID_msg, TTL = get_IDdest(send_pck)
            print("Reenvio de: %d" % id_to_send)
            pck_reenvio = set_IDorigen(send_pck, id_device)
            lora_sock.send(pck_reenvio)
```

Identificación de mensaje

Cuando tienes configurada una red de nodos configurada en un enrutamiento epidémico, se puede dar el caso que un mensaje llegue varias veces al mismo dispositivo y por consiguiente se crea un bucle en la ruta de ese mensaje. Para evitar estos bucles, los dispositivos deben tener la capacidad de descarte de mensajes. Si el dispositivo conoce si un mensaje ha sido recibido anteriormente, permitirá eliminar los bucles que se puedan formar a causa de otros dispositivos. Esta capacidad de filtrar se debe lograr con la identificación de mensajes. Los mensajes poseen un ID único que permitirá a los dispositivos llevar un historial de los mensajes que han sido recibido y descartar los recibidos anteriormente.

Los dispositivos deben asignar un buffer donde almacenan un historial de los ID de los mensajes recibidos. Cada vez que se recibe un mensaje, el sistema compara el ID con su historial. Si en el ID se encuentra en listado, este mensaje es descartado. En caso contrario de no encontrarse el identificador, el sistema aceptará el mensaje, verificará el ID del destino y actualizará el historial, agregando el ID del mensaje recibido. La condición de verificación y listado de ID del mensaje se debe agregar en la función de recepción de mensaje. Cada mensaje que se reciba se extrae el ID del mensaje, se realiza una comparación y se verifica si el mensaje debe ser descartado o debe continuar el proceso y agregar esta nueva ID a la lista.

```
if UID_msg in list_recv:
    return
else:
    list_recv.append(UID_msg)
```

Una consideración que se toma en cuenta es la generación de un nuevo mensaje. Cuando un dispositivo genera un mensaje y realiza el envío, puede presentar el inconveniente de recibir nuevamente su mensaje de otro dispositivo donde si se da el caso, el dispositivo nuevamente reenviará su propio mensaje. Para evitar este problema planteado, el sistema al crear un nuevo mensaje y asignar una ID, ésta deberá ser incluida en el historial de ID para prevenir recibir nuevamente su propio mensaje. Con esto se completa la posibilidad de presentar algún bucle de mensaje.

Al agregar un nuevo elemento a la lista de historial de ID de mensaje, se estará ocupando una parte de la memoria del dispositivo. Al no llevarse una controlada gestion de la memoria de la lista, puede darse el caso que la lista llene la capacidad de la memoria y sature el sistema. Por esto, es necesario diseñar una gestión de memoria del listado. Para llevar un mejor control de esta lista, se realizará una limpieza periódica de esta parte de la memoria para así evitar posible saturación del sistema. Se realizará con un contador de tiempo donde cada 200 segundos el sistema borrará por completo la lista y permitirá nuevos ingreso al sistema. Con 200 segundos se consigue el suficiente tiempo para que todos los mensajes enviados lleguen a su destino y permita recibir luego nuevos mensajes.

```
def clear():
    global list_recv
    time.sleep(200)
    list_recv = []
```

Sistema de Buffer de mensaje

En un sistema donde se presentan dos funciones independientes de envío y recepción, se pueda dar una limitación de la capacidad de datos procesados. Esto se debe a que el sistema solo puede enviar y recibir de forma secuencial. Si el sistema se

maneja de manera secuencial al enviar un dato, limita el número de nodos con quien se puede comunicar. Se da el mismo caso con la recepción de mensajes, donde el sistema debe limitarse a enviar sus propios mensajes o reenviar los mensajes de otros nodos. Se debe pelear por el medio para lograr la transmisión.

Para conseguir un mejor provecho al sistema, se plantea agregar un buffer de mensaje, el cual permitirá almacenar los mensajes y procesarlos para su envío. Con esto el sistema de envío seguirá funcionando de manera secuencial, sin embargo, los paquetes se irán encolando y enviando uno a uno hasta que la cola se encuentre vacía.

Planteando un sistema de buffer de mensajes, permite al dispositivo enviar mensajes a diferentes dispositivos al mismo tiempo. Además, cada mensaje que se desea enviar se procesa y se anexa en la cola para ser enviado a su destino.

El sistema de buffer se debe implementar tanto en las funciones de envío y recepción. Para la función de envío se realiza una división. Se asigna una función que se encargará de ir recibiendo los mensajes de la cola y realizar el envío y la segunda función cumplirá el objetivo de agregar a la cola los mensajes del dispositivo a comunicar. Con esta última función se puede lograr enviar varios mensajes a varios dispositivos al mismo tiempo, realizando varias invocaciones de esta función con destinos diferentes. Para el sistema de reenvío de mensajes, se conecta su ejecución con la función de envío. Todo paquete que se desea enviar, será incluido en la cola para posteriormente ser enviado, logrando una mejora en el comportamiento del sistema.

Respuesta de mensajes para comprobación

Un sistema de comunicación y transporte puede ser visto configurado de dos maneras, con comprobación de envío de mensaje o sin comprobación de esta. Al realizar un sistema con comprobación, se asegura que el mensaje es recibido por el destinatario y en caso contrario de no ser recibido se pueden realizar reenvíos del mensaje. Esta manera aunque es la ideal para poseer seguridad de la comunicación, requiere más recursos para su implementación, ya que el sistema debe esperar una respuesta para dar de exitoso el envío. Además, se debe considerar una carga extra más al medio de comunicación, donde se presentarán paquetes de mensajes y comprobación de mensajes. En un sistema sin comprobación se desconoce si el mensaje llegó a su destino. No existe una seguridad de la recepción del mensaje, sin embargo, la carga de información es más ligera ya que no es necesario generar carga para la espera de comprobación y el medio de comunicación tiene menos mensajes que procesar.

Para la implementación de este trabajo se planteó la idea con un sistema sin comprobación de mensajes. Esto permite la ligereza en el código del sistema y evitar saturación del buffer de los dispositivos. Sin embargo, de manera práctica para realización de pruebas se diseñó un sistema de respuesta que puede ser activado o desactivado según el uso que se desea dar. Para evitar saturar el medio, el sistema no responderá cada uno de los mensajes recibidos. La respuesta se realizará cuando el sistema reciba una cantidad de mensaje específico de dicho dispositivo. A su vez el dispositivo emisor recibirá la recepción de sus mensajes y esperará un tiempo estipulado hasta recibir la próxima respuesta.

Para el diseño de esta etapa se creará una clase que se encargará en los dispositivos ir contando los mensajes recibidos por cada dispositivo. Cuando un mensaje nuevo llega a un dispositivo, éste agrega en su lista el ID del dispositivo emisor y crea un contador para este. Los próximos mensajes recibidos deben sumar un valor más al contador hasta un número finito de mensajes. Al llegar a la cantidad definida el sistema retira de la lista el ID del dispositivo y genera el envío de respuesta al dispositivo emisor. El emisor se encontrará esperando el mensaje de respuesta y en caso de no recibir el mensaje en un tiempo esperado, el sistema asume que no está llegando a su destino.

En la comprobación del sistema se creó una clase denominada `ID_recv` que estará formado por el ID de los dispositivos recibidos y un contador. A su vez, se crean dos funciones las cuales se encargan de ir contando la cantidad de mensajes recibidos y la segunda función se encargará de avisar si no se recibe la respuesta de comprobación de mensaje. Para el listado se creó la función `List_ack` que de variable de entrada tiene el ID del dispositivo emisor y la lista de dispositivos para ir realizando el conteo del emisor. De salida la función devuelve *False o True*, si el dispositivo debe realizar el reenvío o no. Mientras el dispositivo no llegue a la cuenta devolverá un *False* y al llegar a la cuenta la función devolverá *True.* Para la comprobación de la recepción del mensaje se utilizará una función denominada `ack_off,` la cual no posee datos de entrada y salida. Solo será una función que llevará una cuenta atrás esperando recibir la respuesta. Al recibir una respuesta el contador se reinicia. En caso contrario y el contador llegar a cero, esta función avisa que no se está recibiendo la comprobación del mensaje.

Suspensión del sistema para ahorro de energía

Para el sistema de suspensión, se utilizó una de las funciones propias del dispositivo Lopy4. Los dispositivos Lopy poseen en su memoria flash una librería conocida como **machine**. Con esta librería se puede controlar el estado del dispositivo. Se puede realizar desde esta librería modificación de la memoria y suspensión o sueño profundo del dispositivo. Para la implementación de este proyecto se utilizó el manejo del estado de suspensión.

La función de suspender el estado consiste en congelar todo funcionamiento que realice el dispositivo a la hora del llamado. No es un estado de apagado o de reinicio. Solo detiene todos los procesos en ejecución y los mantiene hasta el tiempo que fue configurado. La utilización de este estado fue la mejor para su implementación, ya que el sistema no podía reiniciar los procesos en ejecución. Se debía mantener todos los recursos utilizados en el momento y luego de la suspensión continuar con su funcionamiento. Por eso, no era recomendable utilizar la función de sueño profundo, ya que con esta se finalizan todos los procesos y luego de la activación se deben levantar nuevamente desde el inicio. La función utilizada para la ejecución fue:[6]

```
machine.sleep(time  , resume_wifi_ble = (True/False))
```

Para el desarrollo del sistema se planteó mantener un estado activo de recepción/emisión mientras recibe o emite uno o varios mensajes. Al realizar su funcion, el sistema pasa a su estado de suspensión por un tiempo asignado. Al pasar este tiempo el sistema vuelve a continuar los procesos y realiza su envío o recepción de mensajes nuevamente. Si el sistema recibe o emite varios mensajes, éste se mantendrá activo hasta que no exista mensajes en su buffer para ser enviado.

Es importante considerar en este punto que los nodos deben encontrarse en sincronización para recibir y enviar mensajes. Antes de entrar en el estado de suspensión, el sistema tiene un periodo de espera para culminar la recepción y emisión de mensajes. De igual manera sucede al activar luego de una suspensión, el sistema permite un tiempo ante de enviar un mensaje para que todos los dispositivos se levanten. Para esto todos los dispositivos deben estar sincronizados en estos tiempos. Al entrar un nuevo dispositivo en la red, este se mantendrá activo hasta recibir el primer

[6] https://docs.pycom.io/tutorials/networks/lora/

mensaje de cualquier nodo. Luego de esto, el dispositivo se sincronizará a la red sin problemas.

	T1	T2	T3	T4	T5	T6	T7	T8
Nodo 1		SEND					SEND	
Nodo 2		RECV/SEND					RECV/SEND	
Nodo 3		RECV/SEND					RECV/SEND	
Nodo 4		RECV					RECV	

	Tiempo activo
	Tiempo suspendido

Tabla 2 Tabla de actividad de suspensión de los dispositivos en el tiempo

5. Implementación

Para la implementación y ejecución del código diseñado en el diseño de enrutamiento en Lora, será necesario utilizar los dispositivos LoPy4 con las placas base de Lopysene. El diseño se plantea para ser utilizado en dos o más dispositivos de esta gama.

El código del programa está estructurado en cinco funciones principales. Cada una de ellas elaboran procesos distintos que serán necesarios para lograr realizar la comunicación con otros dispositivos. Estas funciones se ejecutan cada una en subproceso independientes, mediante uso de hilos. En tres de las funciones, se encargará principalmente de realizar el envío y recepción de mensajes. Las dos funciones restantes se encargarán en operaciones secundarias para verificación de respuesta de mensajes o limpieza de buffer del sistema.

Para el sistema de envío y recepción se diseñaron tres funciones que, aunque cada una es independiente, deben trabajar en conjunto para llevar a cabo la operatividad del sistema. Para realizar la recepción de los mensajes se dispone de una función de la librería LoRa, esta función que se conoce como ***lora.callback*** permite al dispositivo mantenerse activo a cualquier cambio en la frecuencia de transmisión. El dispositivo al recibir un evento del medio ya sea por transmisión de un mensaje o recepción de un mensaje, activará un estado en el dispositivo y ejecuta la función correspondiente del estado. Para esta implementación del sistema, solo se utilizará el estado de recepción, ya que la transmisión se realizará partiendo de otra función. Al recibir un mensaje por el medio de transmisión, el sistema ejecuta la función ***Recv***. Con esta función el dispositivo evaluará el encabezado del mensaje para conocer su destino. Si el mensaje es para este dispositivo, desencapsula el mensaje y recibe su información, en caso contrario de pertenecer a otro dispositivo, esté lo agrega a un buffer para pasar al sistema de envío. La función implementada y que se ejecutará en un subproceso para el recibir mensaje es la siguiente:[7]

[7] https://docs.pycom.io/tutorials/networks/lora/

