UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Estudio de la capacidad de transmisión y propagación en protocolos IoT.

TESIS

Que para obtener el título de

Ingeniero en Telecomunicaciones

P R E S E N T A

Iván Enrique Vázquez Toledo

DIRECTOR DE TESIS

Dr. Víctor Rangel Licea

Ciudad Universitaria, Cd. Mx., 2022

Dedicatoria

A mi mamá y a mi abuelo: Claudia Angelica Toledo Núñez y Antonio Toledo Vázquez…
…por enseñarme a no rendirme y como trazar mi propio camino.

A mi abuela y a mi papá: Martha Núñez Preciado y Enrique A. Vázquez Velasco…
…por jamás flaquear en su apoyo incondicional.

A mis hermanos: Erick Enrique Vázquez Toledo y Claudia Samantha Vázquez Toledo…
…por siempre recordarme de dónde vengo y a donde quiero ir.

A mi familia en general…
…por ser pilar indiscutible en mi vida.

A la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería…
…por la oportunidad de superarme y formarme, por ser una segunda familia y casa.

Al Dr. Víctor Rangel Licea…
…por siempre guiar con amabilidad, brindar su conocimiento e inspirar a todos.

A mis profesores…
…por trabajar arduamente no solo en formar excelentes ingenieros, si no también excelentes personas.

A mis amigos…
…por siempre estar ahí con una sonrisa, en las buenas y en las malas.

A todos…
…por permitirme servirles

Agradecimientos

Al Dr. Víctor Rangel Licea por su apoyo al ser director de la tesis “Estudio de capacidad de transmisión y propagación de IoT”.

Al proyecto EWIN, por la oportunidad y el apoyo recibido durante la realización del proyecto de tesis “Estudio de capacidad de transmisión y propagación de IoT”.
Ref EP/P029221/1-1 -J15413/14/15, Ref UNAM: 49547-2363-12-IX-17

A la Facultad de Ingeniería de la UNAM, por el apoyo recibido durante la realización del Proyecto de Investigación y Desarrollo “Estudio de capacidad de transmisión y propagación de IoT”.

Tabla de Contenido

Índice de figuras

Índice de Tablas

Acrónimos y abreviaciones

Acrónimo/abreviación	Significado
AES	Advanced Encryption Standard.
API	Application Programming Interface (Interfaz de programación de aplicaciones).
APN	Access Point Name (Nombre de punto de acceso).
APS	Application Support Sub- Layer (Sub- capa de soporte de Aplicación).
CIU	La Unidad de Inteligencia Competitiva.
CSMA/CA	Carrier Sense multiple access with Collision Avoidance (Censado de portadora de múltiple acceso con evasión de colisiones).
DC	Direct Current (corriente directa).
DM	DigiMesh.
EWIN	Red de Información de Emergencia sobre el Agua.
I2C	Inter- Integrated Circuit (Comunicación entre circuitos integrados).
IDE	Entorno de Desarrollo Integrado.
IEEE	Institute of Electrical & Electronics Engineers (Instituto de Ingenieros Electricos y Electrónicos).
INEGI	Instituto Nacional de Estadística y Geografía.
IoT	Internet de las Cosas.
IP	Internet Protocol (Protocolo de Internet).
ITU	Unión Internacional de telecomunicaciones.
LOS	Line Of Sight (Linea de vista).
MAC	Medium Access Control.
MDA	Máxima Distancia Aceptable.
MIT	Massachusetts Institute of Technology (Instituto Tecnológico de Massachusetts).
MOSFET	Transistor de efecto de campo de Metal- Oxido- Semiconductor.
NM	Nodo Móvil
OSI	Open System Interconection (Interconexión de sistemas abiertos).
P2P	Point to Point (punto a punto).
PAN	Personal Area Networks.
PER	Packet Error Rate (Tasa de Error del Paquete).
PHY	Physical.
PWM	Pulse Width Modulation (modulación de ancho de pulso).
RFID	Identificación por Radio Frecuencia.
RL	Return Loss (perdida de regreso).
RP	Repeater (Puede tradusirse como repetidor o redireccionador).
RSSI	Received Signal Strenght Indicator (Indicador de intensidad de señal recibida.)
SIM	Subscriber Identification Module (Modulo de identificación de subscriptor).
SOI	Silicio sobre Aislante.
SPI	Serial Peripherial Inteface (Interfaz Serial Periférico).
SRAM	Static Random Access Memory (Memoria de acceso aleatorio estático).
SWR	Standing wave ratio (relación de onda estacionaria).
TCP	Transmission Control Protocol (Protocolo de Control de Transmisión).
TIC	Tecnología de información y comunicación.
USB	Bus Serial Universal.
XCTU	Next Generation Configuration Platform for Xbee/RF Solutions
ZB	Zigbee.
ZDO	Zigbee Device Objects (Objetos del dispositivo Zigbee.

Capítulo 1. Introducción

Como parte fundamental del presente trabajo, este capítulo trata los antecedentes y el estado del arte como un enfoque del tema. Posteriormente, el problema y el alcance son evidenciados en las secciones de Definición del problema y objetivo, respectivamente. Finalmente, la estructura de la tesis es detallada.

1.1. Antecedentes

El uso del celular por parte de la población actual es tan común, que las nuevas generaciones consideran el no tenerlo como una situación anormal y nada placentera, los datos reflejan 127 millones de líneas activas en México para Agosto del 2021[1]. En menor medida, el 44.9% de mexicanos cuentan con computadoras y 52.9% de mexicanos tienen acceso a Internet en sus hogares[2]. El incremento de '*wearables*' (Dispositivos electrónicos 'vestibles'[3]) en la sociedad mexicana también es de renombre, quien solo en 2015 vendió 179,030 unidades[4]. A lo largo de los años, los '*wearables*' se han ido popularizando enfocado a relojes y bandas inteligentes, así como con audífonos que han llegado al punto de diseño de los '*Airpods*' por parte de Apple o los '*Airdots*' por parte de Xiaomi. Junto con las nuevas tecnologías entrantes del 5G (quinta generación de comunicaciones), es concluyente que el mundo está en una revolución tecnológica, en el cual, México está incluido.

Los datos anteriormente mencionados, más el aumento de las comunicaciones debido a pandemia, la cual forzó a muchos mexicanos a continuar su día a día remotamente; denota un cambio en la población mexicana (en la cual está centrado este trabajo). La interacción entre las personas y la tecnología favorece la familiarización con mayores avances emergentes, facilitando los procesos repetitivos con precisión y apoyando a la humanidad en complicaciones como en la contingencia actual. Si bien, es cierto que las zonas con mayor concentración poblacional tendrán mayor interacción, la tecnología fomenta la posibilidad de despliegue, de forma que abarque más y más zonas.

Los desarrollos de las TIC's (Tecnologías de información y comunicación) no solo pueden ser enunciados en materia de '*streaming*' o redes sociales. Las implementaciones orientadas a las redes IoT (Internet de las cosas) son bastas y de gran importancia, muchas de las tecnologías actuales están dirigidas a maximizar estas redes debido a su versatilidad y beneficios. Favoreciendo la existencia de coches, casas y hasta ciudades inteligentes; donde la automatización, el ahorro de energía y de costes, entre muchos otros; son sus principales beneficios.

Las redes de IoT obtienen un impacto critico al favorecer el desarrollo de las telecomunicaciones de emergencia bajo las recomendaciones de la ITU(Unión Internacional de las telecomunicaciones)[5]. Este desarrollo, conlleva a mejoras sustanciales y de alto impacto ante la sociedad debido a que abre la posibilidad de redes de IoT, automatizadas y de bajo costo; que realizan un sensado constante para la prevención de desastres naturales, que atenten contra vidas humanas y/o daños irreparables en las comunidades cercanas.

1.2. Estado del arte

El IoT ha sido definida por la ITU en la recomendación ITU-T Y.2060 como "Infraestructura mundial para la sociedad de la información que propicia la prestación de servicios avanzados mediante la interconexión de objetos (físicos y virtuales) gracias a la interoperatividad de tecnologías de la información y la comunicación presentes y futuras"[6]. Si bien, el concepto de interconectar dispositivos bajo el nombre de *Internet Incrustado* ya se tenía desde los años 70's, no fue hasta que en 1982 una máquina de coca cola fue programada para determinar si había bebidas en la máquina o si estaba vacía[7][8].

Después de la máquina de Coca, diferentes artículos mostraban avances en los diversos aspectos del tema, en los que el objetivo era la comunicación entre dispositivos para automatizar todo tipo de procesos en la industria y en la vida diaria[9][10]. Sin embargo, es probable que el término "Internet de las Cosas" fuera acuñado por Kevin Ashton de P&G, posteriormente del Auto-ID Center del MIT (Instituto Tecnológico de Massachusetts), donde consideraba que el RFID (Identificación por Radio Frecuencia) era esencial para el IoT[11]. Lo cierto es, que los avances en tecnologías de comunicación inalámbrica y transistores cada vez más pequeños poco a poco facilitaron la creación de tecnologías IoT[12].

El uso del IoT ha ido en incremento ocupando protocolos de comunicaciones como 'Bluetooth Low Energy', 'WiFi', 'Zigbee' y 'LoRa'. Poniendo al IoT como un punto importarte a nivel global debido a su versatilidad en industrias, agricultura, mercado, casas, medicina, etc. En el año en curso (2021), hay una aproximación de 35.82 billones de dispositivos de IoT, con una estimación de 75.44 billones de dispositivos para el 2025 [13].

De entre los avances más actuales que competen al presente trabajo está la publicación de Li Xiaoman y Lu Xia[14], el artículo presenta del diseño de una red de sensores inalámbricos para la acuacultura usando el protocolo Zigbee. Dicha red pretende ocupar microcontroladores y chips con el protocolo Zigbee para recolectar,

procesar y transmitir la información del ambiente; con tal de facilitar los procesos dentro de la acuacultura. Concluyendo que la red propuesta incluye un soporte robusto para el monitoreo y control en tiempo real.

El artículo presentado por Ala Khalifeh et al[15], hace una comparación entre los protocolos Zigbee y DigiMesh (ambos en una frecuencia de 2.4 GHz) en temas de *throughput*, tiempo que tarda en llegar el mensaje del transmisor al receptor y viceversa, potencia de la señal recibida y el tiempo que tarda en cambiar la ruta tras un incidente en la red. La publicación hace uso de los radios Xbee S1 para el protocolo DigiMesh y Xbee S2 para el protocolo Zigbee, tomando los datos de las mediciones con la herramienta XCTU (Plataforma de configuración de nueva generación para soluciones Xbee o RF) proporcionada por la empresa Digi Inc. que fabrica los radios Xbee. El artículo concluye que DigiMesh presenta mayor Throughput, pero Zigbee lo supera en menor tiempo de retraso en los mensajes y en distancia (120 metros frente a los 40 metros de DigiMesh).

En 2019, Imran A. Zualkernan et al[16], presenta la implementación de un sistema basado en IoT para gestionar y optimizar los escapes de las personas usando microcontroladores “Bluetooth Low Energy” y sensores de humo con redes de WiFi y DigiMesh. Esto con la intención de que las persas puedan encontrar las mejores rutas de escape sin congestiones ni peligros desde una aplicación celular. El artículo señala que la implementación puede ser hasta en la escala de una ciudad.

En el mismo año, Herman Yuliandoko y Abdul Rohman[17] presentaron un Sistema de detección de inundaciones con el monitoreo del agua usando el protocolo Zigbee. Este sistema está basado en Indonesia, y hace uso de las tarjetas con microcontroladores Arduino, junto con un sensor de velocidad para el agua y un módulo detección ultrasónica; usando los radios XBee S2 Pro. La publicación concluye con que el sistema con una distancia en los nodos de 75m es lo óptimo para tener un buen resultado en la antelación de un suceso de inundación.

En México, está en proceso el proyecto EWIN (Red de Información de Emergencia sobre el Agua) [18], el cual es una colaboración de la Universidad Nacional Autónoma de México en conjunto la universidad de Loughborough en Inglaterra, la Universidad de Colima y el Estado de Colima. El proyecto, enfocado más hacia las telecomunicaciones de emergencia, ocupa las tecnologías emergentes LoRa para hacer una red de boyas que midan el nivel de agua en ríos. Con el fin de que, en temporadas de lluvia sobre todo, la red detecte un aumento significativo en el nivel de agua y que este suponga un desborde; la red avise a la localidad o localidades en peligro, para que así se tomen medidas preventivas. Salvando vidas y reduciendo gastos que en un inicio están valorados en millones de dólares.

1.3. Definición del problema

El desarrollo de TIC's y su implementación en México puede potenciar las tecnologías enfocadas en IoT, las cuales a su vez automatizan procesos y reducen costos de manera significativa desde desastres (naturales y artificiales), hasta costos por producción y administración. Por lo que el uso de estas redes es necesario, teniendo en cuenta el manejo de información, el nivel de independencia esperado para el sistema (duración de batería y almacenamiento de información). De igual forma, la optimización de espacio toma relevancia para la versatilidad de los sistemas que pueden ser implementados, así como bajo costo de la implementación.

La existencia de dispositivos para sensar, procesar y comunicarse entre ellos permiten la capacidad de lo mencionado anteriormente; dichos dispositivos pueden ocupar diferentes protocolos para la comunicación. Este trabajo está enfocado en ocupar 2 protocolos de comunicación con el fin de realizar un estudio de la capacidad de transmisión y propagación de estos en su uso de IoT, con el fin de compararlos y presentar los resultados para que sean considerados en futuros estudios o implementaciones.

1.4. Objetivos

1.4.1. Objetivo General

El objetivo de este trabajo consiste en realizar un estudio acerca de la capacidad de transmisión empleando 2 protocolos de comunicación: el protocolo Zigbee y el protocolo DigiMesh, con el propósito de determinar cuál sería el protocolo de comunicación más robusto para ser usado en monitoreo de ríos. Zigbee es uno de los protocolos más usados para IoT y DigiMesh, un protocolo emergente por parte de la compañía Digi International. Ambos serán puestos a prueba en un entorno de campo abierto, ocupando las series 2 y 3 de Xbee respectivamente, junto con la tarjeta Arduino (en su variante Leonardo) como procesador de datos.