```
lora.callback(trigger=(LoRa.RX_PACKET_EVENT|LoRa.TX_PACKET_
EVENT), handler=lora_cb)
```

Para la emisión de mensajes desde el dispositivo se utilizarán dos funciones. El sistema se diseñó basado en buffer de mensajes, donde los paquetes que se desean enviar son agregados a un buffer de memoria y luego retirados uno por uno para su entrega. Este buffer funcionará en FIFO (*First In, First Out*) y para ello se diseñó la función `Send_buffer`**.** Esta función se ejecutará en un hilo para trabajar de forma independiente de los demás procesos. Cuando el sistema desee enviar un mensaje bajo la función `Send` o enviar un mensaje desde la función `Recv`**,** deberán colocar el mensaje en el buffer. La función del buffer, al evaluar el destino del mensaje desde su encabezado, debe realizar modificaciones de este si es necesario, para el caso de los reenvíos. El subproceso se ejecutará de la siguiente forma:

```
_thread.start_new_thread(Send_buffer, ())
```

Para la función de `Send`, el objetivo es preparar el mensaje con quien se desea comunicar el dispositivo. La función tiene como valor de entrada la ID de un dispositivo y con esta prepara el mensaje donde el encabezado colocara esta ID como destino del mensaje. Esta función está programada para realizar el envío de mensaje en cada cierto tiempo, programando para efectos prácticos de este trabajo en 15 segundos entre cada mensaje que se desea enviar. La función `Send` como se muestra más adelante, está diseñada para ejecutar un subproceso de la función `Packet_buffer`**.** La ventaja de realizar de esta manera es poder realizar varios envíos al mismo tiempo ejecutando la función cuantas veces sea necesario con la especificación de ID.

```
def Send(ID_send):
    _thread.start_new_thread(Packet_buffer, (ID_send, ))
```

Cuando se recibe un mensaje de un dispositivo, está almacena el ID del mensaje. Esto se realiza para evitar bucle en el sistema. Ahora bien, al almacenar los ID en la memoria del equipo se debe estar pendiente de no ocupar todo el espacio de memoria. Para eso se implementa un sistema de limpieza, que a una cierta cantidad de

tiempo transcurrida el sistema borra el historial de ID de mensaje en la memoria. Para esta ejecución se utiliza un subproceso que ejecuta la función ***clear***.

```
_thread.start_new_thread(clear,())
```

La última función para utilizar en forma de subproceso es la función `ack_off`. Esta función sólo se activará en caso de realizar sistema de comprobación de mensajes, en donde la función llevará un contador hasta la espera de respuesta del dispositivo final. Si al finalizar el contador no se recibe una respuesta del dispositivo final, este activará una bandera y para efectos visual se iluminará el dispositivo con un led en rojo. En caso contrario de recibir una respuesta del dispositivo, el contador se reinicia y vuelve a esperar hasta recibir otro mensaje de respuesta.

```
_thread.start_new_thread(ack_off, ())
```

Pruebas

Para la realización de las pruebas de la propuesta, es necesario diseñar en el código una función para realizar test. Este sistema está diseñado para operar a largo alcance. Al poseer los dispositivos en un laboratorio, es complicado realizar las pruebas en un ambiente ideal. Para esto es necesario realizar un modo de prueba en el sistema que permita limitar la conexión entre nodos.

En el diseño del modo de prueba se implementará una opción de activación y desactivación de este modo. Así, si desea implementar el sistema en un ambiente ideal se puede utilizar sin ningún inconveniente. El modo de prueba se plantea trabajar como filtro de identificadores de dispositivos. El sistema poseerá una lista donde se colocará las ID de los dispositivos que no pueden recibir mensajes. El dispositivo al recibir un paquete de cualquier nodo, éste evalúa la ID de origen y compara con su lista negra. Si el dispositivo se encuentra en la lista, el paquete es descartado. En caso de no encontrarse en la lista, el mensaje continúa el proceso de verificación de paquete.

La condición de modo de prueba es un código que será incluido en la función de `Recv`, ya que en esta es donde se realiza la recepción de mensajes. Al activar el modo

de prueba el sistema hace una simple comparación en la lista. La linea de codigo de comparación quedaría de la siguiente forma:

```
if(modeTEST):
    if id_to_send in ID_block:
        return
```

La lista es un archivo de texto en el cual se incluirá los dispositivos que no tiene permitido recibir. Esta lista se diseñó de tal manera que cada dispositivo posee su propia lista. El dispositivo al iniciar el proceso lee un archivo el cual está identificado con su ID. Guarda cada ID en el archivo en un arreglo y con este arreglo es que se realizará la comparación. Para la lectura del archivo, se creó una funcion llamada `Open_test` donde la función recibe de entrada el nombre del archivo y devuelve el arreglo con las ID a denegar.

Para realización de prueba se plantea utilizar tres topologías. En cada una de estas topologías se intentará evaluar la capacidad de soportar cualquier modo de conexión y su flexibilidad de uso. Además, en cada uno se evaluarán diferentes características de tolerancia. Para las prácticas se plantea utilizar una topología de anillo, una topología punto a punto y por último una topología de malla.

La topología de anillo es una de las topologías más sencillas de realizar, donde cada dispositivo solo puede ver a un nodo en un sentido. Con esta topología se puede realizar pruebas de enrutamiento. Donde se verificará los saltos que realiza hasta llegar a su destino y la posibilidad de comunicarse los dispositivos entre todos al mismo tiempo. En esta topología se medirá también el consumo de energía que genera el envío de un paquete y además se medirá el consumo en el estado de comprobación de mensajes y modo de suspensión del sistema. Con esta topología se podrá realizar la mayor cantidad de pruebas del funcionamiento del sistema.

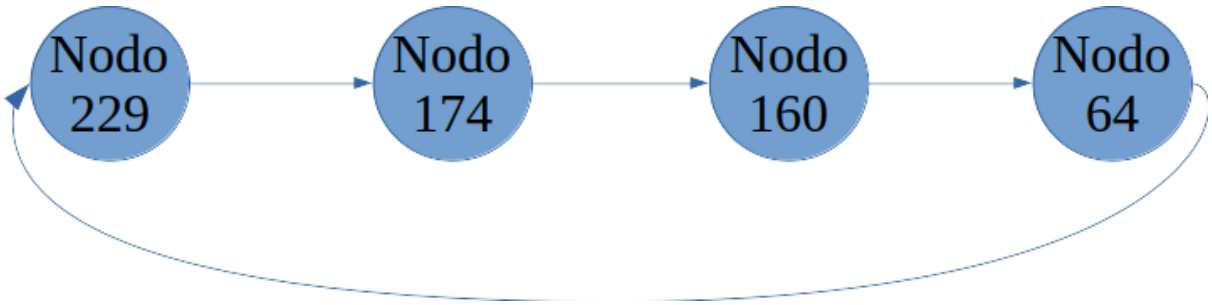

Ilustración 14 Diseño de prueba para topología en anillo

En la topología punto a punto, se plantea realizar pruebas de separación de enlaces. Se configura el enlace en parejas y se comprobará la independencia del sistema al recibir mensajes de dispositivos no permitidos. Para esta prueba se intentara de generar varias ejecuciones de envío para medir la tolerancia a la recepción de varios mensajes.

Ilustración 15 Diseño de prueba para topología punto a punto

Por último se realizaron pruebas con una topología de malla, donde todos estarán enviando mensajes entre todos y veremos la capacidad de soportar los mensajes y el comportamiento si hay un nodo saturado. Se medirán los saltos que realizan hasta llegar a su destino para comprobar si utilizan la ruta óptima.

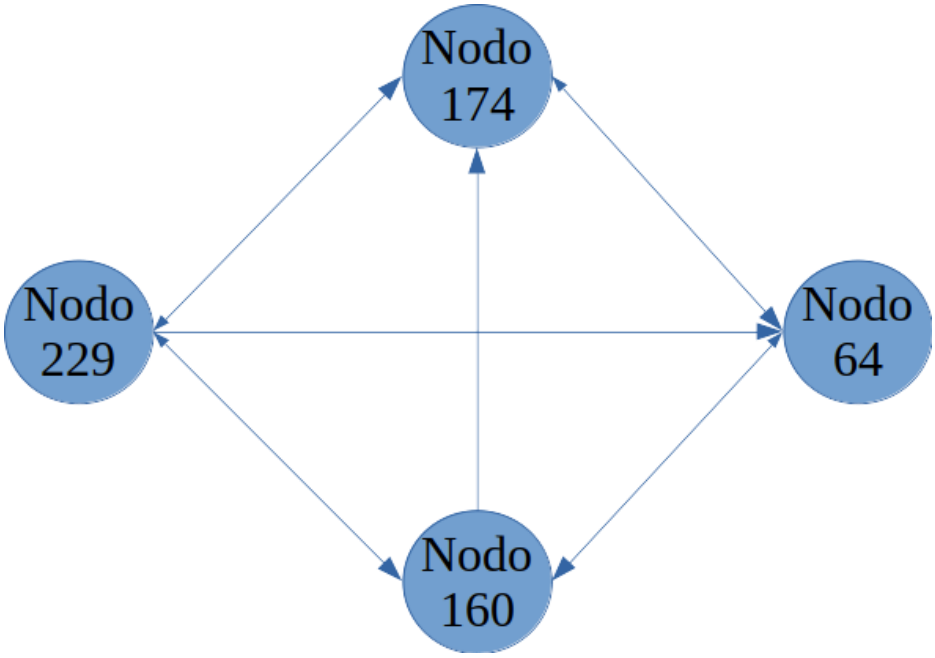

Ilustración 16 Diseño de prueba para topología Mesh

Resultados

Pruebas realizadas en topología anillo

Para las pruebas realizadas en la topología anillo se comenzó evaluando la entrega éxito desde un nodo a un extremo. Para este ejemplo se realizó la comunicación entre el nodo 229 al nodo 64, el cual se encuentra en los extremos y deberán pasar por los nodos 160 y 174 para llegar al dispositivo final. Se preparó un archivo para cada dispositivo donde se agregó las ID de los nodos que no pueden realizar comunicación y se puso en marcha la aplicación. Con el comando `Send(#ID)` se comenzaba la comunicación con los nodos. En él no 229 se realizó la comunicación con el nodo 64 con el comando `Send(64).` Al comenzar la transmisión el nodo 229, envía su mensaje a todos los nodos, sin embargo, el único que tiene permitido escucharlo es el nodo 160. Éste a su vez verifica el destino del mensaje y realiza el reenvío del paquete a otro próximo nodo que en su caso sería el 174. Al realizar el reenvío nuevamente hasta el nodo 64, éste verifica los datos y reconoce que es un paquete suyo. Realiza el desencapsulamiento del mensaje y muestra el remitente del mensaje y los saltos que realizó para llegar a su destino. En las siguientes imágenes se puede mostrar los resultados que los dispositivos arrojaban al recibir un mensaje. Para el dispositivo de origen como para el dispositivo final se lleva un contador de la cantidad de mensajes enviados y recibidos, para ir midiendo sincronización y pérdidas de paquetes, los cuales en esta prueba no se evidencio ninguna pérdida de paquete.