1.4.2. Objetivos particulares

- Observar la intensidad de señal recibida por parte de los radios durante la transmisión de paquetes de cada uno de los protocolos de comunicación.
- Buscar la distancia máxima de comunicación entre 2 radios.
- Observar el desempeño de la comunicación ante un salto por medio de un repetidor.
- Analizar el rendimiento de las comunicaciones ante el manejo de varios transmisores enviando a un solo receptor.
- Analizar el comportamiento de las comunicaciones entre 2 radios, donde uno de ellos es puesto a movimiento constante.

1.5. Contribuciones

Este trabajo pretende servir como contribución al realizar un estudio donde no solo los protocolos sean puestos a prueba de una forma práctica, sino también los radios y microprocesadores en un entorno no ideal. Esto con la finalidad de conocer el desempeño de las tecnologías emergentes para satisfacer las necesidades de personas físicas y morales que consideren implementar las mismas, ya sea por sus usos de gestión inteligente de producción hasta redes de control ante desastres naturales.

1.6. Estructura de tesis

La estructura de la tesis consta de 7 capítulos, los cuales son presentados de la siguiente forma:

- El Capítulo 2, abarca la descripción los protocolos de comunicación empleados en la presente tesis.
- El capítulo 3 describe el *Hardware* y el *Software* empleado con especificaciones técnicas y configuraciones para los dispositivos usados en este trabajo.
- El Capítulo 4 presenta el desarrollo y los procedimientos de las mediciones para cada uno de los 5 escenarios empleados, así como de las especificaciones a considerar para cada medición y cada protocolo empleado.
- El Capítulo 5, presenta los puntos de medición, resultados y análisis de los primeros 2 escenarios de las mediciones. El capítulo 6 abarca los resultados de los 2 últimos escenarios.
- Finalmente, el Capítulo 7, concluirá el trabajo con discusiones finales, trabajo futuro, las contribuciones de este trabajo y conclusiones finales. Como añadidos, la información extra usada en este trabajo es depositada en 3 anexos.

Capítulo 2. Protocolos de Comunicación

2.1. Introducción

Los protocolos de comunicaciones son una serie de reglas para que un mínimo de 2 entes puedan transmitir información. En sistemas de telecomunicaciones son de vital importancia, pues estos indican la forma en la que las computadoras realizarán sus transmisiones a través de cualquier medio, siendo el más ocupado para comunicaciones el Internet.

La base del presente trabajo es sustentada por el estudio y la comparación de 2 protocolos: Zigbee y DigiMesh, ambos orientados a usos en redes IoT. Debido a esto y a la importancia que representa el conocer brevemente los protocolos, este capítulo abarcará aspectos generales de cada protocolo, junto con algunas especificaciones que resulten de importancia.

2.2. Zigbee

Zigbee (Logo en la Figura 1) es un protocolo de comunicación inalámbrica enfocado a redes IoT. Dicho protocolo está basado en la especificación de la IEEE 802.15.4[19] lo cual lo enfoca para redes de área personal, PAN por sus siglas en inglés. Zigbee es un protocolo de bajo consumo de energía, bajo coste y también con bajo tráfico de información (250 Kbits/s, según la Zigbee Alliance).

Figura 1: Logo de Zigbee, obtenido de la pagina principal de la Zigbee Alliance[20]**.**

Al ser un protocolo de malla, las redes que usen Zigbee pueden formarse por sí mismas, al igual que cambiar la ruta automáticamente ante un enlace roto. Zigbee ocupa una frecuencia base de 2.4 GHz que al ser la banda ISM, sigue la regla del espectro distribuido, por lo que su señal aparenta ser ruido para otros protocolos como WiFi y Bluetooth. Zigbee también puede ocupar 16 canales con un ancho de banda de 2 MHz por canal, todas las características mencionadas dan una oportunidad teórica de poder manejar redes de hasta 65,000 nodos (según establece la Zigbee Alliance) .

La Zigbee Alliance es un grupo de compañías que mantienen, publican y actualizan el protocolo Zigbee que fue establecida en el 2002. Actualmente hay más de 500 compañías, de entre las cuales destacan: Comcast, Ikea, Samsung SmartThings y Amazon. Dentro de las especificaciones provistas por la Alianza, agregan que, además de las especificaciones ya mencionadas, el protocolo tiene un rango de comunicación de más de 300 metros en LOS(Línea de vista) y llega hasta los 100 metros en interiores. La Alianza igual revela que cuenta con una encriptación AES-128 (Estándar avanzado de encriptación) en la capa de Red.

El protocolo está pensado para automatizaciones en hogares y edificios, control en procesos industriales, sensores con información médica y hasta juguetes.

2.2.1. Topología

La topología de una red Zigbee es de tipo malla y centralizada y los nodos de la red pueden están conformados por un Coordinador (C) que es el nodo principal de la red. Este nodo tiene como funciones el configurar y agregar elementos a la red. El coordinador está acompañado de Routers (R) los cuales mediarán el tráfico de la red, harán los mapeos de esta, así como la ruta que la información tendrá que seguir para llegar a su destino. Al final están los End Devices (E ó ED) los cuales capturan información y la transmiten hacia los routers para que llegue a su destino, ya sea al coordinador, otros End Device o Routers. Los End Devices aparte pueden dormir y leer los mensajes que recibieron tras dejar de dormir.

La Figura 2 muestra un esquema alusivo a una conexión en malla bajo el protocolo Zigbee, donde los nodos están interconectados según su jerarquía.

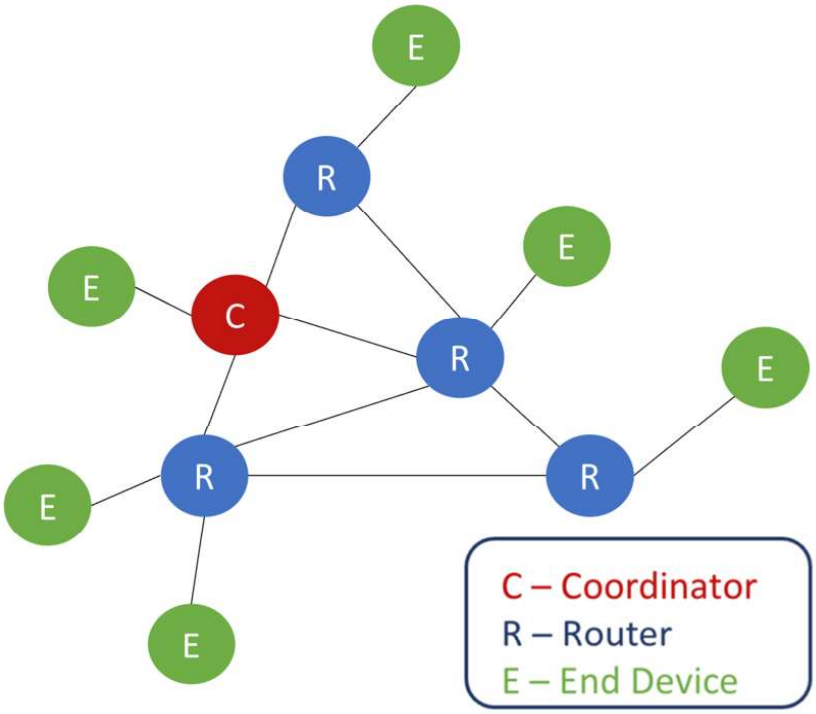

Figura 2: Topología de una red Zigbee en su modelo de malla.

2.2.2. Arquitectura

La arquitectura muestra la forma en como el protocolo interactúa con la información a través del modelo OSI (Interconexión de sistemas abiertos). Dicho modelo menciona 7 capas para una comunicación digital: empezando desde la capa física (PHY) la cual se traduce como antenas o cables, pasando por el acceso el medio (MAC), la red en la que se encuentra (NWK), el transporte, la sesión, la presentación y termina en la capa de aplicación. En la mayoría de las ocasiones, el usuario final solo tendrá interacción a la capa de aplicación que es la interfaz que todo mundo observa en sus celulares y computadoras.

Dependiendo del protocolo o el estándar, será el número de capas que tendrán especificaciones. Debido a que Zigbee está basado en el estándar 802.15.4 de la IEEE, éste inicialmente define las 2 capas inferiores (capa física y la capa de acceso al medio). La Zigbee Alliance, durante su fundación, construyó la capa de red y un *framework* (conjunto de conceptos y prácticas para tomar referencia, también es conocido como 'Entorno de trabajo') para la capa de aplicación, la cual consiste en una sub-capa de soporte de aplicación (APS) y los objetos del dispositivo Zigbee (ZDO).

IEEE 802.15.4 establece que la capa física (PHY) puede trabajar a 2 frecuencias: 868 MHz (para Europa) / 915 MHz (para países como USA y Australia) y 2.4 GHz(usada en todo el mundo). Por su parte, la capa de acceso al medio (MAC) controla el acceso al canal del radio por medio de CSMA-CA (protocolo de comunicación de capa MAC que sensa las señales y evade las colisiones entre paquetes iniciando la comunicación una vez

que el canal quede libre); también se encarga de mandar información de la red (para mantenimiento), sincronización y provee un mecanismo de transmisión confiable.

La Figura 3 muestra, en un diagrama a bloques; las diferentes capas, interfaces y funciones definidos en el protocolo Zigbee; desde los definidos en el protocolo de la IEEE 802.15.4, pasando por los definidos por la Zigbee Alliance, hasta los que son definidos por los desarrolladores al momento de ocupar el protocolo en sus productos.

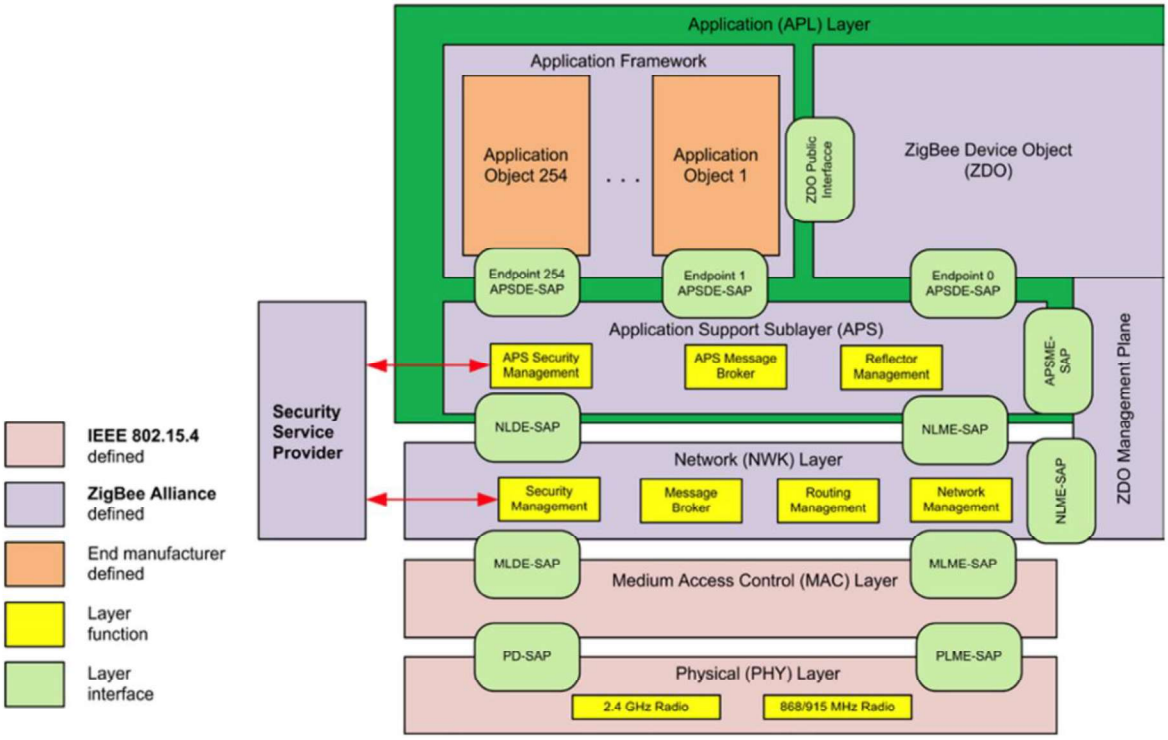

Figura 3: Arquitectura de Zigbee, obtenida de la especificación de Zigbee, pagina 2[20].

2.2.3. Frame

Los *frames* pueden definirse como una plantilla para los bits con diferentes y específicos campos, de tal forma que, siguiendo su orden, las comunicaciones bajo éste protocolo puedan "entenderse de forma automática".

Para el caso de Zigbee, estos formatos competen al APS(Sub-capa de soporte de Aplicación), en el cual cada *frame* tiene 2 componentes básicos:

- El APS *header*, que contiene la información del control y el direccionamiento.
- El APS *payload*, que contiene información del tipo del frame y tiene una longitud variable.

Como fue mencionado, el *frame* tiene una secuencia de campos en un orden específico. Los *frames* mostrados en la Figura 4 y en la Figura 5 están representados en la forma en la que son transmitidos desde la capa NWK, de izquierda a derecha (empezando por izquierda) y son numerados iniciando desde el 0 y pueden ser separados en 8 bits, también llamado Octeto o *Byte*. La Figura 6 muestra los tipos de mensajes que pueden existir, donde el tipo de *frame* corresponde al tipo de mensaje posible. La Figura 7, muestra los modelos de entrega, lo que se traduce a si el mensaje iba directamente al receptor, a todo un grupo de nodos o en *broadcast* (a toda la red).

Si hay campos con mayor número de *bits* que un octeto, estos se mandarás a la capa NWK en orden, desde el octeto con menor número de *bits*, hasta el octeto con mayor número de bits. Los campos nombrados como "*Reserved*" deberán quedar a 0 ó el mensaje será descartado durante la transmisión[20].