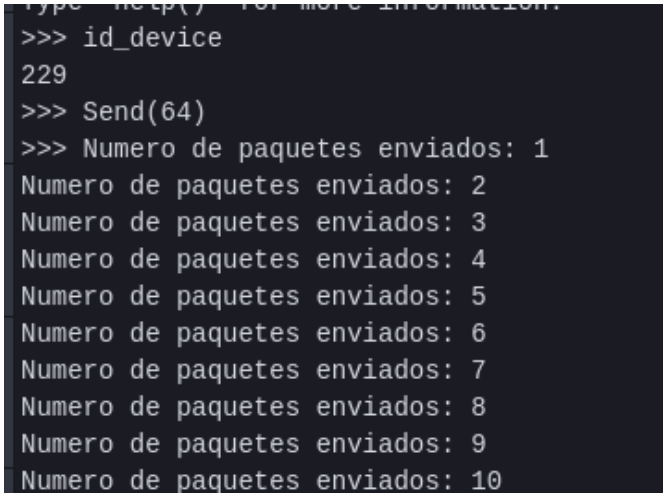

Ilustración 17 Muestra de envió del dispositivo 229

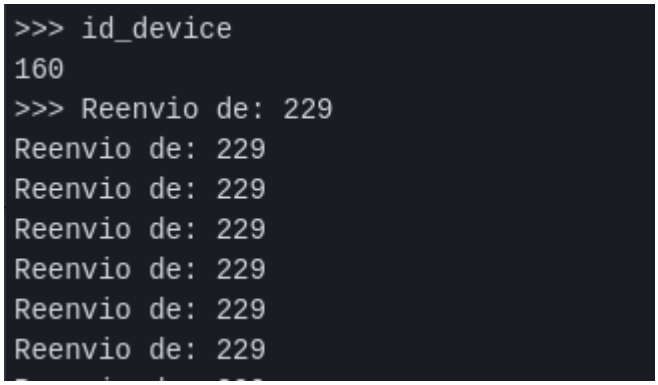

Ilustración 18 Muestra de respuesta de reenvío de dispositivo 160

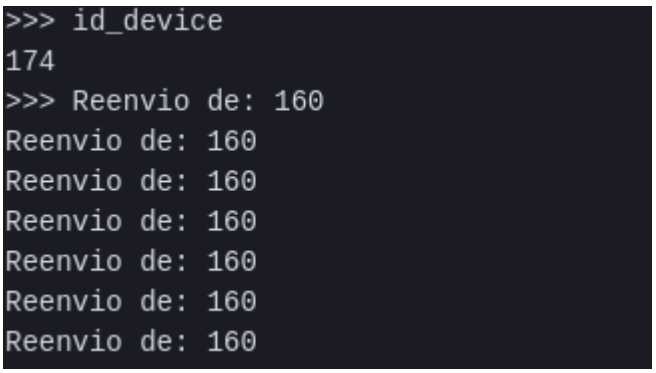

Ilustración 19 Muestra de respuesta de reenvío de dispositivo 174

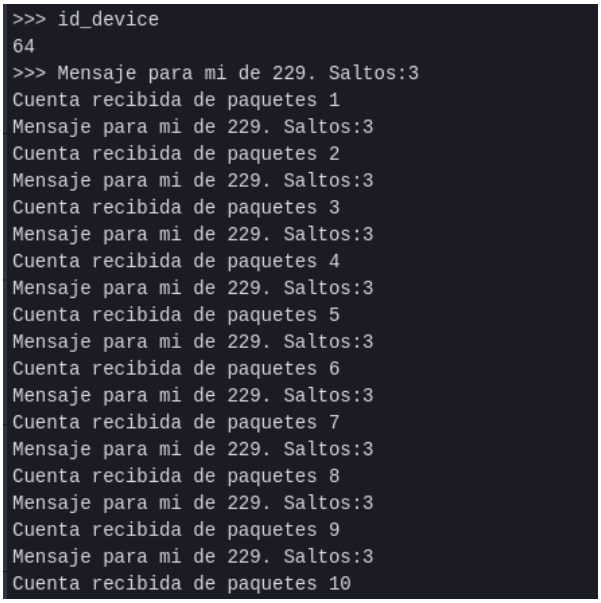

Ilustración 20 Muestra de recepción de mensajes por el dispositivo 64

Para la misma prueba se realizaron mediciones de consumo de energía en uno de los nodos. En este caso se eligió el nodo 160, el cual era un nodo intermedio y era el ideal para realizar las pruebas, ya que al ser un nodo intermedio debía realizar recepción y transmisión para llevar el mensaje a su destino. Al realizar la medición de consumo de energía se pudo observar el consumo que genera el dispositivo en su estado activo de recepción en LoRa. Al realizar las mediciones el sistema en el estado activo de recepción de mensaje consume aproximadamente 65,6 mA. En este estado, como el dispositivo siempre se mantendrá activo, el valor promedio en el tiempo se encontrará cerca de este valor. Para este caso de prueba el sistema presentaba de valor promedio 72,6 mA de consumo. En la siguiente gráfica se puede observar los resultados obtenidos:

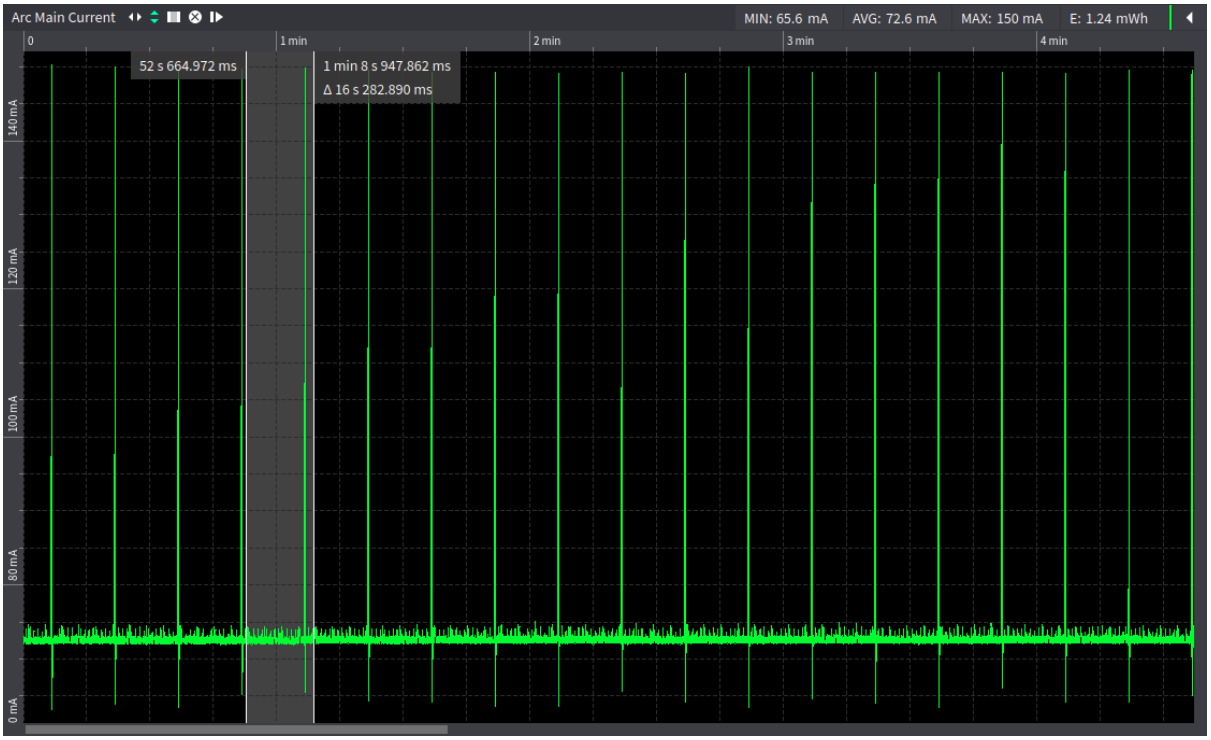

Ilustración 21 Grafica de medición de consumo de dispositivo 160 al realizar recepción y reenvío

Al realizar las mediciones se puede observar también el consumo que genera una transmisión. En este caso. cada transmisión se mostrará como un pico en el transcurso de la prueba. Se evidencia que existe en el tiempo una periodicidad en la transmisión, el cual al medir da un aproximado de 15 segundos en cada envío. Esto concuerda con lo configurado en el código donde se configuró de tal manera, que el sistema enviará un mensaje cada 15 segundos. Al realizar la medición del consumo de energía que genera la transmisión está arrojó un consumo de 150 mA aproximadamente en cada envío. Este consumo se genera en un corto periodo de tiempo, aproximadamente en 146 ms.

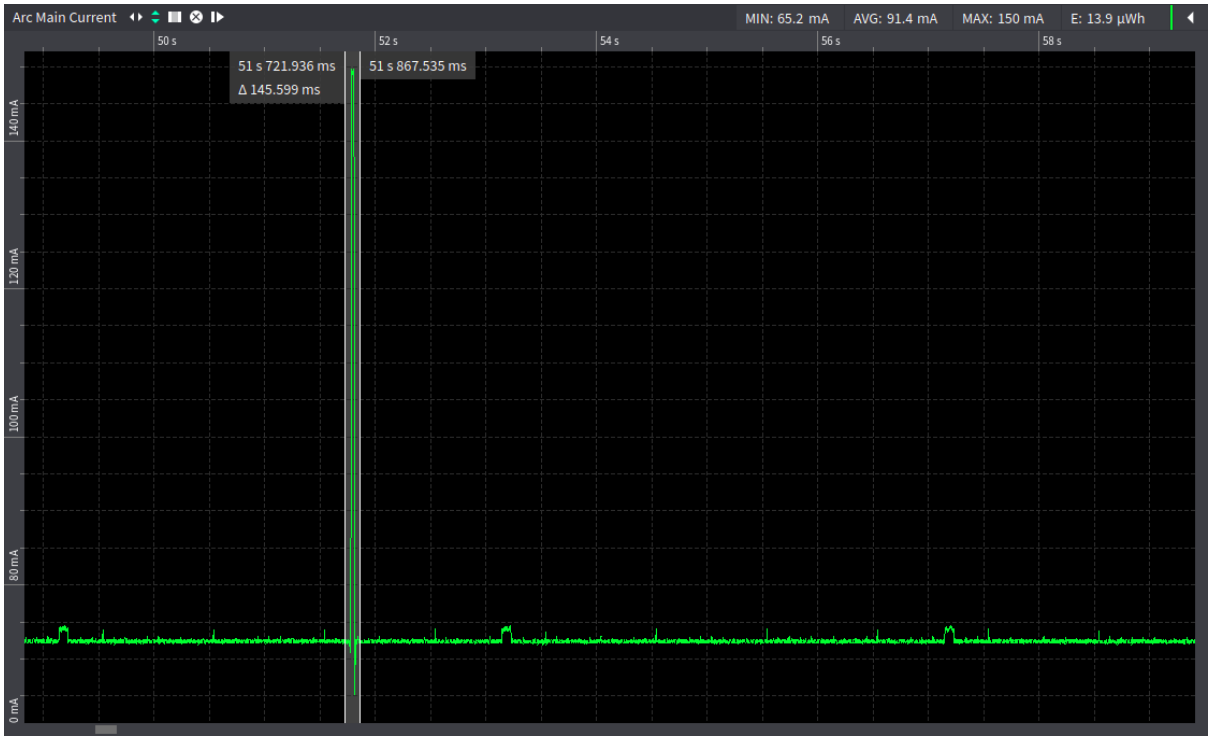

Ilustración 22 Grafica de medición de consumo de envió de un dispositivo

En la segunda fase de la prueba en esta topología se realizó en el modo de comprobación de mensaje. En éste modo se demostraría el éxito en su operatividad y el consumo de energía que genera este al realizar respuestas de mensajes.Para esto se realizo una modificación en la topología y se bloqueo la comunicación directa desde el nodo 64 al 229 como se muestra en la siguiente imagen.

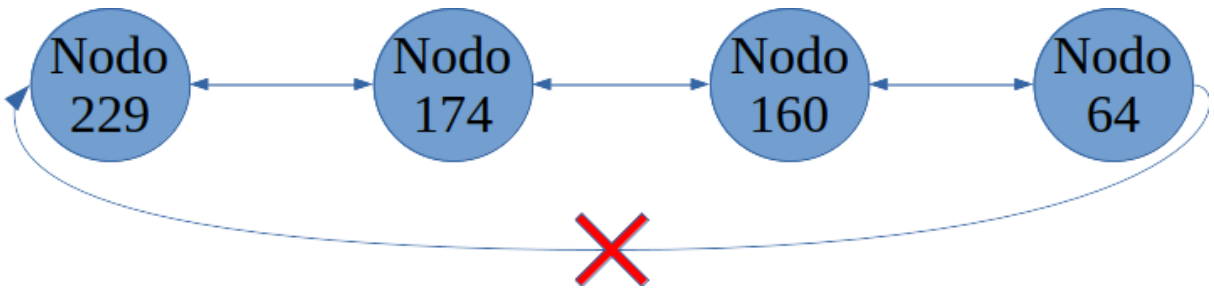

Ilustración 23 Modificación de modo de conexión de nodos en topología en anillo

En este sistema se puede llegar a observar un mayor consumo de energía ya que para efectos prácticos se configuró los dispositivos para activar el de manera de comprobar el éxito de la entrega. Si el sistema lograba comprobar el éxito de una

entrega, el sistema mantenía un led en verde. En caso contrario de no recibir respuesta el sistema mantiene activa un led en rojo.

Para este caso de prueba se observará más el dispositivo que realiza el envío de mensaje, el cual, cada cierto tiempo recibirá una respuesta del envío de sus mensajes. En la siguiente imágenes que se podrán observar se muestra el envío de paquetes que se realiza y cada cinco paquetes enviado por el nodo 229, éste recibe una respuesta del nodo 64. De igual manera el nodo 64, recibe sus mensajes del nodo 229 y no se evidencia alguna pérdida de paquete en la comunicación.