<table>
<tr><td>Octets: 1</td><td>0/1</td><td>0/2</td><td>0/2</td><td>0/2</td><td>0/1</td><td>1</td><td>0/
Variable</td><td>Variable</td></tr>
<tr><td rowspan="2">Frame control</td><td>Destination endpoint</td><td>Group address</td><td>Cluster identifier</td><td>Profile identifier</td><td>Source endpoint</td><td rowspan="2">APS counter</td><td rowspan="2">Extended header</td><td rowspan="2">Frame payload</td></tr>
<tr><td colspan="5">Addressing fields</td></tr>
<tr><td colspan="8">APS header</td><td>APS pay-load</td></tr>
</table>

Figura 4: Formato general de los *frames* en Zigbee, obtenida de la especificación de Zigbee, pagina 44[20]**.**

Bits: 0-1	**2-3**	**4**	**5**	**6**	**7**
Frame type	Delivery mode	Ack. format	Security	Ack. request	Extended header present

Figura 5: Formato general del *frame* en el campo de control, obtenida de la especificación de Zigbee, pagina 45[20]**.**

Frame Type Value b_1 b_0	Frame Type Name
00	Data
01	Command
10	Acknowledgement
11	Inter-PAN APS

Figura 6: Tipos de Mensajes para el campo de *'Frame types'* del *frame* de control, obtenida de la especificación de Zigbee, pagina 45[20]**.**

Delivery Mode Value b_1 b_0	Delivery Mode Name
00	Normal unicast delivery
01	Reserved
10	Broadcast
11	Group addressing

Figura 7: Modelos de entrega para el campo *'Delivery mode'* del *frame* de control, obtenida de la especificación de Zigbee, pagina 45[20]**.**

2.3. DigiMesh

DigiMesh, es un protocolo de comunicación inalámbrica orientado a redes IoT. DigiMesh es propiedad de la compañía Digi Intenational (logo en la Figura 8) y ésta se reserva los derechos e información de su protocolo, sin embargo, el proveedor pone al alcance cierta información relevante.

Figura 8: Logo de la empresa Digi International, obtenida de la página principal de Digi International [21]**.**

El protocolo es, de igual forma que Zigbee, un protocolo de malla, misma razón porque igual puede formarse y cambiar rutas ante un enlace roto de forma automáticamente. Opera con una frecuencia base tanto de 915 MHz como de 2.4 GHz y la empresa especifica que pueden crearse redes de 128 Nodos, aunque puede ofrecer optimización de RF para redes de más de 1000 nodos.

Digi International especifica con sus radios (Xbee S3B), un alcance hasta 15.5 km en LOS (con una velocidad de 10 kbps y antenas dipolo de 2.1 dBs). Cuenta con una tasa de transmisión máxima de hasta 200 kbps (con la que se reduce el alcancé de los radios hasta 6.5 km en LOS) y al igual que Zigbee trabaja con un bajo costo de energía (desde 2.5 μA hasta 290 mA). La empresa también especifica una disponibilidad de 64 canales, direcciones de hasta 64 bits y una encriptación de 128 AES (256 AES con algunos productos).

2.3.1. *Topología*

DigiMesh ocupa una topología de tipo malla y descentralizada, esto significa que a diferencia de Zigbee, sus nodos no tienen una jerarquía establecida y todos pueden tener las mismas habilidades de ruteo (mostrado en la Figura 9). Esto le da ventajas ante otros protocolos; ya que cualquier nodo puede ser reemplazable. La red es independiente en su totalidad y todos sus nodos pueden entrar en modo sueño, donde consumen menor energía, de forma periódica o no periódica (pero con una forma alámbrica para despertarlo o perdiendo la posibilidad de dormir toda la red).

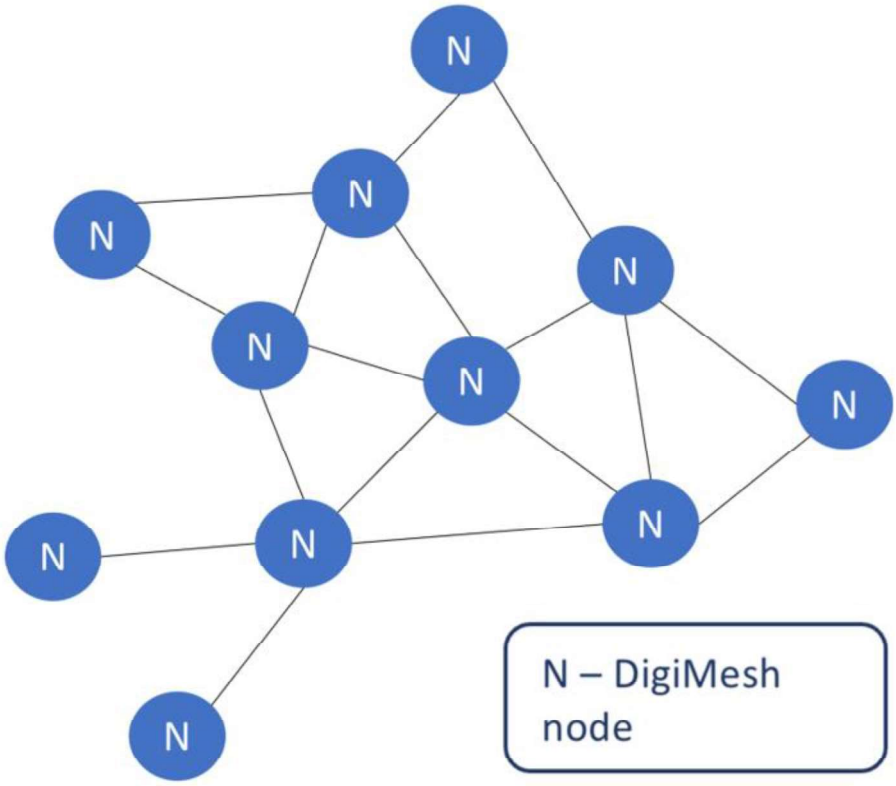

Figura 9: Topología de una red DigiMesh en su modelo de malla.

2.3.2. *Arquitectura*

La empresa Digi International se reserva las especificaciones de la arquitectura del protocolo. Sin embargo, la documentación del radio Xbee S3B ofrece una visión en bloques del *Frimware* con el que trabaja el radio, lo cual es lo más similar al modelo OSI que puede obtenerse durante la realización del presente trabajo (Figura 10) con el que trabaja el radio. Destaca la capa PHY y MAC con la capacidad de recepción punto multipunto, la capa NWK con el protocolo DigiMesh y los 3 modos con los que puede operar.

Los comandos AT es la forma en cómo puede configurar parámetros de la red de forma alámbrica o inalámbrica, el modo API (Interfaz de programación de aplicaciones) ocupa mayor seguridad en las comunicaciones y el modo transparente donde cualquiera puede tener acceso a la información de los radios (que es el ocupado durante la realización de las mediciones).

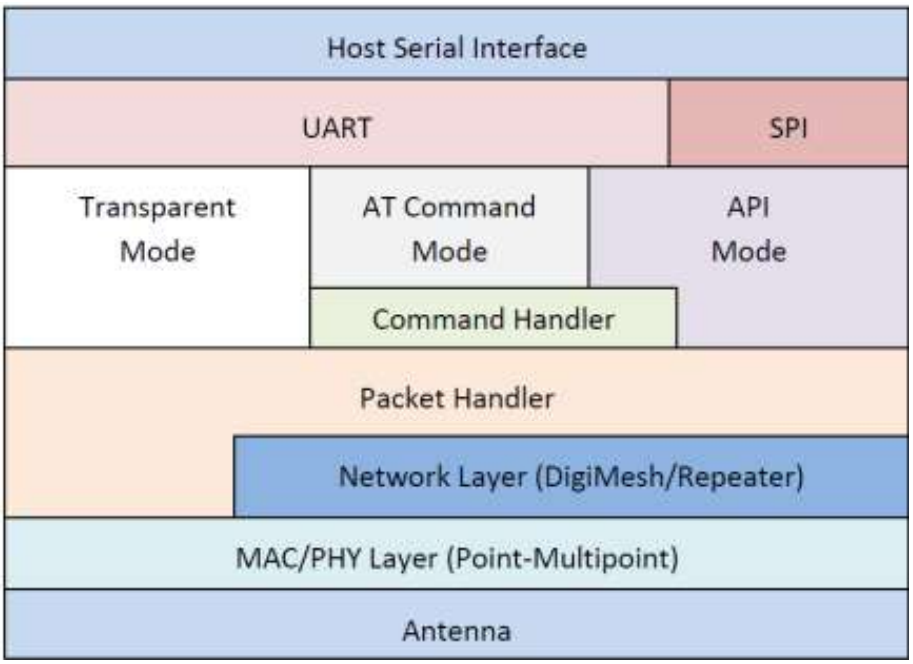

Figura 10: Arquitectura del radio Xbee S3B, obtenida de las especificaciones de radio XBee S3B, página 37[22]**.**

2.3.3. Frame

Los radios Xbee S3B tienen 2 modos para las comunicaciones de la red: el modo transparente y el modo API.

El modo Transparente es empleado en el trabajo, debido a que permite que la información recibida en el radio pueda ser captada por Arduino a través de pines de información del radio. Este modo es el más inseguro, pero es la forma en la que es posible tener lectura de las mediciones.

El modo API está basado en *frames*, con los cuales es posible tener más información sobre la red, así como mayor seguridad. Este modo es ideal para controlar grandes redes de nodos, obtener información de diferentes remitentes o configurar la red a distancia. A diferencia del modo transparente, la información no es observable a través de los pines de los radios por Arduino (Solo con ciertas bibliotecas para Zigbee).

Cabe destacar que los *frames* en modo API, no corresponden al protocolo propiamente dicho, si no a una configuración en la capa de aplicación para mejorar/facilitar la comunicación entre radios. Este modo API puede usarse con otros protocolos que manejen los radios, tales como el Zigbee o directamente el 802.15.4.

En el caso exclusivo de usar API, el formato de los *frames* es mostrado en la Figura 11, mientras que la Figura 12 y la Figura 13 muestran los tipos de mensajes que pueden ser mandados al radio o los que se pueden recibir de él[22].

Start delimiter	Length		Frame data								Checksum
			API identifier	Identifier-specific Data							
1	2	3	4	5	6	7	8	9	...	n	n+1
0x7E	MSB	LSB	cmdID	cmdData							Single byte

Figura 11: Formato del *frame* usado por los radios Xbee S3B en el modo API, obtenida de las especificaciones de radio XBee S3B, página 117[22]**.**

API frame names	API ID
AT Command	0x08
AT Command - Queue Parameter Value	0x09
TX Request	0x10
Explicit TX Request	0x11
Remote Command Request	0x17

Figura 12: Tipos de mensajes que pueden ser mandados al dispositivo, obtenida de las especificaciones de radio XBee S3B, página 118[22]**.**

API frame names	API ID
AT Command Response	0x88
Modem Status	0x8A
Transmit Status	0x8B
Route information packet	0x8D
Aggregate Addressing Update frame	0x8E
RX Indicator (AO=0)	0x90
Explicit Rx Indicator (AO=1)	0x91
Data Sample Rx Indicator frame	0x92
Node Identification Indicator (AO=0)	0x95
Remote Command Response	0x97

Figura 13: Tipos de mensajes que pueden ser recibidos del dispositivo, obtenida de las especificaciones de radio XBee S3B, página 118[22]**.**

Capítulo 3. Descripción del Hardware y Software empleado

3.1. Introducción

Este trabajo se sostiene mediante la información experimental. Por lo que la descripción del *hardware* (comprendido como radios y procesadores) es de vital importancia. Conocer los parámetros físicos es de utilidad para formular especulaciones en los resultados con respecto a las limitantes.

La parte de *software* en este trabajo toma importancia para conocer la configuración de los protocolos en los radios, así como las herramientas destinadas a digitalizar y visualizar la información obtenida durante los escenarios. Por lo que describirlos favorecerá el entendimiento de las consideraciones y procedimientos durante la realización de las mediciones en general.

3.2. Hardware

En la parte del *hardware*, 2 tipos de radios y una tarjeta Arduino son empleados. Los radios están fabricados por la compañía Digi International, misma que se especializa en la realización de radios que puedan comunicarse inalámbricamente. La empresa es considerada una de las compañías emergentes enfocadas a la tecnología IoT[21].

La tarjeta Arduino, por su parte, es una tarjeta de bajo costo que tiene un microprocesador, así como diversas salidas y entradas de voltaje. El microprocesador puede interpretar programas en el lenguaje de programación C y C++, estos son cargados mediante una comunicación serial de un puerto USB (Bus Serial Universal) desde la computadora con la ayuda de un IDE (Entorno de Desarrollo Integrado). Debido a su versatilidad de entradas y salidas, múltiples módulos y bibliotecas han sido diseñadas para

conectarse a las diferentes tarjetas de Arduino. Estos módulos crean una serie de dispositivos capaces de realizar múltiples acciones y mediciones de diversos tipos; tales como medir temperatura, proximidad, humedad, mandar datos, recibir datos, encender un LED, etc[23].

3.2.1. Arduino Leonardo

Arduino presenta una amplia gama de tarjetas de desarrollo con costos accesibles que permiten su acceso a niveles de educación, siendo el modelo Arduino UNO de entre los más usados y populares. La tarjeta Arduino Leonardo, mostrada en la Figura 14, asemeja al Arduino uno en dimensiones y cantidad de pines (20), por lo que en un principio aparenta ser lo mismo, ambos ocupan microcontroladores Atmel y tienen las mismas características de alimentación.

Figura 14: Arduino Leonardo, obtenido del sitio de compras de Arduino [24]**.**

El trabajo hace uso de varias tarjetas Leonardo debido a que resultan más versátiles en comparación con el Arduino UNO; debido a la conexión con la computadora, el número de entradas analógicas, salidas PWM (Modulación por ancho de pulso), interrupciones y el aumento de SRAM (Memoria de acceso aleatorio estático). Algunas especificaciones de internet están representadas en la Tabla 1.

El proveedor, además de las especificaciones, ofrece el diagrama de pines de la tarjeta, donde la Figura 15 resalta los pines digitales; la Figura 16 los pines analógicos, I2C e SPI y PWM; la Figura 17 los pines de alimentación y la Figura 18 un esquema de colores para diferenciar los pines anteriormente mencionados [24].

Tabla 1: Tabla de especificaciones de Arduino Leonardo.