```
>>> Send(64)
>>> Numero de paquetes enviados: 1
Numero de paquetes enviados: 2
Numero de paquetes enviados: 3
Numero de paquetes enviados: 4
Numero de paquetes enviados: 5
Recibo un ACK
Numero de paquetes enviados: 6
Numero de paquetes enviados: 7
Numero de paquetes enviados: 8
Numero de paquetes enviados: 9
Numero de paquetes enviados: 10
Recibo un ACK
Numero de paquetes enviados: 11
Numero de paquetes enviados: 12
Numero de paquetes enviados: 13
Numero de paquetes enviados: 14
Numero de paquetes enviados: 15
Recibo un ACK
```

Ilustración 24 Envío y recepción de respuesta del dispositivo 229

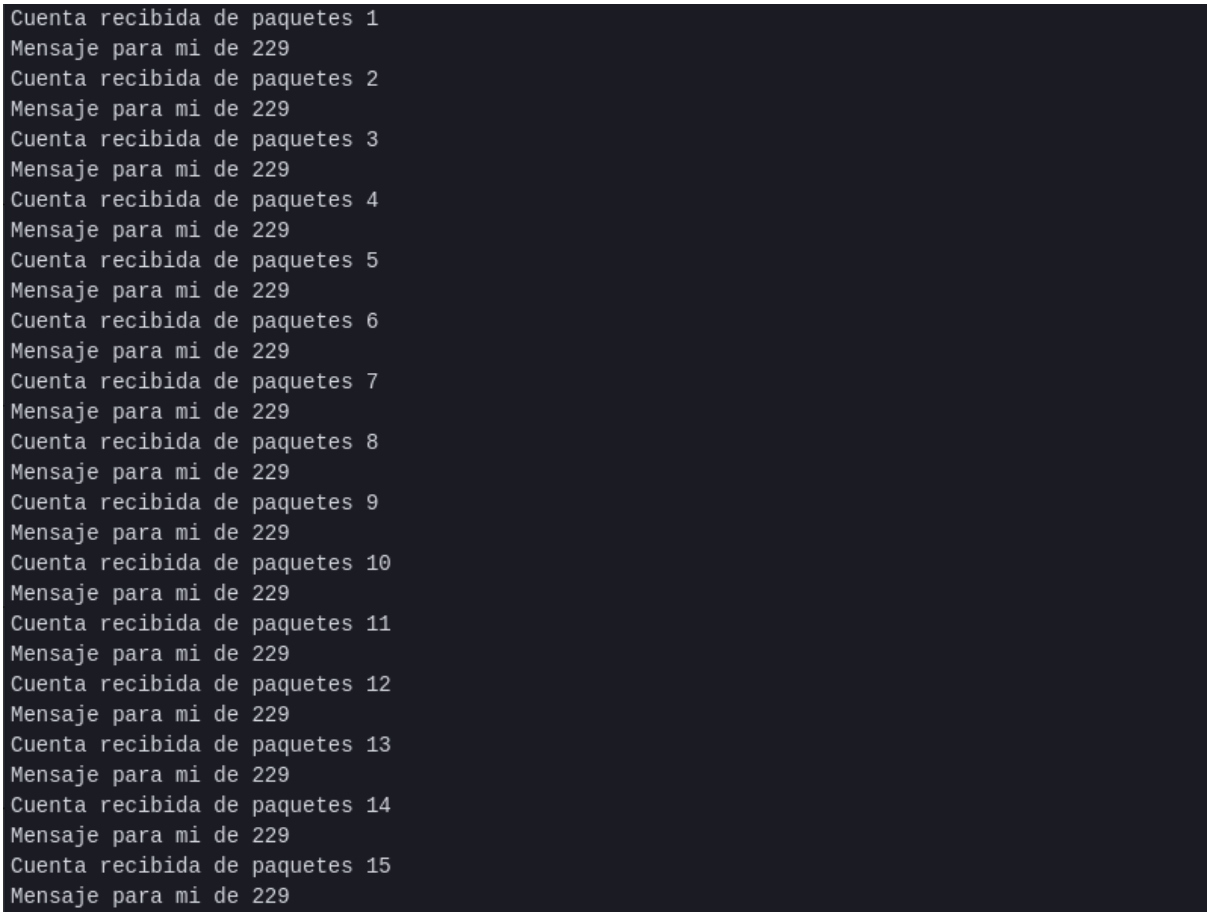

Ilustración 25 Recepción de mensaje en el dispositivo 64

En la medición de consumo de potencia se esperaba observar algún cambio al realizar un envío de respuesta con un consumo mayor de potencia en el tiempo de transmisión o con un pico a destiempo de la transmisión. Sin embargo, al realizar las mediciones, a simple vista no se logra notar alguna diferencia al realizar la respuesta de mensaje. Esto se debe a que el sistema realiza la recepción y respuesta de manera rápida mientras que la señal realiza la transmisión apenas recibe el mensaje. Al ampliar la gráfica en el tiempo en los momentos de respuesta, se puede observar los dos picos de respuesta donde la separación entre esta se encuentra en unos 55,5 ms. Se puede observar también que el consumo de ambos picos es igual para envío y respuesta donde para este dispositivo consume unos 134 mA en la transmisión. En la recepción se tiene un consumo promedio de 77 mA y es mayor al caso anterior debido al consumo que generan los leds. En la siguiente gráfica se puede apreciar lo descrito:

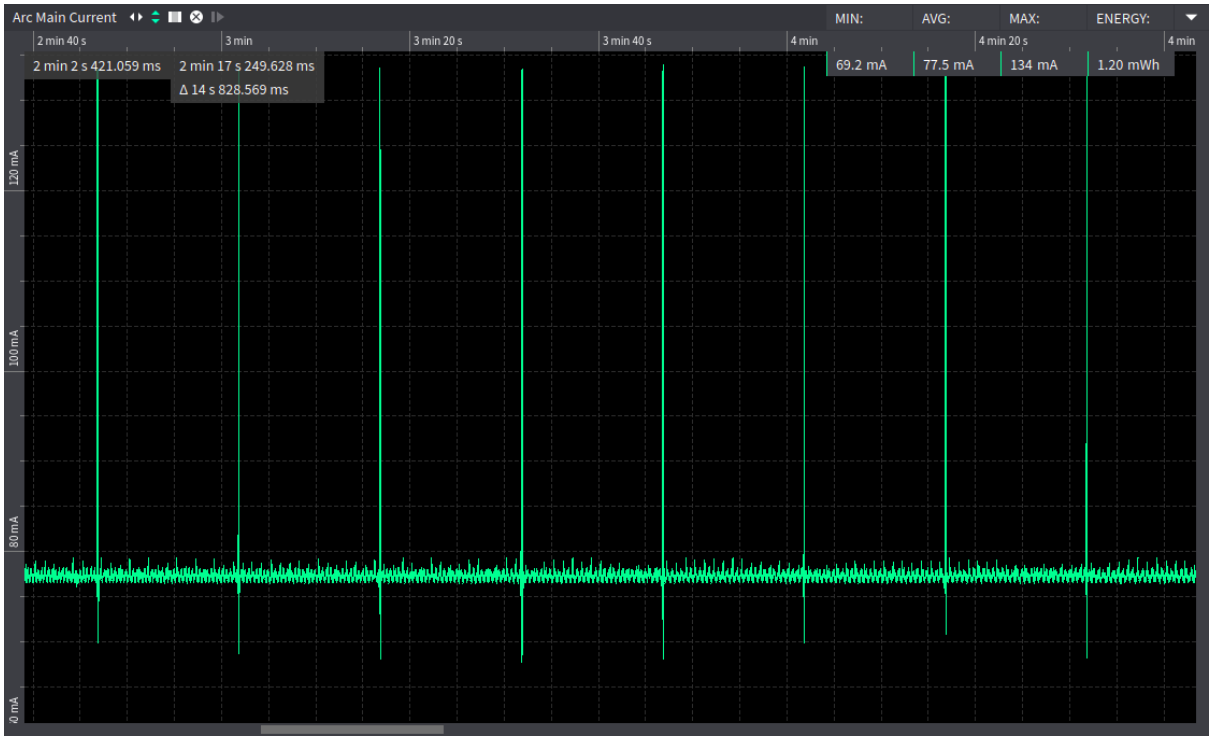

Ilustración 26 Medición de consumo de energía de emisión y reenvío del dispositivo 160

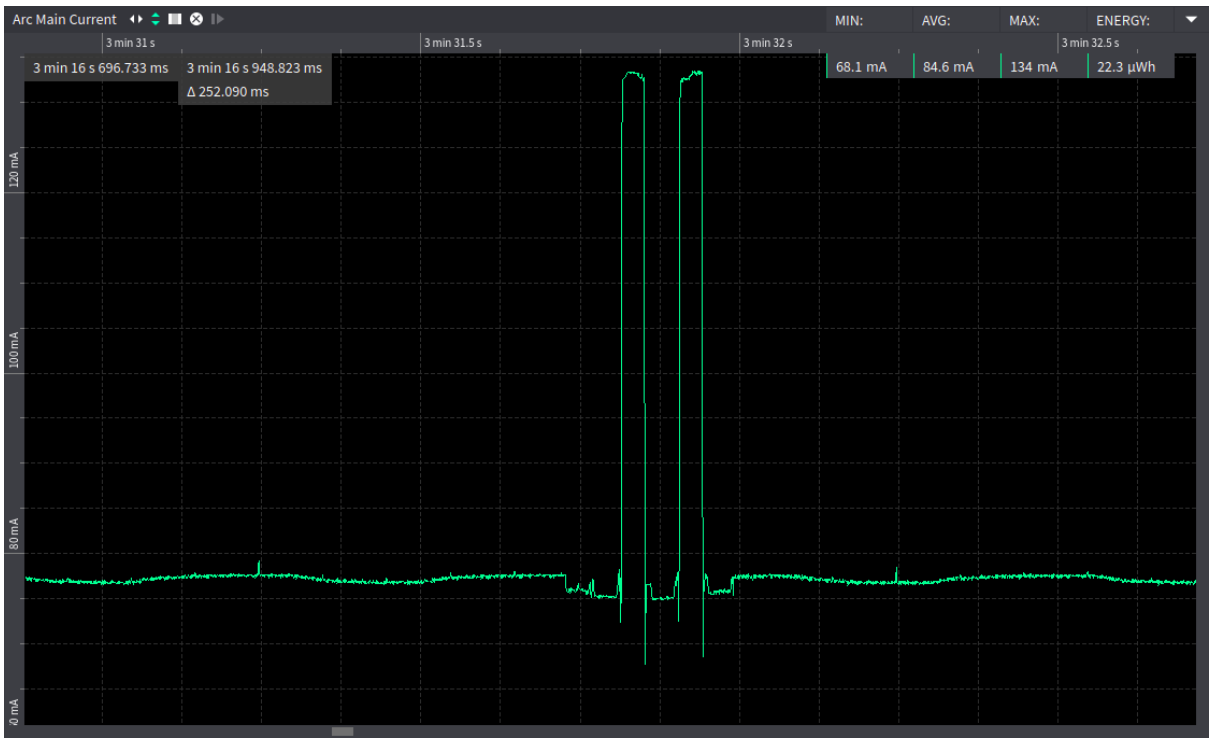

Ilustración 27 Medición de energía del reenvío de mensajes y respuesta de mensajes en el dispositivo 64

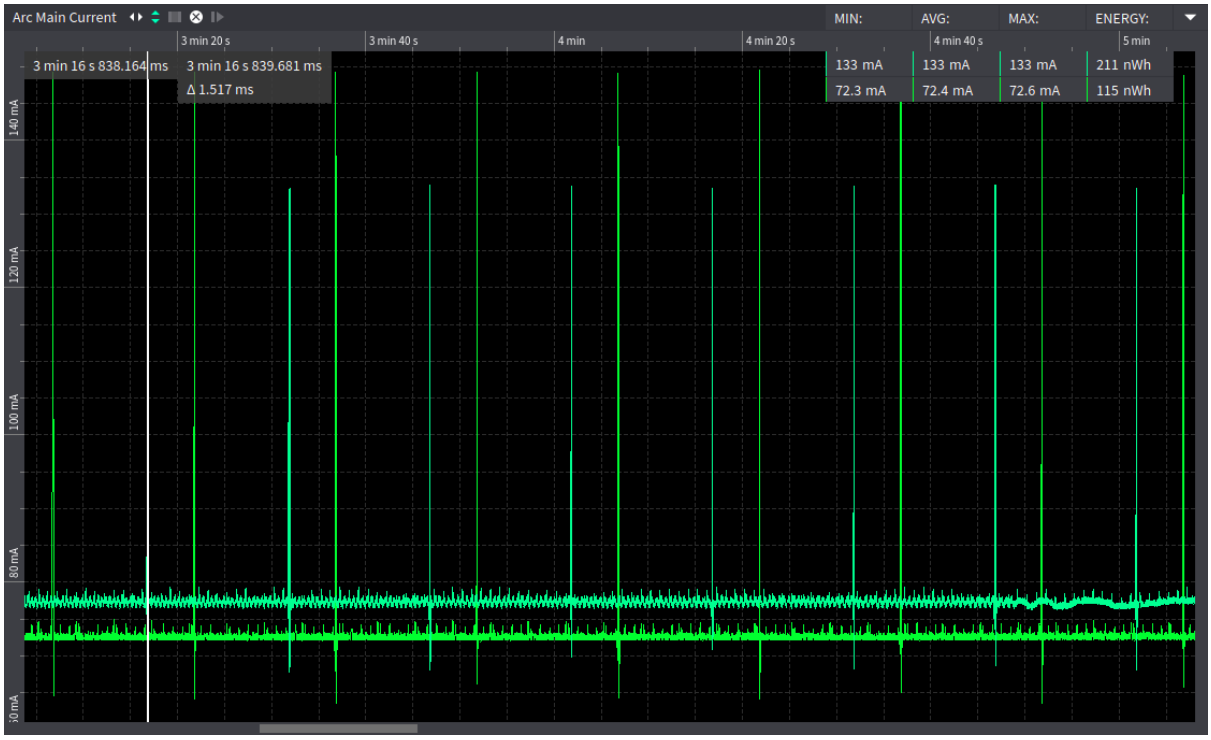

Ilustración 28 Comparación de consumo de energía entre modo normal y modo de comprobación de mensajes

Para el siguiente caso se realizó las pruebas con el sistema de suspensión. En este caso la idea es lograr que el sistema reduzca el consumo de energía en cada envío. Ya que en su mayor tiempo el sistema se encontrará en suspensión se deseaba lograr conseguir disminuir el consumo en el tiempo de inactividad. Del mismo modo que en las prácticas anteriores se dispondrá la comunicación entre el nodo 229 al 64 sin comprobación de mensaje. Para este caso al realizar un envío el sistema pasará a un estado de suspensión por 20 segundo y al pasar al estado de activación el sistema se mantendrá activo por 8 segundo hasta realizar los envíos

En este sistema se pueden presentar más errores en su utilización debido a que el sistema va a depender del tiempo de respuesta de los dispositivos. En una que otra prueba los dispositivos presentaban falla al activarse del estado de suspensión lo que causaba un error en el proceso. Por ende, se trató de dar un margen de 8 segundos para que el sistema presente el suficiente tiempo de recuperación y logre enviar los mensajes con éxito.

Al observar la respuesta de los dispositivos inicial y final, se puede observar como el sistema envía un mensaje y pasa al estado de hibernación, se levanta y vuelve a repetir el ciclo. Para el caso de recepción elabora el mismo ciclo con la diferencia de que recibe los mensajes.

```
>>> Send(64)
>>> Numero de paquetes enviados: 1
Sleep
Wake up
Numero de paquetes enviados: 2
Sleep
Wake up
Numero de paquetes enviados: 3
Sleep
Wake up
Numero de paquetes enviados: 4
Sleep
Wake up
Numero de paquetes enviados: 5
Sleep
```

Ilustración 29 Emisión de mensajes y cambio de estado de suspensión del dispositivo 229

```
>>> Mensaje para mi de 229
Cuenta recibida de paquetes 1
Sleep
Wake up
Mensaje para mi de 229
Cuenta recibida de paquetes 2
Sleep
Wake up
Mensaje para mi de 229
Cuenta recibida de paquetes 3
Sleep
Wake up
Mensaje para mi de 229
Cuenta recibida de paquetes 4
Sleep
Wake up
Mensaje para mi de 229
Cuenta recibida de paquetes 5
Sleep
Wake up
Mensaje para mi de 229
Cuenta recibida de paquetes 6
Sleep
```

Ilustración 30 Recepción de mensajes y cambio de estado de suspensión del dispositivo 64

Al analizar la gráfica de medición de potencia se puede apreciar mejor el estado de suspensión del sistema. Al suspender el sistema, el consumo de energía llega a valores por debajo de los 20 mA. El sistema realiza un periodo de 20 segundos de suspensión y 8 segundos en modo activo logrando así que el envío entre cada mensaje es de 28 segundos. Cuando el dispositivo genera una transmisión el sistema consume de energía 136 mA, en estado de recepción de mensaje el sistema consume 57.1 mA en promedio y en estado de suspensión logra obtener un consumo de 12,4 mA

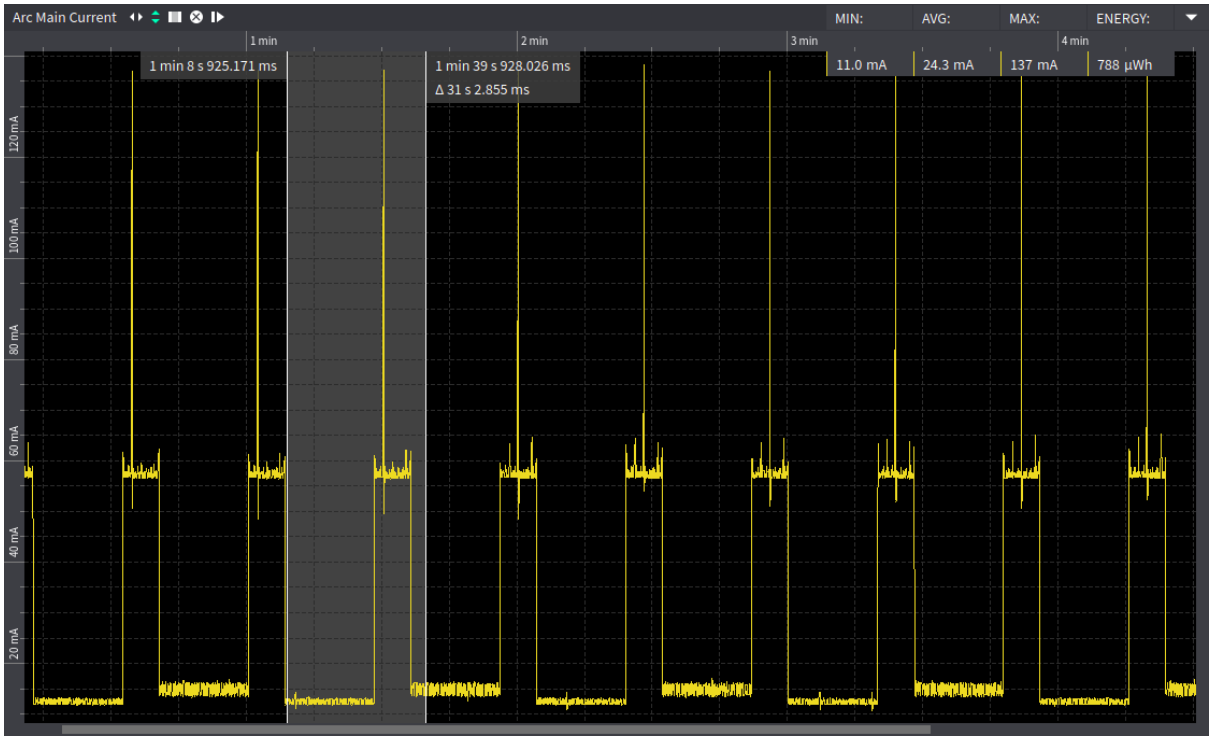

Ilustración 31 Medición de consumo de energía en modo de suspensión

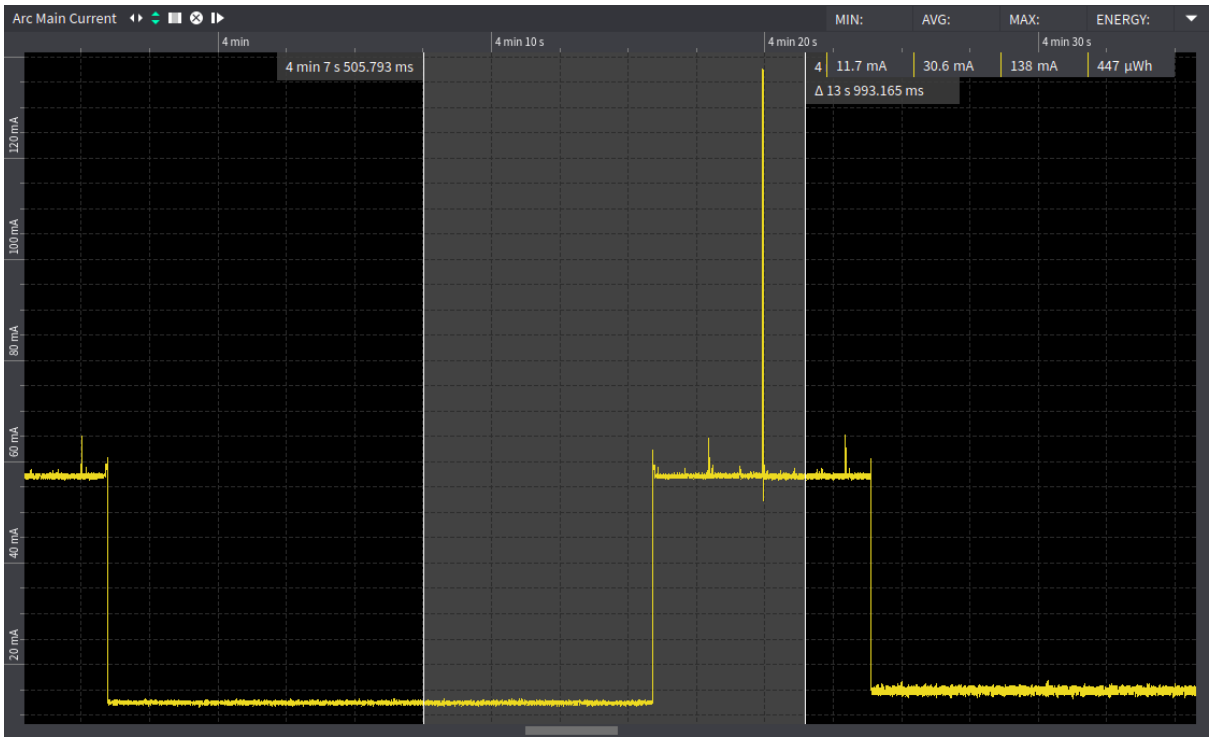

Ilustración 32 Medición de máximo y mínimo en modo de suspensión

Realizando una comparación entre los tres modos propuestos anteriormente se puede plantear que el sistema de suspensión logra reducir bastante el consumo de energía. En promedio el sistema de suspensión logra conseguir un consumo de 1,24 mWh, en comparación al sistema normal y con comprobación que logran un consumo de 3,31 mWh y 3,53 mWh respectivamente. En la siguiente tabla se muestra los resultados obtenidos:

	MIN	AVG	MAX	Energia
Estado Simple	63.4 mA	77.5 mA	138 mA	3.31 mWh
Estado Comprobación	68.8 mA	72.7 mA	150 mA	3.53 mWh
Estado de Hibernado	11.7 mA	29.5 mA	134 mA	1.34 mWh

Tabla 3 Comparación entre modo simple, modo de comparación y modo de hibernación

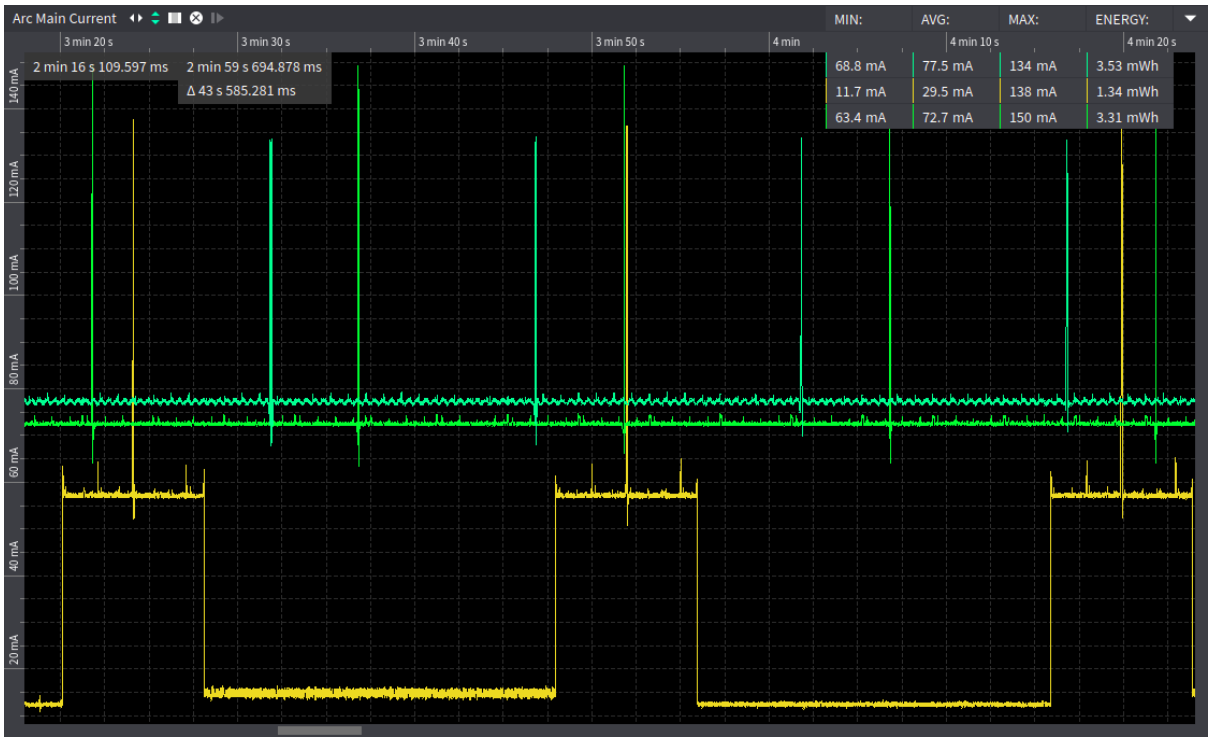

Ilustración 33 Grafica comparativa de consumo de energía entre los tres modos.

Pruebas topología punto a punto

Para esta prueba se realizará conexión punto a punto entre dos parejas. Entre ellos se realizará una comunicación y se intentará enviar la mayor cantidad de mensajes entre dispositivos al mismo tiempo. Con esto se trata de medir la capacidad de tolerancia a fallos cuando el canal se encuentra saturado de mensajes. Se trata de saturar el medio de transmisión y el buffer del sistema al recibir mensajes del mismo equipo. En la siguiente imagenes se muestra el envío y recepción de los mensajes por parte de las parejas. Ambos sistemas se encuentran enviando simultáneamente y a su vez cada dispositivo envía 5 mensajes a su pareja.