Especificación	Valores [0.5ex]
Microcontrolador	ATmega32u4
Voltaje de Alimentación	7V - 12V
Límites de alimentación	6V - 20V
Pines de entrada/salidas digitales	20
Canales PWM	7
Canales para entrada analógica	12
Corriente DC por cada entrada/salida	40 mA
Corriente DC para pin 3.3V	50 mA
Memoria Flash	32 KB (4KB ocupados para el *bootloader*)
SRAM	2.5KB
EEPROM	1KB
Reloj	16MHz
Longitud	68.66mm
Ancho	53.3mm
Peso	20g

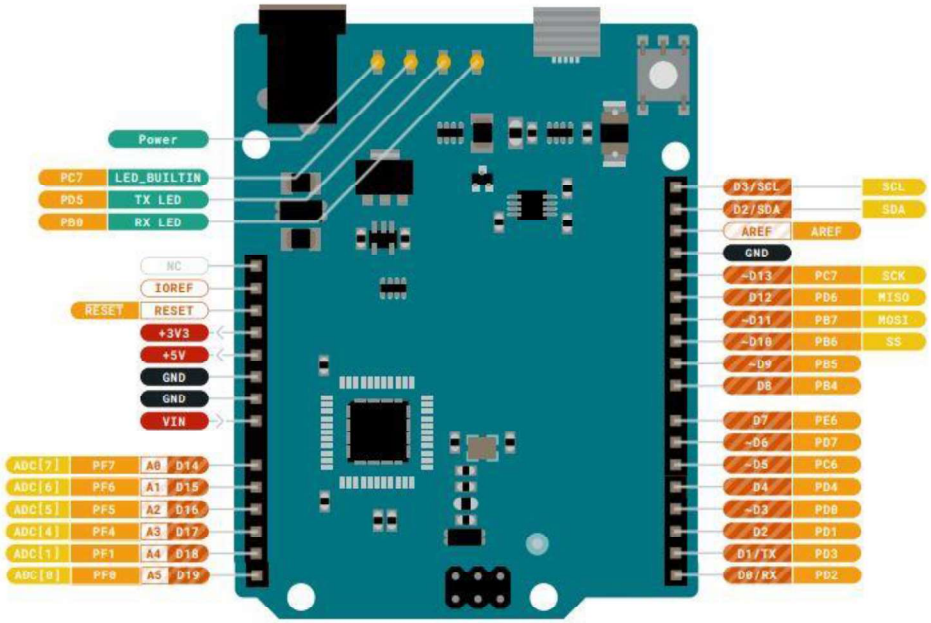

Figura 15: Diagrama de pines digitales de Arduino Leonardo, obtenido del sitio de compras de Arduino [24]**.**

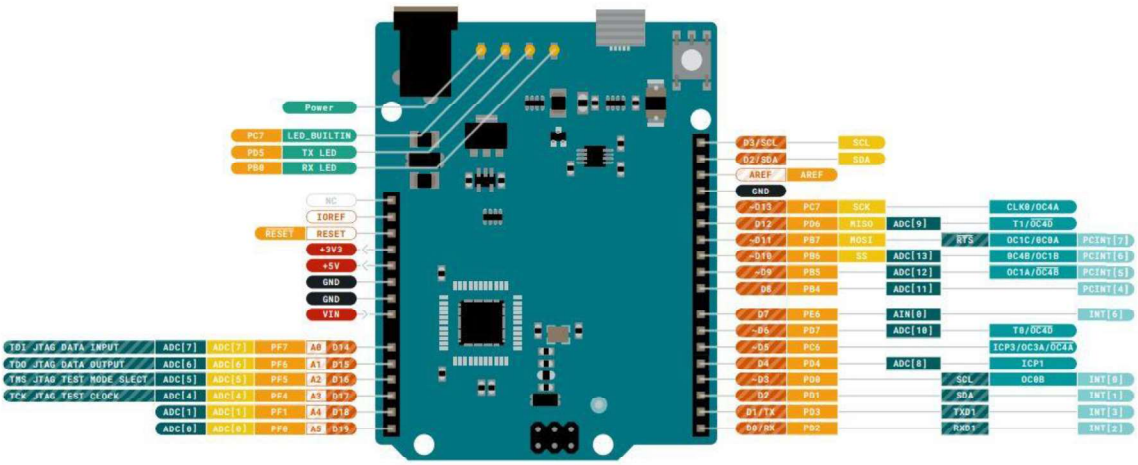

Figura 16: Diagrama de pines analógicos, conexiones I2C e SPI y PWM de Arduino Leonardo, obtenido del sitio de compras de Arduino [24].

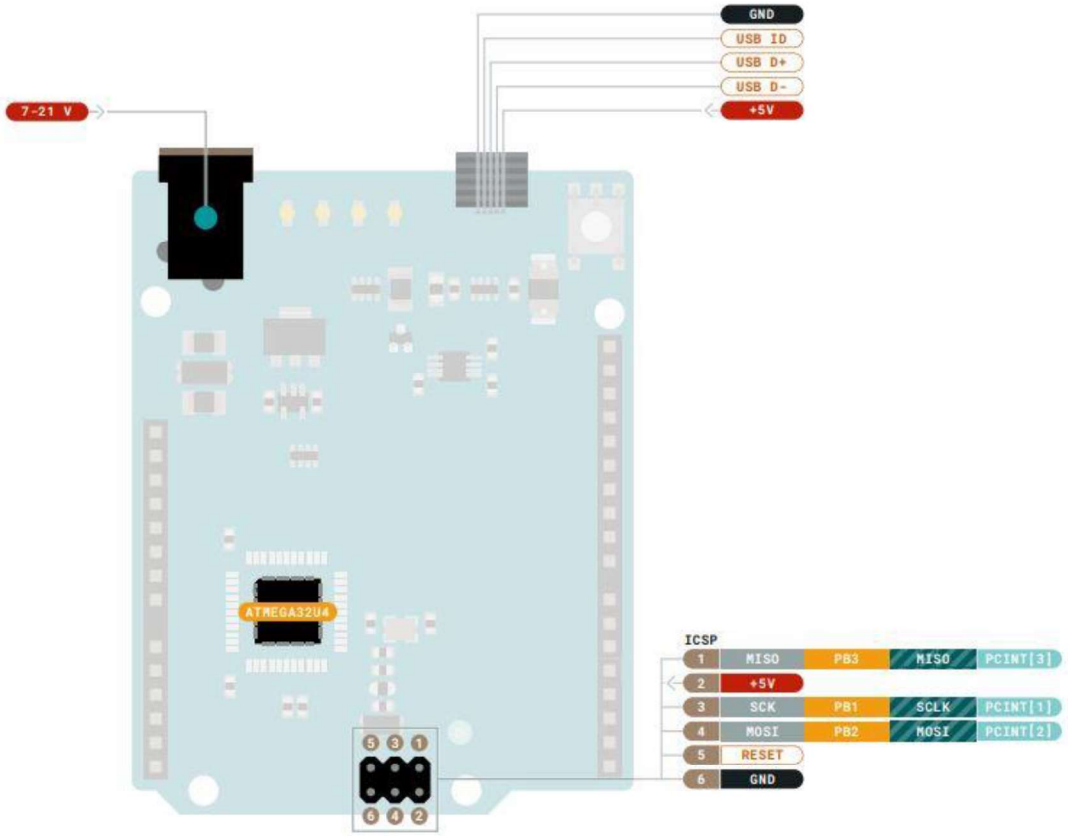

Figura 17: Diagrama de pines para alimentación de Arduino Leonardo, obtenido del sitio de compras de Arduino [24].

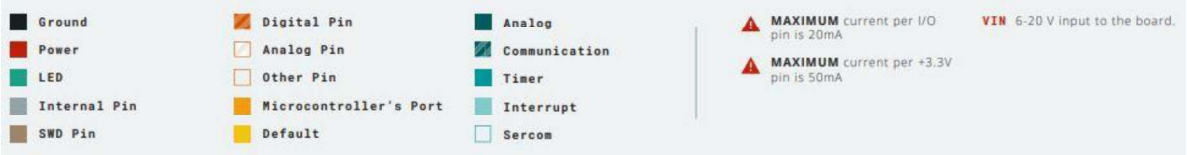

Figura 18: Indicaciones de colores en los diagramas anteriores, obtenido del sitio de compras de Arduino [24].

3.2.2. Xbee S3B

De entre los productos que ofrece Digi International existen los productos Xbee S, una serie de radios para comunicaciones inalámbricas. El Xbee S3B (mostrada en la Figura 19) resulta de utilidad ya que opera en la banda de frecuencias de 915 MHz, así como también soportar el protocolo DigiMesh. Estas características ofrecen una versión diferente a ocuparlo a 2.4 GHz, ya que teóricamente, este radio debe ser capaz de obtener mayores rangos de comunicación debido a la frecuencia y es un buen comparativo con respecto al protocolo Zigbee, que opera a la banda de los 2.4 GHz.

Figura 19: Radio Xbee S3B, obtenido del sitio de compras Mercado Libre[25]**.**

Entre la documentación que ofrece el proveedor, menciona que los Xbee S3B pueden ser parte de una red de hasta 128 nodos de la misma serie. Los radios además cuentan con las especificaciones denotadas en la Tabla 2, las cuales son de relevancia en el trabajo para conocer limitaciones tanto en el desarrollo como en la toma de mediciones:

Tabla 2: Tabla de especificaciones de radio Xbee S3B.

Especificación	Valores
Rango LOS	10kb/s - 15.5km | 200kb/s - 6.5km
Potencia Tx	24dBm - 250mW
Max tasa de Tx	200kb/s
Min tasa de Tx	10kb/s
Interfaz serial UART	CMOS asíncrona con estabilidad en *bauds* de 1%
tasa de datos de la interfase serial	9600 - 230400 *bauds*
Sensibilidad de Rx	-101 dBm (200kb/s) | -110 dBm (10kb/s)
Voltaje de alimentación	2.1 a 3.6 VDC
Corriente en Tx	290mA (PL = 4 + *boost*) - 60mA (PL = 0)
Corriente en idle	29mA - 35 mA
Corriente en sueño (*sleep*)	2.5 µA
Frecuencia de operación	902 - 928 MHz
Dimensiones (sin antena)	3.29cm x 2.44cm x 0.546cm
Peso	5 - 8 gramos (depende de antena)
Temperatura de operación	-40° C a 85 C°
Entradas/Salidas digitales	15
ADC	4 entradas analógicas de 10 bits
Topologías	Mesh, P2P, P2MP, *peer2peer*
Canales	64
Encriptación	AES con 128 bit
Reloj	3.5MHz
Flash *memory*	120kB

Finalmente, el proveedor también tiene una serie de dibujos mecánicos correspondientes al radio, como se aprecia en la Figura 20, la Figura 21 y la Figura 22 [26]. Estas Figuras permiten dimensionar correctamente el concepto físico del radio, el cual es bastante pequeño y, a su vez, versátil.

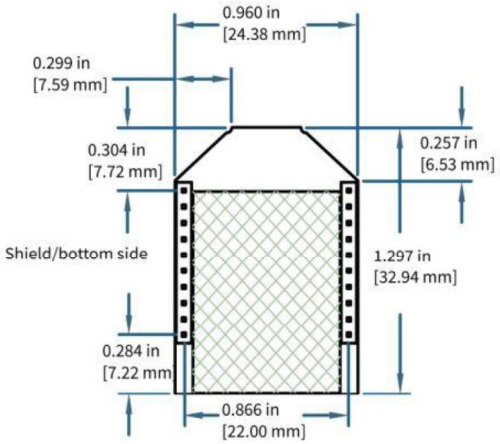

Figura 20: Dibujo transversal superior del radio, obtenida de las especificaciones del radio XBee S3B, página 24[22]**.**

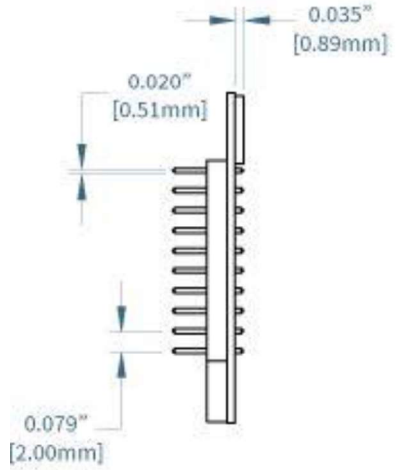

Figura 21: Dibujo sagital izquierdo del radio, obtenida de las especificaciones del radio XBee S3B, página 24[22]**.**

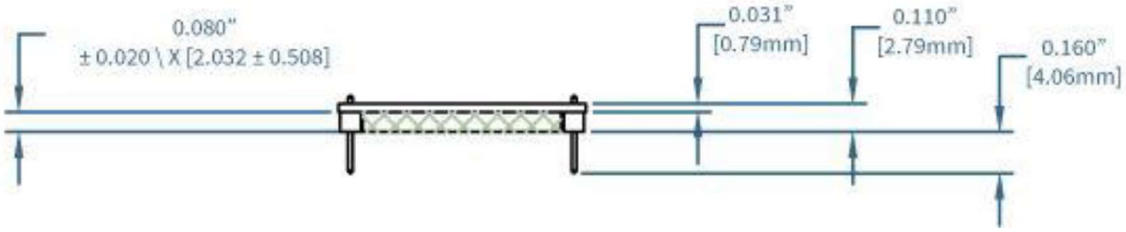

Figura 22: Dibujo frontal anterior del radio, obtenida de las especificaciones del radio XBee S3B, página 24[22]**.**

3.2.3. Xbee S2C

Los radios de las series 1 y 2 proporcionados por Digi International son de entre los más usados debido a que operan en la banda de los 2.4 GHz y ocupan el protocolo Zigbee.

La serie 1 es considerada *Legacy* (termino para algo que ya es obsoleto, pero sigue siendo usado) debido a las nuevas actualizaciones de *firmware* (*software* de bajo nivel que permite el control de un *hardware* específico[27]), y de entre los modelos más recientes se encuentra el Xbee S2C, el cual es mostrado en la Figura 23.

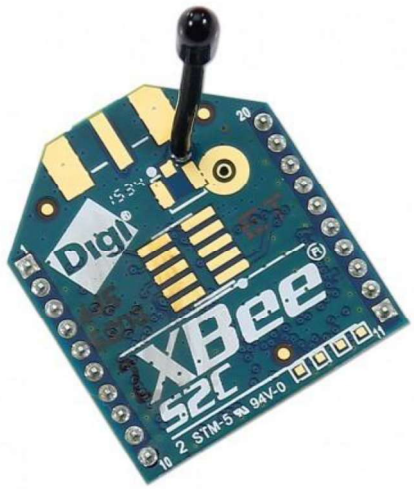

Figura 23: Radio Xbee S2C con antena Whip, obtenida de un sitio de compras "buyhatke!"[28] **.**

El proveedor muestra las especificaciones de suma importancia del radio, como se presentan en la Tabla 3. Estas especificaciones están enfocadas con las variantes de antena que este radio tiene. La tabla expone la variación de la antena Whip, la cual es una antena que viene soldada al radio. Existe la versión del radio con una antena "desmontable", las especificaciones de esta antena son abarcadas más adelante, en este mismo capítulo.

Tabla 3: Tabla de especificaciones de radio Xbee S2C.

Especificación	Valores
Rango LOS	*Indoor* - 60m | *Outdoor* - 1.2km
Potencia de Tx	8dBm - 6.3mW
Max tasa de Tx	250kb/s
Interfaz	250 Kbps(UART) | 5Mbps SPI [1ex]
Tasa de datos de la interfase serial	9600 - 115200 *bauds*
Sensibilidad de Rx	-102 dBm (*boost mode*) | -100 dBm (*normal*)
Voltaje de alimentación	2.1 a 3.6 VDC
Corriente en Tx	45mA (*boost*) - 33mA (*normal*)
Corriente en idle	31mA(*boost*) - 35 mA(*normal*)
Corriente en sueño (*sleep*)	1 µA
Frecuencia de operación	2.4 GHz - 2.5 GHz
Dimensiones (sin antena)	2.2cm x 3.4cm x 0.305cm
Peso	5 - 8 gramos (depende de antena)
Temperatura de operación	-40 °C a 85 °C
Entradas/Salidas digitales	15
ADC	6 entradas analógicas de 10 bits
Topologías	P2P, P2MP, peer2peer, DigiMesh
Canales	11 - 26
Encriptación	AES 128 bits
Reloj	3.3MHz
Flash *memory*	32kB

Dentro de la documentación, el proveedor ofrece los dibujos mecánicos de las diferentes versiones del modelo S2C; como ya fue mencionado, los radios varían en el tipo de antena o en el número de pines. El trabajo hace uso del modelo de 20 pines con antenas U.FL y WireWhip mostrados en los dibujos mecánicos de la Figura 24, la Figura 25 y la Figura 26. Las antenas son abordadas más adelante, dentro de este capítulo.

El proveedor no ofrece un diagrama de las capas del modelo OSI (Interconexión de sistemas abiertos), pero debido a que se trabaja con el protocolo Zigbee, la Figura 3 del capítulo anterior mostraría un diagrama de capas similares con la que el radio debería trabajar, siendo el modelo provisto por la Zigbee Alliance [26].

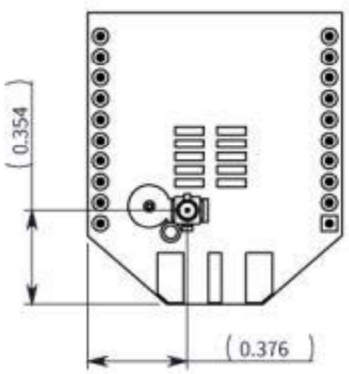

Figura 24: Dibujo transversal superior del radio con variante de antena U.FL, obtenido de las especificaciones del radio XBee S2C, página 40[26]**.**

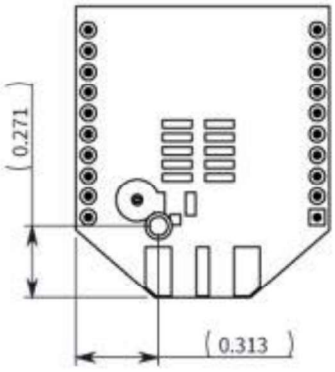

Figura 25: Dibujo sagital izquierdo del radio con variante WireWhip, obtenido de las especificaciones del radio XBee S2C, página 40[26]**.**

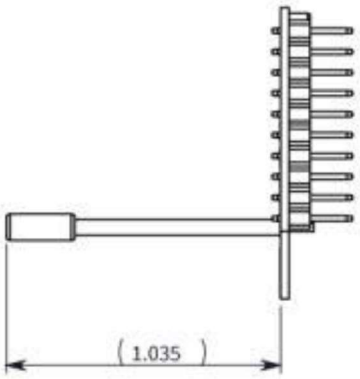

Figura 26: Dibujo transversal superior del radio con variante WireWhip, obtenido de las especificaciones del radio XBee S2C, página 40[26]**.**

3.2.4. Antenas

Este trabajo hace la comparación de los protocolos Zigbee a 2.4 GHz y de DigiMesh a 915 MHz, para los que se usaron 2 tipos de antenas para los radios Xbee S2C (Zibee), usando las variaciones WireWhip y U.FL(la antena “desmontable”). Los radios Xbee S3B (DigiMesh) solo ocuparon la variante U.FL.

3.2.4.1. WireWhip

La antena WireWhip es muy similar a lo que es una antena dipolo, esta antena da la primera apariencia de estar soldada al radio, lo que ayudaría a tener un SWR (Relación de onda estacionaria) bajo. Puede estar tanto en los Xbee normales como en los PRO, teniendo una diferencia de potencia de transmisión (1mW y 60mW respectivamente). Dado que el presente trabajo fue realizado con un radio estándar (no PRO), las especificaciones a ser resaltadas son el tipo de radio ya mencionado.

Las pruebas del desempeño de esta antena, las cuales son mencionadas por los proveedores, involucran 3 escenarios: 2 para interiores y 1 para exteriores. Resultan, respectivamente, en un edificio de oficinas, un almacén para los interiores (ejemplos de los escenarios en la Figura 27 y en la Figura 28) y un parque de negocios (*business park*) con árboles y edificios de diferentes alturas (Figura 29).

Figura 27: Ejemplo de Almacén (locación desconocida).

Figura 28: Ejemplo de edificio de oficina, Grzybowska, Polonia.

Figura 29: Ejemplo de Parque de Negocios, Cabrillo *business park*, California, EUA.

Los resultados obtenidos en estas pruebas, en materia de rango de alcance, son los siguientes:

Tabla 4: Tabla de rangos de la antena WireWhip.

Escenario	Rango con Whip	Rango con Whip y PRO
Almacén	26m	108m
Edificio de oficinas	24m	43m
Parque de negocios (LOS Visual)	258m	1335m

Finalmente, el proveedor proporciona una vista al patrón de radiación plano de la antena WireWhip (Figura 30), la importancia de este patrón reside en cómo es que la energía es radiada por parte de la antena. Con el patrón de radiación es posible conocer el comportamiento de las señales irradiadas por la antena. La antena posee similitudes físicas a un monopolo, y la radiación tiende a ser de una potencia similar en todas las direcciones de forma horizontal. En el caso de la antena WireWhip, aparecen inconsistencias por un par de lóbulos en los 0° de la imagen (eje horizontal, del lado derecho), sugerente a una muy ligera directividad hacia el frente de la antena[29].

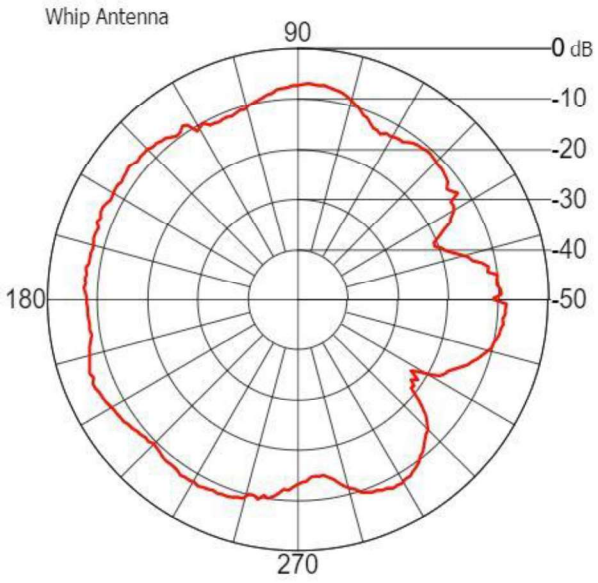

Figura 30: Patrón de Radiación de Antena WireWhip, obtenido de las consideraciones de la antena para XBee [29]**.**

3.2.4.2. U.FL

3.2.4.2.1 2.4 GHz

Este trabajo empleó también el uso de la variante del conector U.FL para el radio Xbee 2SC con el protocolo Xbee. La antena usada, la llamada FXP74, es proporcionada por

la empresa Taoglas (Figura 31). Por ser "Desmontable", el proveedor tiene una serie de especificaciones sobre la antena, mostradas en la Tabla 5.

Tabla 5: Tabla de especificaciones de la antena FXP74.

Especificación	Valores
Rango de frecuencia	2.4GHz - 2.483GHz (depende de protocolo)
Perdida de Retorno (RL)	< -20 dB
Eficiencia	50%
Ganancia	4 dBi
Impedancia	50Ω
VSWR	≤ 2:1
Polarización	Lineal
Límite de potencia	5 W
Temperatura de operación/guardado	-40 a 85 ° C
Dimensiones	47x7x0.1mm
Peso	1.2g
Conector	MHFII (compatible con U.FL)
Estándar del cable	Mini-Coaxial 1.13 mm
Adhesivo	3M 467

Figura 31: Antena FXP74, obtenida de las especificaciones de la antena, página 1[30]

La Figura 32 es una gráfica, obtenida de la documentación del proveedor, enfocada en mostrar la potencia de pérdida ante la reflexión por parte de la antena al ser conectada a la radio. Dependiendo de la frecuencia a la que este la señal, puede haber mayor o menor perdida; siendo la frecuencia a la que opera la antena, donde haya menor pérdida. A pesar de las variaciones del cable, la antena tiene menor pérdida con señales de 2.4 GHz, siendo la frecuencia operada por los radios Xbee S2C.

La Figura 33 contiene una gráfica proveniente del proveedor enfocada en la eficiencia de la antena, siendo esta el porcentaje de potencia radiada con respecto a la subministrada. En este caso, el pico máximo de eficiencia (sin importar la longitud del cable) de la antena aparece en los 2.45GHz. Por último, la Figura 34 muestra la ganancia de la antena, la cual está concentrada también en la banda de los 2.4GHz [30].

Figura 32: Gráfica de RL de la antena FXP74, obtenida de las especificaciones de la antena, página 6[30]**.**

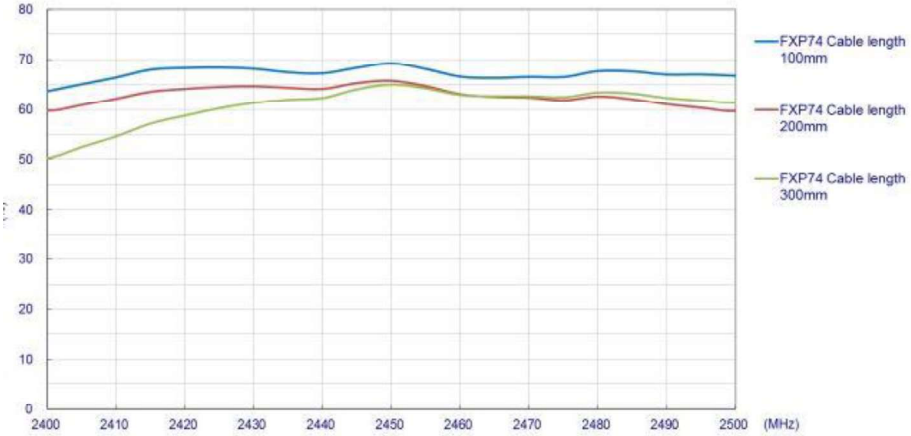

Figura 33: Gráfica de eficiencia de la antena FXP74, obtenida de las especificaciones de la antena, página 7[30]**.**

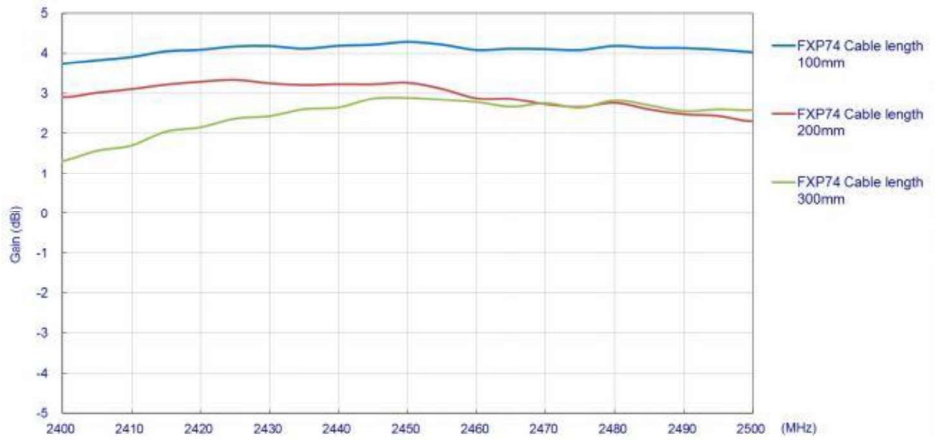

Figura 34: Gráfica de la ganancia de la antena FXP74, obtenida de las especificaciones de la antena, página 6[30]**.**

El proveedor ofrece más información, mostrando los patrones de radiación en todos los planos de la antena, lo cual permite un conocimiento de la radiación en la antena en las 3 dimensiones (Figura 35 para el plano XY, Figura 36 para el plano ZX y Figura 37 para el plano ZY). Resulta observable un patrón con que radia, horizontalmente, con casi la misma potencia en todas las direcciones y en planos verticales un poco más orientado hacia el frente de la antena [30].