```
>>> Numero de paquetes enviados: 1
Send(64)
>>> Numero de paquetes enviados: 1
Send(64)
>>> Numero de paquetes enviados: 1
Send(64)
>>> Numero de paquetes enviados: 1
Send(64)
>>> Numero de paquetes enviados: 1
Numero de paquetes enviados: 2
Numero de paquetes enviados: 2
Numero de paquetes enviados: 2
Numero de paquetes enviados: 2
Numero de paquetes enviados: 2
Numero de paquetes enviados: 3
Numero de paquetes enviados: 3
Numero de paquetes enviados: 3
Numero de paquetes enviados: 3
Numero de paquetes enviados: 3
```

```
>>> Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 1
Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 2
Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 3
Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 4
Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 5
Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 6
Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 7
Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 8
Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 9
Mensaje para mi de 160. Saltos:1
Cuenta recibida de paquetes 10
```

Ilustración 34 Resultados de envío y recepción de mensajes por los nodos 64 y 160

```
>>> Numero de paquetes enviados: 1
Send(174)
>>> Numero de paquetes enviados: 1
Send(174)
>>> Numero de paquetes enviados: 1
Send(174)
>>> Numero de paquetes enviados: 1
Send(174)
>>> Numero de paquetes enviados: 1
Numero de paquetes enviados: 2
Numero de paquetes enviados: 2
Numero de paquetes enviados: 2
Numero de paquetes enviados: 2
Numero de paquetes enviados: 2
Numero de paquetes enviados: 3
Numero de paquetes enviados: 3
Numero de paquetes enviados: 3
Numero de paquetes enviados: 3
Numero de paquetes enviados: 3
```

```
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 2
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 3
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 4
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 5
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 6
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 7
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 8
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 9
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 10
Mensaje para mi de 229. Saltos:1
```

Ilustración 35 Resultados de envío y recepción de mensajes por los nodos 229 y 174

Al transcurrir el tiempo de la prueba. Se evidencia poca pérdida de paquetes en la transmisión. La mayoría de los paquetes llegan a su destino, exceptuando en algunos casos que el sistema pierde algunos paquetes. Sin embargo, se obtiene con éxito enviar diferentes mensajes desde el mismo dispositivo.

Prueba de topología malla

Para la topología malla es necesario que todos los dispositivos se comuniquen entre sí. Para ello no es necesario colocarlo en modo Test. Al desactivar el modo Test todos los dispositivos tienen libertad de comunicarse sin limitación. Para esta prueba el sistema podrá enviar mensajes a todos los nodos. Se plantea verificar que el sistema tome la mejor ruta la cual sería la de un salto y los demás mensajes deben ser descartados por el dispositivo final. Esto evitaría que existieran bucles en el sistema.

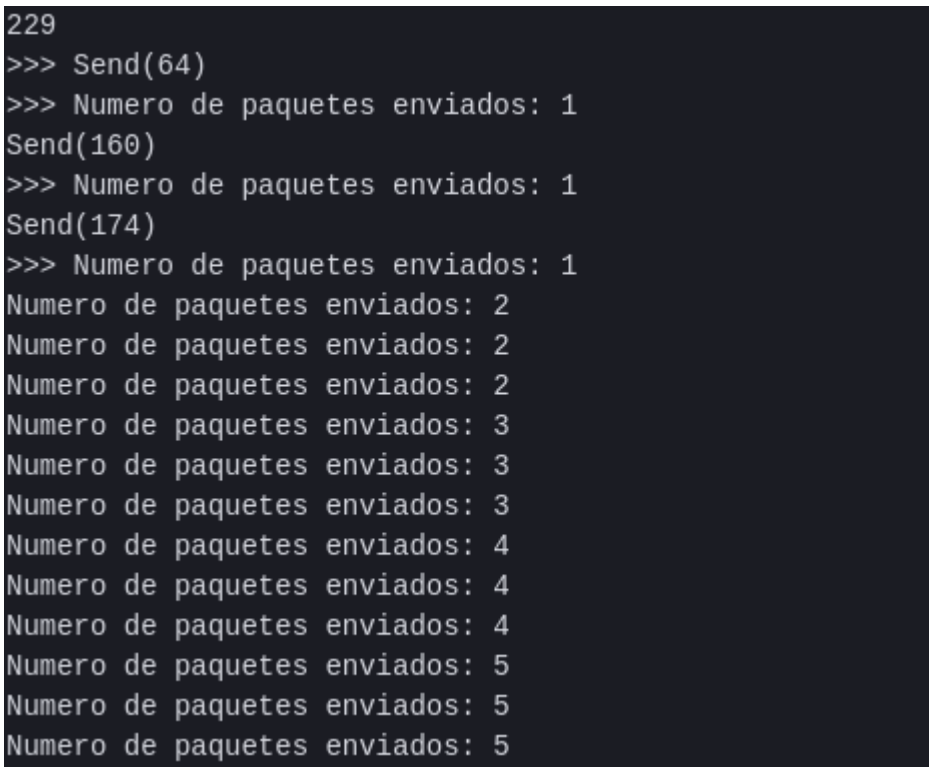

Ilustración 36 Envío de mensajes a los nodos 64, 160 y 174 por parte del nodo 229

En la primera imagen se puede observar el dispositivo con ID 229 realizando un envío de mensaje a los otros nodos. En este caso, aunque otro nodo realice reenvío de mensaje, éste descarta los mensajes ya que el ID de cada mensaje coincide con los enviados por él mismo.

```
64
>>> Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 1
Reenvio de: 229
Reenvio de: 229
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 2
Reenvio de: 229
Reenvio de: 229
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 3
Reenvio de: 229
Reenvio de: 229
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 4
```

Ilustración 37 Resultado de recepción de mensajes por el nodo 64

```
>>> Reenvio de: 229
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 1
Reenvio de: 229
Reenvio de: 229
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 2
Reenvio de: 229
Reenvio de: 229
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 3
Reenvio de: 229
Reenvio de: 229
Mensaje para mi de 229. Saltos:1
Cuenta recibida de paquetes 4
```

Ilustración 38Resultado de recepción de mensajes por el nodo 160

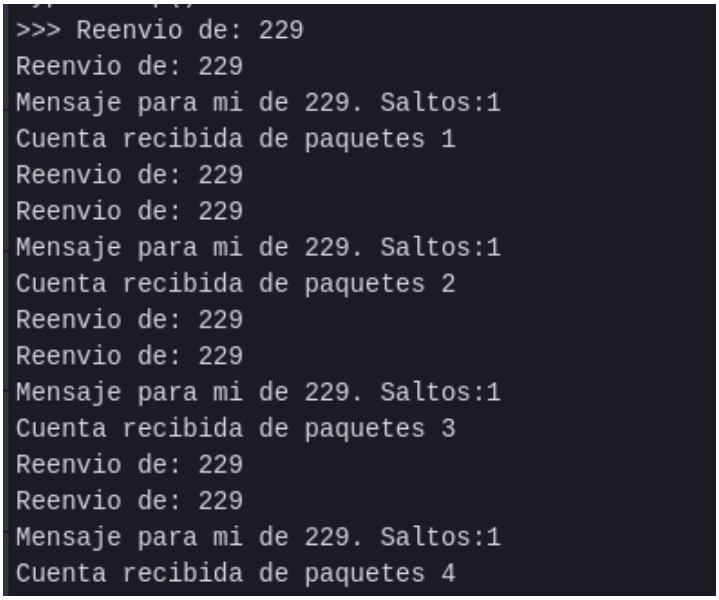

Ilustración 39 Resultado de recepción de mensajes por el nodo 174

En las imágenes anteriores se muestra a los dispositivos recibiendo cada mensaje del nodo principal. En cada dispositivo se puede observar que recibe un mensaje de un mismo nodo y que la distancia hasta este es de un salto. Con esto se cumple el hecho de que el mensaje está llegando de forma directa desde el nodo emisor. Se puede observar también, que cada nodo realiza un reenvío de mensajes. Esto se da, ya que, al recibir un mensaje para un destino diferente a él, este automáticamente lo envía. Sin embargo, los dispositivos descartan este reenvío ya que el mensaje lo recibe de forma directa, consiguiendo el camino óptimo.

6. Conclusiones

Realizando una evaluación del diseño del protocolo con los resultados obtenidos en la prueba, se puede llegar a una conclusión bastante favorable en el trabajo. Con el diseño del programa expuesto se logran los objetivos planteados al inicio del trabajo. Con la tecnología LoRa y la utilización de los dispositivos Pycom, se puedo realizar las funciones que permitían una comunicación en una red de topología Mesh. Con el código diseñado se pudo realizar comunicación entre varios dispositivos y se realizaron pruebas para comprobar que tiene una gran capacidad de soportar el tráfico de numerosos mensajes. Como código para la utilización de IoT en bajo consumo se realizaron mediciones donde se evidencia que el sistema posee un bajo consumo energético. De igual manera, se diseñó el sistema de hibernación, donde el dispositivo realiza la espera de emisión y recepción de nuevos mensajes entrando en un estado de suspensión.

En la comprobación de mensajes se planteo de tal manera que la respuesta que se realice a un dispositivo se envíe cada cierta cantidad de mensajes. Esto pensando en disminuir el consumo que puede generar responder cada mensaje y el aumento del uso de recursos de la red al tener mas mensajes en tráfico. De igual manera el código esta diseñado para poder realizar la respuesta a la cantidad de mensajes que el usuario desee.

La estructura posee algunos errores que se debe tomar en consideración. Los dispositivos Lopy por su diseño y su baja capacidad de sistema puede llegar a presentar fallas en su middleware, por eso al realizar las pruebas se llegaron a presentar fallas de recepción errónea de un paquete aislado o en la congelación del dispositivo. En el modo de hibernación los diferentes dispositivos no realizan el mismo tiempo de respuesta de reinicio. Por eso se debió considerar un tiempo prudencial para que todos los dispositivos estuvieran activos nuevamente y no superar el número de envío en este estado entre dispositivos para evitar estos errores. Es recomendable para futuros proyectos estudiar mejor la sincronización del tiempo entre los dispositivos y probar la utilización de diferentes dispositivos que tengan mejores recursos de proceso y memoria.

Uno de los puntos que se puede plantear para trabajos futuros es el tema de la seguridad entre dispositivos. Para este proyecto se planteó un código que pudo ser útil para uso de seguridad, sin embargo, se utilizó de manera superficial para realización de

pruebas. Partiendo de este código se puede indagar mas y llegar a buenos resultados mejorando la seguridad del sistema.

Aunque todas las pruebas se realizaron con simulaciones, debido a no contar con los recursos humanos para ser realizado a grandes distancias. Con el modo de prueba implementado fue bastante efectivo y se logró probar el código para evaluar si se cumplía los objetivos.

Con la finalización de este trabajo se cumple grandes objetivos que fueron estudiados a lo largo de estos estudios. En el trabajo se pudo implementar diseño de arquitecturas de red y manejo de dispositivos de procesamiento y sensorización. Del trabajo se consiguió conocimientos de las nuevas tecnologías de comunicación, como lo es LoRa y la programación de los dispositivos que forman parte de esta tecnología.

7. Referencias

1. **Rodríguez Munca, José Daniel.** *Dispositivo LoRa de Comunicación a Largo Alcance Y Bajo.* Madrid - España : s.n., 2016.
2. **Muñoz, Alda Marin.** *Despliegue De Una Red De Sensores Basada En Chips Esp-8266.* [Universidad Complutense de Madrid] Madrid : s.n., 2020.
3. **Moya, Michael André.** *Evaluación de pasarela LoRa/LoRaWAN.* Valencia : Universidad Politecnica de Valencia , 2018.
4. *Analysis of LoRa and Bluetooth Low.* **Pérez Campos, Rafael.** Cartagena : Universidad Politécnica De Cartagena, 2019.
5. **Pickering, Paul.** *Desarrollar con LoRa para aplicaciones IoT de baja tasa y largo alcance.* s.l. : Colaboración de Editores de Digi-Key de América del Norte, 2017.
6. **Chinchilla, Franco Mateo y Yepes, Jorge Andrés.** *Radio enlace mesh con tecnología LoRa para el monitoreo remoto de.* [Unidades Tecnologicas Santander] 2020.
7. **Esnoz, Ignacio.** *Internet de las Cosas de largo alcance (LoRa).* [Teldat Blog] 2017.
8. *Epidemic Routing for Partially-Connected Ad Hoc Networks.* **Vahdat, Amin y Becker, David.** Durham : Duke University.
9. **Alrfaay, Mohamad y Lenando, Halikul.** EpSoc: Social-Based Epidemic-Based Routing Protocol in. [En línea] 2018. https://downloads.hindawi.com/journals/misy/2018/6462826.pdf.

Anexos

Anexo 1: Estructura de código principal