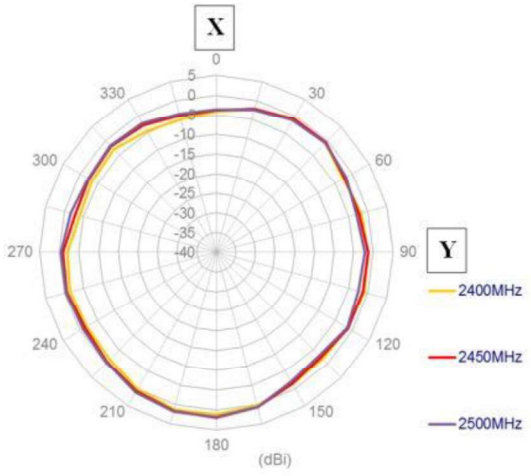

Figura 35: Plano XY del patrón de radiación de la antena FXP74 obtenida de las especificaciones de la antena, página 9[30]**.**

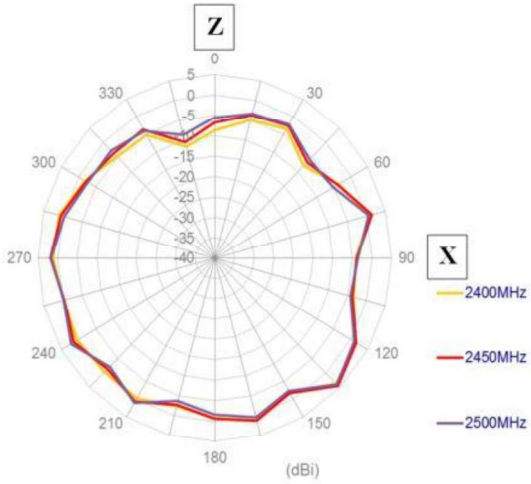

Figura 36: Plano ZX del patrón de radiación de la antena FXP74 obtenida de las especificaciones de la antena, página 9[30]**.**

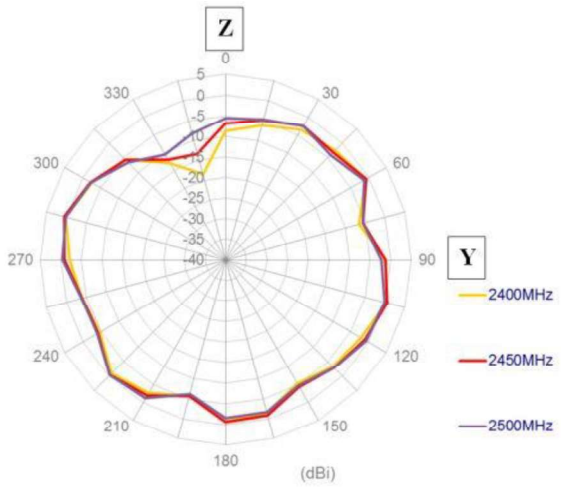

Figura 37: Plano ZY del patrón de radiación de la antena FXP74 obtenida de las especificaciones de la antena, página 9[30]**.**

3.2.4.2.2 915 MHz

El radio Xbee empleado para el protocolo DigiMesh, el Xbee S3B, cuenta también con un conector U.FL, este conector permite tener una gama de antenas que son "desmontables". Para dicho protocolo, las antenas de 950 MHz proporcionadas por la empresa Taoglass fueron las ocupadas igualmente. FXP.290 es el nombre que tiene la antena en uso (Figura 38), el proveedor brinda las especificaciones de relevancia mostradas en la Tabla 6.

Figura 38: Antena FXP.290, obtenida de las especificaciones de la antena, página 1[31]**.**

Tabla 6: Tabla de especificaciones de la antena FXP.290.

Especificación	Valores
Rango de frecuencia	902 MHz – 928MHz
Perdida de Retorno (RL)	-20 dB
Eficiencia	40%
Ganancia	1.5 dBi
Impedancia	50Ω
VSWR	≤ 2:1
Polarización	Lineal
Límite de potencia	5 W
Temperatura de operación/guardado	-40° C a 85° C
Dimensiones	75x45x0.1mm
Peso	1.5g
Conector	MHFII (compatible con U.FL)
Estándar del cable	Mini-Coaxial 1.13 mm
Adhesivo	3M 467

La gráfica de la Figura 39 muestra la pérdida de potencia a diferentes frecuencias. A 915 MHz, aproximadamente, aparece la menor pérdida (similar en potencia a la antena FXP74) y, ergo, la frecuencia de operación. La Figura 40 y la Figura 41 muestran la eficiencia y la ganancia de la antena a la frecuencia de operación, donde la antena revela una menor eficiencia y ganancia con respecto a la FXP74. El proveedor ofrece más información respecto a la antena, agregando una gráfica el voltaje de la onda estacionaria (Figura 42), ocasionada por la reflexión de ondas; y una Carta Smith (Figura 43) con la es posible conocer la impedancia de la antena a las ondas a través de diferentes frecuencias [31].

Figura 39: Gráfica de RL de la antena FXP.290 obtenida de las especificaciones de la antena, página 4[31]**.**

Figura 40: Gráfica de eficiencia de la antena FXP.290 obtenida de las especificaciones de la antena, página 5[31]**.**

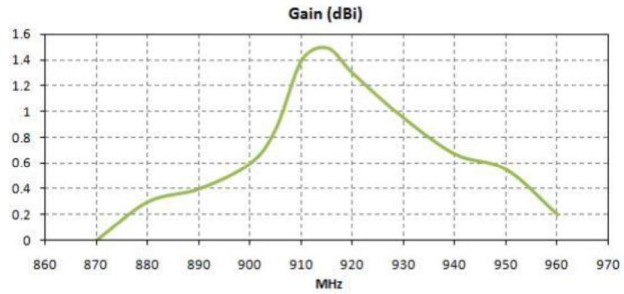

Figura 41: Gráfica de la ganancia de la antena FXP.290 obtenida de las especificaciones de la antena, página 6[31]**.**

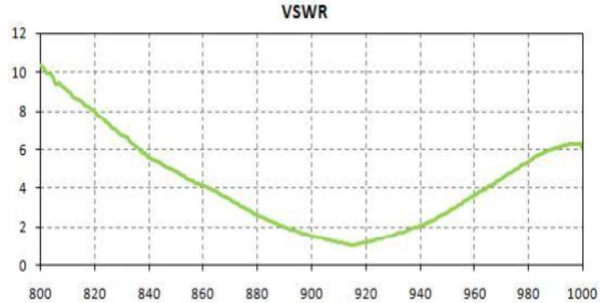

Figura 42: Gráfica de VSWR de la antena FXP.290 obtenida de las especificaciones de la antena, página 4[31]**.**

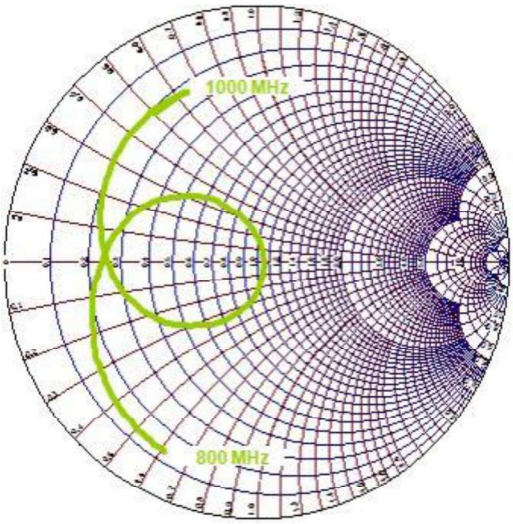

Figura 43: Carta de Smith de la antena FXP.290 obtenida de las especificaciones de la antena, página 5[31]**.**

Para esta antena, el proveedor muestra un patrón de radiación en modelos 3D, en la Figura 44, la Figura 45, la Figura 46 y la Figura 47. Dicho patrón muestra bastantes irregularidades, con una direccionalidad más marcada hacia la parte de enfrente de la antena, que es también donde reside la mayor potencia de radiación [31].

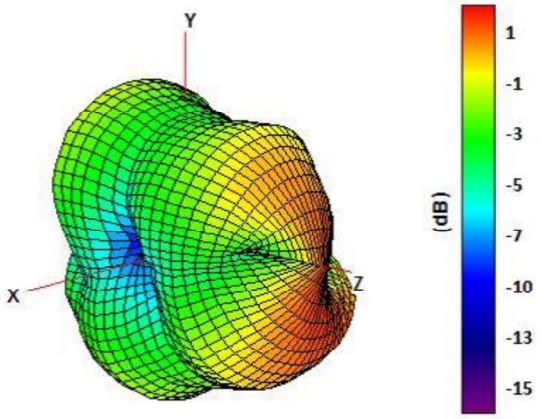

Figura 44: Modelo general del patrón de radiación de la antena FXP.290 obtenida de las especificaciones de la antena, página 6[31]**.**

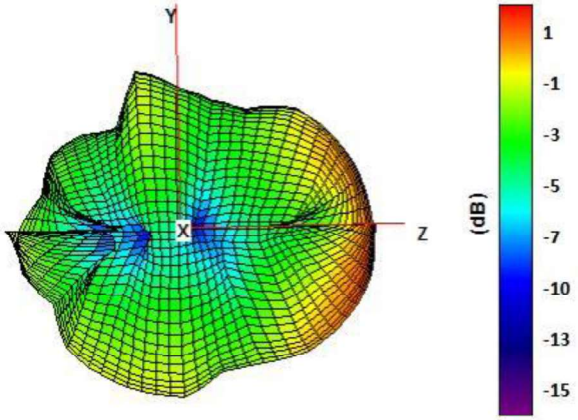

Figura 45: Plano YZ del patrón de radiación de la antena FXP.290 obtenida de las especificaciones de la antena, página 7[31]**.**

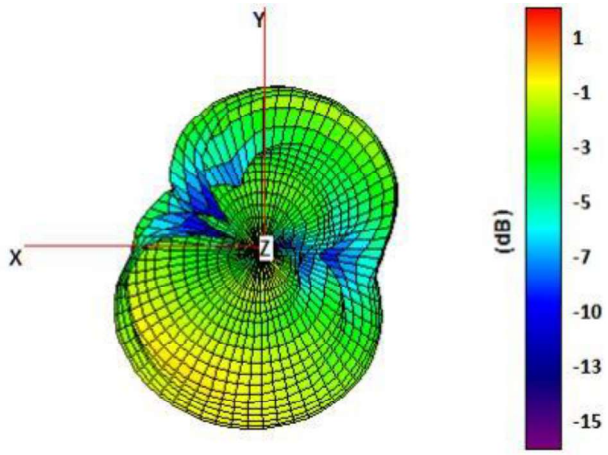

Figura 46: Plano XY del patrón de radiación de la antena FXP.290 obtenida de las especificaciones de la antena, página 7[31]**.**

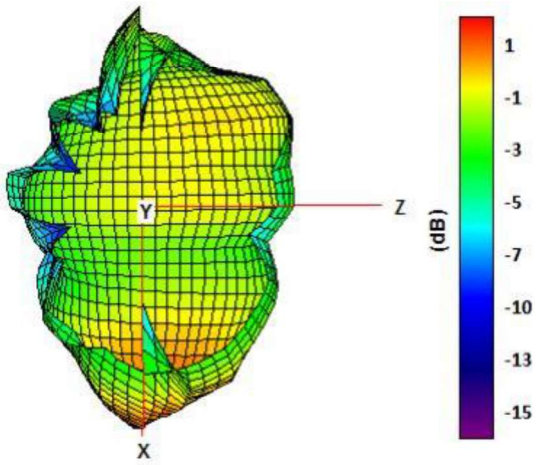

Figura 47: Plano ZX del patrón de radiación de la antena FXP.290 obtenida de las especificaciones de la antena, página 8[31]**.**

3.3. Software

3.3.1. XCTU

Los radios Xbees de la compañía Digi International solo pueden ser configurados mediante comandos AT. La configuración puede ser alámbrica o inalámbrica (es recomendable hacer la configuración alámbrica la primera vez). Para ello, la empresa Digi desarrolló la herramienta XCTU, mostrada en la Figura 48. Dicho *software* ofrece una visión "*Friendly*" de la configuración de radios. Este software puede ser instalado en sistemas operativos tales como Linux, Windows y MacOS.

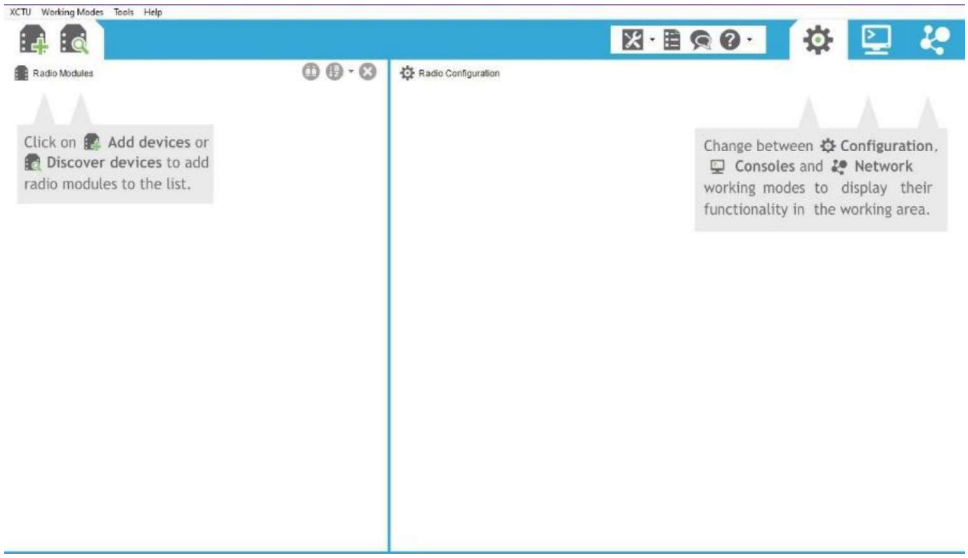

Figura 48: Captura de pantalla del *software* XCTU.