```
import os
import socket
import time
import struct
import pycom
import ubinascii, network
from network import LoRa
import machine

from LoraPack import *
from env_recv import *
import _thread

#lock = _thread.allocate_lock()

# A basic package header, B: 1 byte for the deviceId, B: 1 byte for the pkg size, %ds: Formatted string for string
_LORA_PKG_FORMAT = "!BBIBB"


# A basic ack package, B: 1 byte for the deviceId, B: 1 byte for the pkg size, B: 1 byte for the Ok (200) or error messages
_LORA_PKG_ACK_FORMAT = "!BBIB"

MAX_RECV = 1
MAX_BUFFER = 4

#Activador de comprobacion de mensaje
ACK_ACTV = 0

#Activador de modo Test
modeTEST = 1

#Activador de suspesion del sistema
ModeSleep = 0

# Open a LoRa Socket, use rx_iq to avoid listening to our own messages
# Please pick the region that matches where you are using the device:
# Asia = LoRa.AS923
# Australia = LoRa.AU915
# Europe = LoRa.EU868
# United States = LoRa.US915

lora = LoRa(mode=LoRa.LORA, region=LoRa.EU868)
```

```
    lora_sock = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
    lora_sock.setblocking(False)
    location = 0x01
    id_device = Def_ID(lora)
    count = 100
    send_OK = 0
    cuenta_recibo = 0
    list_recv = []           # LISTA DE LOS ID DE LOS MENSAJES
RECIBIDOS


    active_sleep = 0

    buffer = []             # LISTA DE LOS MENSAJES EN BUFFER A
ENVIAR

    List_send = ID_recv()    #LISTA DE CONTEO PARA ACK

    if ACK_ACTV:

        pycom.heartbeat(False)


    #lista ID para enviar y bloquear
    if(modeTEST):
        namefile = "ID" + str(id_device)
        ID_block = Open_Test(namefile)


    def Send_buffer():
        global lora, lora_sock, location, id_device, list_recv,
active_sleep

        while(True):
            if len(buffer)>0:
                #if id_send not in list:
            #ENVIO
                send_pck = buffer.pop(0)
                id_to_send,       id_end,       UID_msg       =
get_IDdest(send_pck)
                if(id_to_send == id_device):
                    lora_sock.send(send_pck)
                else:
                    print("Reenvio de: %d" % id_to_send)
                    pck_reenvio    =    set_IDorigen(send_pck,
id_device)
                    lora_sock.send(pck_reenvio)
            elif active_sleep and ModeSleep:
                print("Sleep")
                machine.sleep(1000*20,False)
                lora = LoRa(mode=LoRa.LORA, region=LoRa.EU868)
                print("Wake up")
                active_sleep = 0
```

```
            if ModeSleep:
                time.sleep(2)


    def Packet_buffer(ID_send):
        global   location,   id_device,   list_recv,   buffer,
active_sleep
        cuenta_envio = 0
        while(True):
            if(ID_send):
            #ENVIO AL BUFFER
                if ModeSleep:
                    if not active_sleep:
                        time.sleep(4)
                        UID = UID_message(id_device)
                        list_recv.append(UID)
                        pck = pack_Lora(id_device, ID_send, UID,
0, location)
                        if(len(buffer) < MAX_BUFFER):
                            cuenta_envio = cuenta_envio + 1
                            print("Numero de paquetes enviados:
%d" % cuenta_envio)
                            buffer.append(pck)
                        active_sleep = 1
                else:
                    UID = UID_message(id_device)
                    list_recv.append(UID)
                    pck  =  pack_Lora(id_device,  ID_send,  UID,
location, 0)
                    if(len(buffer) < MAX_BUFFER):
                        cuenta_envio = cuenta_envio + 1
                        print("Numero de paquetes enviados: %d"
% cuenta_envio)
                        buffer.append(pck)
                    time.sleep(15)


    def Recv():
        global lora, lora_sock, location, id_device, list_recv,
modeTEST, buffer, count, active_sleep, cuenta_recibo

        #Recibo
        recv_pkg = lora_sock.recv(256)
        recv_pkg_len = recv_pkg[1]
        id_to_send, id_end, UID_msg = get_IDdest(recv_pkg)

        if(modeTEST):
            if id_to_send in ID_block:
                return
        if UID_msg in list_recv:
            return
        else:
            list_recv.append(UID_msg)
```

```
        if (len(recv_pkg) > 7):
            if(id_end == id_device):
                id_to_send, id_end, UID_msg, TTL, location =
unpack_Lora(recv_pkg,  recv_pkg_len)
                cuenta_recibo = cuenta_recibo + 1
                print("Mensaje para mi de %d. Saltos:%d" %
(get_SendID(UID_msg), TTL))
                TTL = TTL + 1
                print("Cuenta recibida de paquetes %d" %
cuenta_recibo)
                if List_ack(get_SendID(UID_msg), List_send) and
ACK_ACTV:
                    send_ack(get_SendID(UID_msg))
                if ModeSleep:
                    time.sleep(5)
                    if not len(buffer):
                        active_sleep = 1
            else:
                if(len(buffer) < MAX_BUFFER):
                    buffer.append(recv_pkg)
                    if ModeSleep:
                        time.sleep(5)
                        active_sleep = 1

        elif (len(recv_pkg) > 2 and len(recv_pkg) < 8 and
ACK_ACTV):
            if(id_end == id_device):
                id_to_send,    id_end,    UID_msg,    TTL    =
unpack_AckLora(recv_pkg,  recv_pkg_len)
                TTL = TTL + 1
                print("Recibo un ACK. Saltos %d" % TTL)
                send_OK = 1
                pycom.rgbled(0x007f00)
                count = 100
            else:
                if(len(buffer) < MAX_BUFFER):
                    buffer.append(recv_pkg)


    def send_ack(id_respond):
        global lora, lora_sock, list_recv, buffer
        UID = UID_message(id_device)
        list_recv.append(UID)
        pkg_ack = pack_AckLora( id_device, id_respond, UID, 0)
        buffer.append(pkg_ack)


    def clear():
        global list_recv
        time.sleep(200)
        list_recv = []
```

```
def ack_off():
    global  count , send_OK
    while(ACK_ACTV):
        time.sleep(1)
        count = count - 1
        if not count:
            pycom.rgbled(0x7f0000)
            send_OK = 0


def lora_cb(lora):
    events = lora.events()
    if events & LoRa.RX_PACKET_EVENT:
        Recv()

lora.callback(trigger=(LoRa.RX_PACKET_EVENT                |
LoRa.TX_PACKET_EVENT), handler=lora_cb)

def Send(ID_send):
    _thread.start_new_thread(Packet_buffer, (ID_send, ))

time.sleep(10)

_thread.start_new_thread(Send_buffer, ())

_thread.start_new_thread(ack_off, ())

_thread.start_new_thread(clear,())
```

Anexo 2: Codigo de la libreria diseñada LoraPack

```
import os
import socket
import time
import struct
from network import LoRa
import uos
import ubinascii, network

_LORA_PKG_FORMAT = "!BBIBB"
_LORA_PKG_ACK_FORMAT = "!BBIB"


class ID_recv:

    def __init__(self):
        self.id = []
        self.count = []
    def new_id(self, id):
```

```
            self.id.append(id)
            self.count.append(0)
        def add_count(self, id):
            i = self.id.index(id)
            self.count[i] = self.count[i] + 1
        def remove_id(self, id):
            i = self.id.index(id)
            self.id.pop(i)
            self.count.pop(i)


    def    pack_Lora(DEVICE_ID,Device_next,    UID_msg,    TTL,
location):
        global _LORA_PKG_FORMAT, _LORA_PKG_ACK_FORMAT
        pkg    =    struct.pack(_LORA_PKG_FORMAT,    DEVICE_ID,
Device_next, UID_msg, TTL, location)
        return pkg

    def pack_AckLora(DEVICE_ID, DEVICE_Resp, UID_msg, TTL):
        global _LORA_PKG_FORMAT, _LORA_PKG_ACK_FORMAT
        pkg_ack = struct.pack(_LORA_PKG_ACK_FORMAT, DEVICE_ID,
DEVICE_Resp, UID_msg, TTL)
        return pkg_ack


    def unpack_Lora(recv_pkg,  recv_pkg_len):
       global _LORA_PKG_FORMAT, _LORA_PKG_ACK_FORMAT
       #device_id,     location,     Px,     Device_next     =
struct.unpack(_LORA_PKG_FORMAT % recv_pkg_len, recv_pkg)
       device_id,  Device_next,  UID_msg,  TTL,  location  =
struct.unpack(_LORA_PKG_FORMAT, recv_pkg)
       return device_id, Device_next, UID_msg, TTL, location


    def unpack_AckLora(recv_ack,  recv_pkg_len):
       global _LORA_PKG_FORMAT, _LORA_PKG_ACK_FORMAT
       #device_id,         data_null,         ack_Px         =
struct.unpack(_LORA_PKG_ACK_FORMAT % recv_pkg_len, recv_ack)
       device_id,    device_respond,    UID_msg,    TTL    =
struct.unpack(_LORA_PKG_ACK_FORMAT, recv_ack)
       return device_id, device_respond, UID_msg, TTL

    def get_IDdest(recv_pkg):
        global _LORA_PKG_FORMAT, _LORA_PKG_ACK_FORMAT
        ID_send,  ID_end,  UID  =  struct.unpack_from("!BBI",
recv_pkg, 0)
        return ID_send, ID_end, UID


    def set_IDorigen(pack, ID_sent):
        global _LORA_PKG_FORMAT, _LORA_PKG_ACK_FORMAT
        if (len(pack) > 7 ):
```

```
            device_id, Device_next, UID_msg, TTL, location =
struct.unpack(_LORA_PKG_FORMAT, pack)
            TTL = TTL + 1
            pkg = pack_Lora(ID_sent, Device_next, UID_msg, TTL,
location)
            return pkg
        elif(len(pack) > 2 and len(pack) < 8):
            device_id,     Device_next,     UID_msg,     TTL=
struct.unpack(_LORA_PKG_ACK_FORMAT, pack)
            TTL = TTL + 1
            pkg  =  pack_AckLora(ID_sent,  Device_next,  UID_msg,
TTL)
            return pkg
        #struct.pack_into('B', pack, offset, v1, v2, ...)

    def Random():
        result = (uos.urandom(1)[0] / 256) * 1000
        return int(result)


    def Def_ID(lora):
        id = ubinascii.hexlify(lora.mac()).upper().decode('utf-
8')
        id = id[14:16]
        return int(id, 16)

    def UID_message(id_device):
        return (id_device * 1000) + Random()

    def get_SendID(UID):
        SendID = UID/1000
        return int(SendID)

    def List_ack(ID, list):
        if ID in list.id:
            i = list.id.index(ID)
            list.add_count(ID)
            if list.count[i] >= 5:
                list.remove_id(ID)
                return 1
            else:
                return 0
        else:
            list.new_id(ID)
            list.add_count(ID)
            return 0


    def Open_Test(file_test):
        f = open (file_test,'r')
        mensaje = f.readlines()
        mensaje = [int(x) for x in mensaje]
        f.close()
```

```
    return mensaje
```