El *software* hace uso de los puertos seriales de la computadora o laptop, para acceder al radio mediante un módulo de conexión entre el radio y un conector *microUSB*, después puede ser conectado a la computadora por medio de un cable. Una vez conectado, el *software* pide parámetros al usuario como baudios y el puerto para identificar al radio y cargar la configuración actual de este. Estos pasos para la configuración de los radios son mostrados y descritos en la Figura 49, la Figura 50, la Figura 51 y la Figura 52.

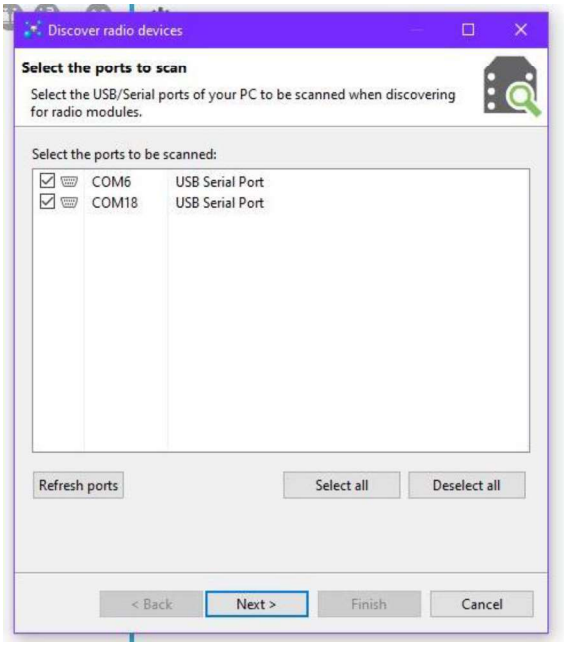

Figura 49: Paso 1. Seleccionar los puertos de la computadora.

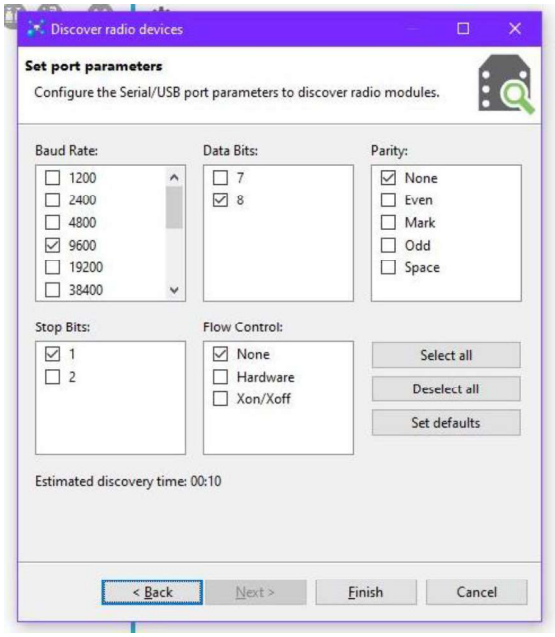

Figura 50: Paso 2. Seleccionar las características del radio.

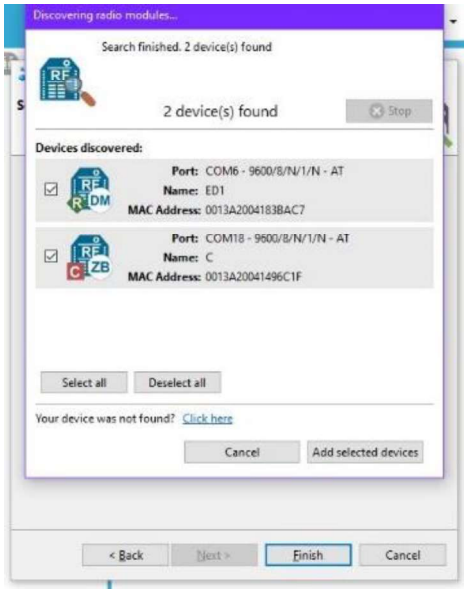

Figura 51: Paso 3. Seleccionar los Radios encontrados.

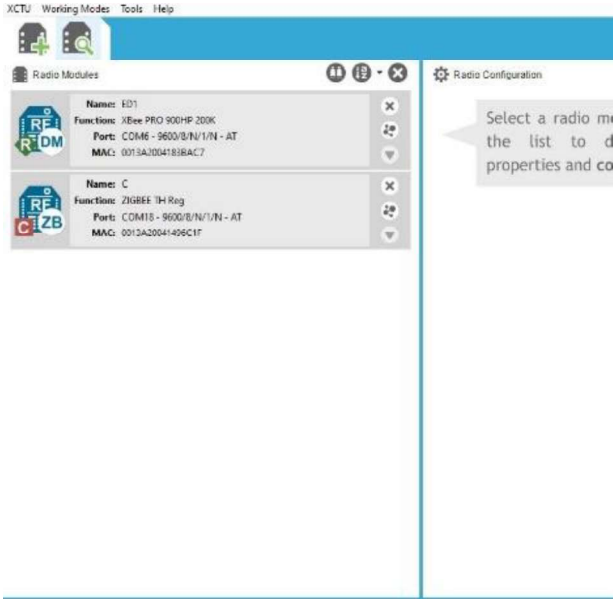

Figura 52: Paso 4. Agregarlos y seleccionar el radio para acceder a su configuración.

El *software* puede conectarse a la red para buscar actualizaciones tanto para el mismo *software* como en los *firmwares* de los diferentes radios, los cuales también puede actualizar. Aparte de lo mencionado, el *software* permite la configuración de los parámetros AT del radio, mismos que varían dependiendo el protocolo y, por tanto, del *firmware* (Figura 53).

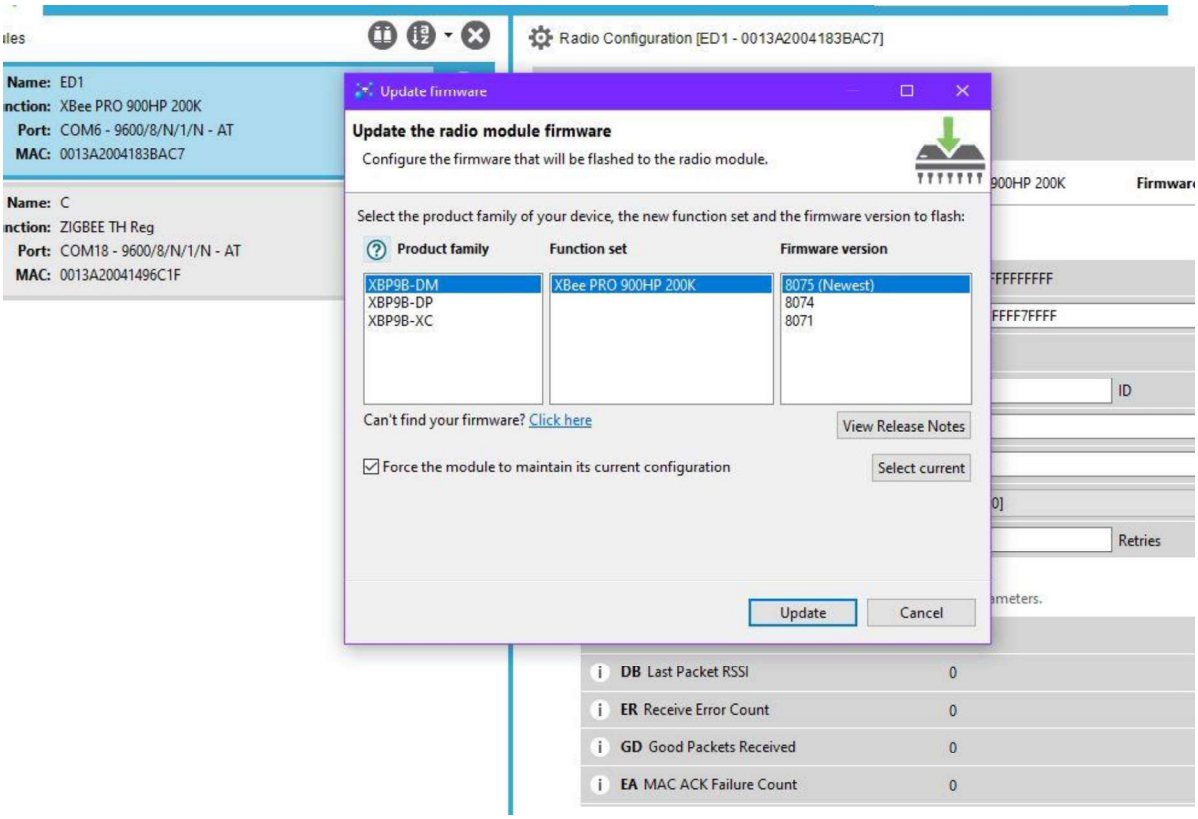

Figura 53: Actualización de *firmware* en XCTU.

De entre los parámetros configurables en los radios que son de importancia para este trabajo están:

- ID, PAN ID o Network ID - Identificación de la red.
- *Preamble* ID (Solo en DigiMesh) - Modulo para comunicarse que reduce la interferencia.
- PL, TX *Power Lever* - Nivel de potencia (valores de 1 - 4).
- PM, *Power mode* - Mejora la sensibilidad en 2 dB y la potencia de salida en 3dB.
- CE, *Coordinator Enabled*, *Routing Mode* - Implementa la jerarquía empleada en la red (C, R, E).
- DH, *Destination High* - Primera parte de la dirección destino.
- DL, *Destination Low* - Segunda parte de la dirección destino.

- NI, *Node Identifier* - Nombre particular del nodo.
- BD, *Baud Rate* - Taza de baudios (capacidad de escribir símbolos por segundo de forma analógica o digital).
- AP, *API Mode* - El modo API usado (valores de 0 - 2).
- SM, *Sleep Mode* - Tipo de sueño que pueda tener el radio (valores de 0 - 5).

De igual forma, el *software* provee herramientas para encriptación (con AES), un generador de *frames* para el caso de ocupar los radios en modo API (Interfaz de programación de aplicaciones), pero al no emplearse en el trabajo, los mensajes no estarán encriptados. XCTU cuenta también con herramientas para la conexión tales como: Generador e intérprete de *frames*, Recuperador de Xbees, Sesión de consola, *MicroPython*, Prueba de rango, Explorador de *Firmware*, Consola serial, Analizador de espectros y *Throughput*. El *software* cuenta con un apartado para probar una conexión entre radios con escritura en el teclado (Figura 54) o con un *frame* generado por el usuario (Figura 55) [32].

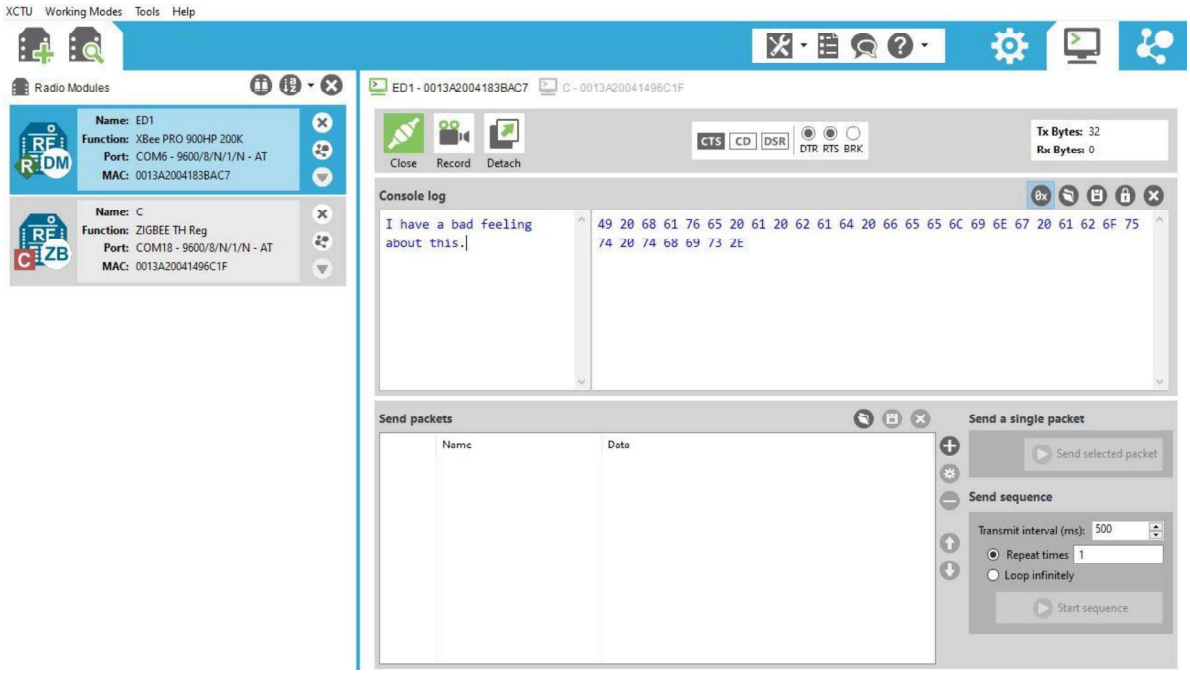

Figura 54: Texto escrito y enviado en XCTU.

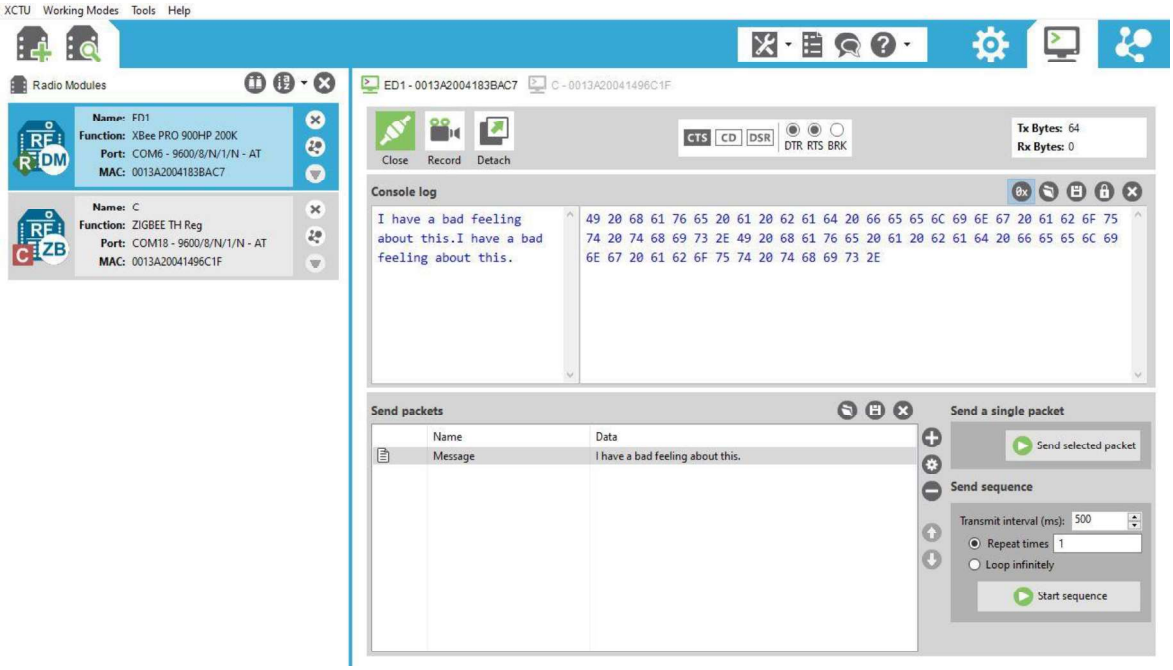

Figura 55: Frame generado y enviado en XCTU.

3.3.2. Arduino IDE

Para poder meter (también llamado como subir o cargar) algún programa a la tarjeta Arduino es necesario ocupar un IDE (Figura 56). En este programa es posible escribir desde un computador el código que se pretende emplear en la tarjeta, para después cargarlo a la misma. Con un puerto serial USB la computadora puede conectarse a la tarjeta y, posteriormente, subir el programa; siendo cuestión de seleccionar el puerto serial donde esté la tarjeta, verificar la sintaxis del código y cargar el programa para que la tarjeta empiece a operar bajo las órdenes de este. La Figura 57, la Figura 58 y la Figura 59 muestran los pasos a seguir del procedimiento descrito.

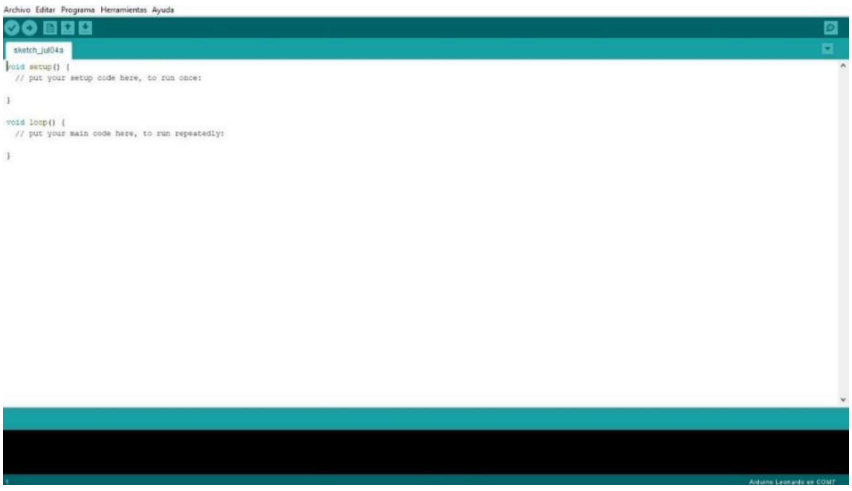

Figura 56. Captura de pantalla del Arduino IDE.

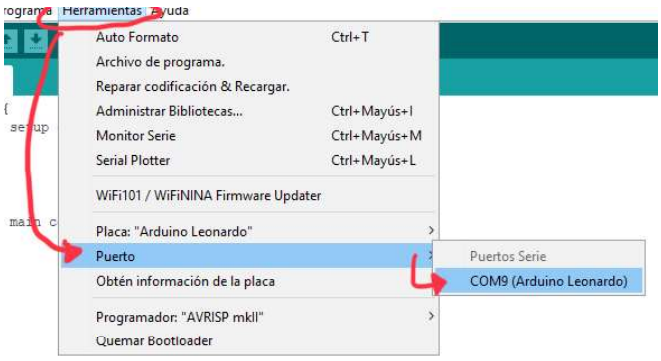

Figura 57: Paso 1. Identificación del puerto Serial de la tarjeta Arduino

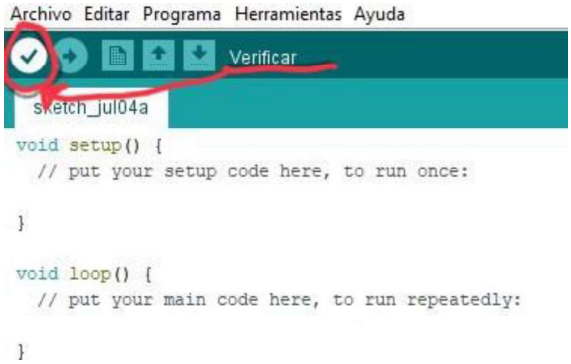

Figura 58: Paso 2. Verificación de código en el software de Arduino.

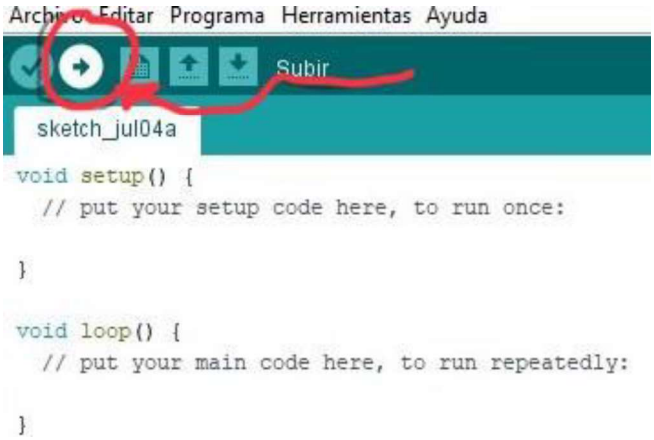

Figura 59: Paso 3. Carga de un programa a la tarjeta Arduino.

Los microcontroladores de las tarjetas Arduino leen el programa en 2 secciones: la primera, corresponde a la sección donde los comandos se ejecutan una única vez al arranque del Arduino, llamada *Set up*; y la segunda, conocida como *Loop*, donde los comandos son ejecutados repetidamente. Estas secciones acceden a la programación de Arduino, la cual está basada en el lenguaje C y C++.

El *software* es gratuito y puede ser instalado en sistemas operativos como Windows, Linux y MacOs. Durante la instalación, el programa agrega los *Drivers* para los puertos seriales, así como un número considerable de bibliotecas que pueden ser usadas en la programación de Arduino y que a su vez vienen, en su mayoría, con ejemplos del uso básico de las funciones dentro de las bibliotecas como del Arduino en general (Figura 60).

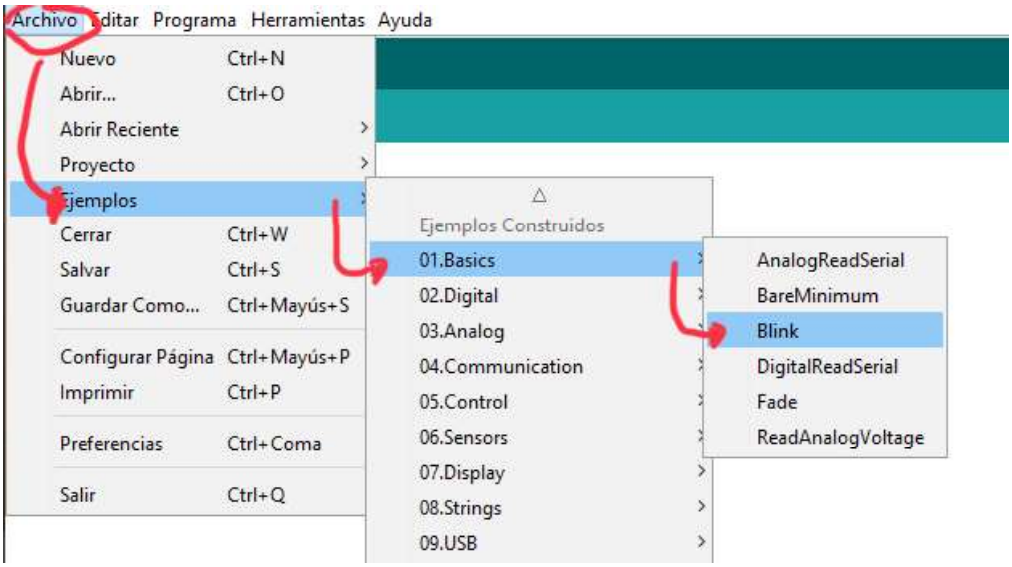

Figura 60: Ejemplos de Arduino IDE.

Arduino IDE es un código abierto, por lo que acepta bibliotecas realizadas por terceros en formato .zip y programado en lenguaje C++, o bien, suscribiéndose a Arduino es posible subir la biblioteca al administrador de bibliotecas, agregarla e instalarla desde ahí (Figura 61). Sin embargo, muchas de estas bibliotecas dependen del modelo de la tarjeta que se empleé por las características de cada tarjeta Arduino. Por lo que, si bien, Arduino IDE es compatible para programas con los diferentes tipos de tarjetas Arduino, no todas las librerías son compatibles con las mismas tarjetas.

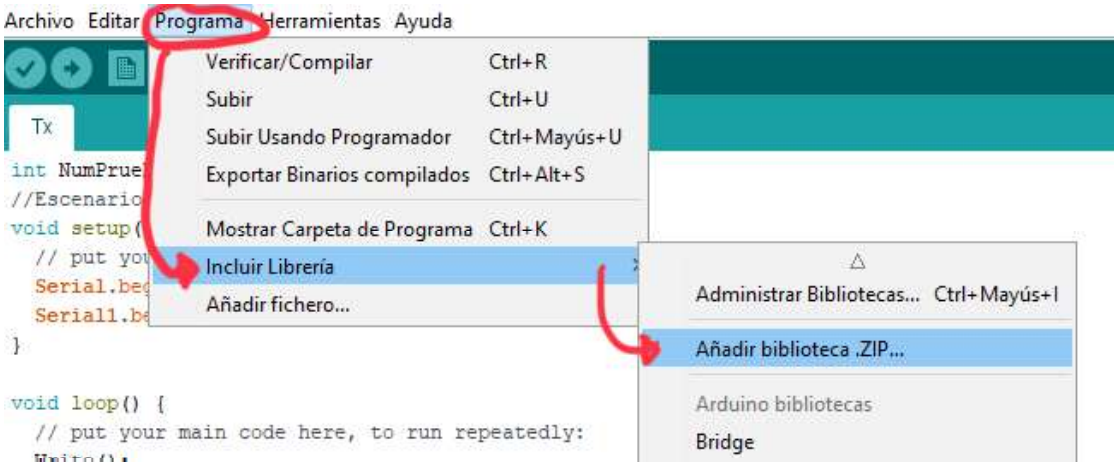

Figura 61: Administrador de Bibliotecas de Arduino.

El *software* a su vez cuenta con un monitor serial. Si una tarjeta Arduino está conectada al computador, la información de la tarjeta puede ser vista en la pantalla a través del monitor y tiene la opción de mostrar la hora registrada en el computador en relación con

la llegada de información por parte de la tarjeta. La Figura 62 muestra el botón del monitor serial, mientras que la Figura 63 muestra un ejemplo de las entradas de Arduino a la computadora con la hora registrada.

Figura 62: Botón para abrir el monitor serial de Arduino.

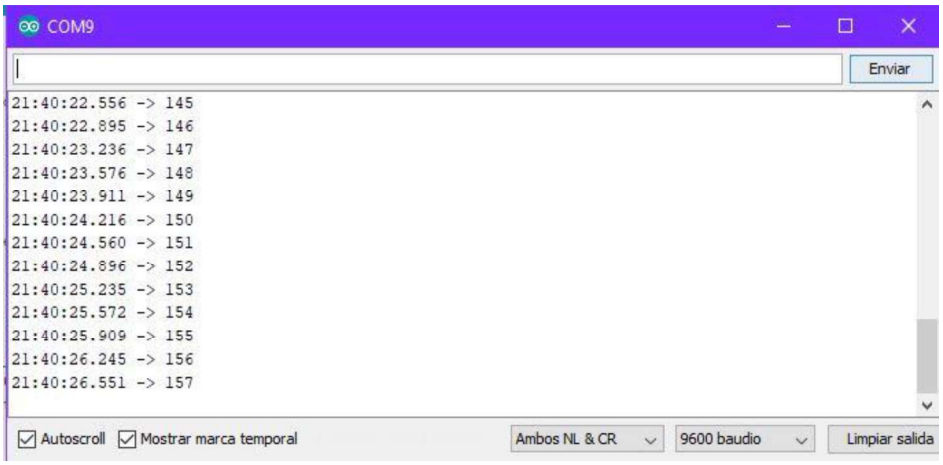

Figura 63: Monitor Serial de Arduino con algunas lecturas y con la opción del tiempo de llegada habilitada.

Capítulo 4. Desarrollo e implementación del sistema de comunicación

4.1. Introducción

La explicación de los procedimientos para la toma de mediciones, la configuración tanto en *software* como en *hardware* de los dispositivos usados y las consideraciones tomadas; determinan información de gran importancia al encontrar una mejor relación con los resultados del estudio. Misma razón por la que son detallados a lo largo de este capítulo.

4.2. Mensaje y configuración inicial de equipos

Para los diferentes escenarios, los Arduinos tienen diferentes programas, los cuales están ubicados en el apéndice C de este trabajo. Dichos programas tienen la finalidad de mandar un mensaje único e identificarlo, repitiendo este proceso hasta completar un paquete de 200 mensajes. Posteriormente, el programa suspende la comunicación por 2 segundos para después reiniciar la tarea.

El Arduino transmite a través de los puertos seriales de *byte* en *byte*, por lo que los mensajes constan de 2 *bytes*. El primero, como un *header* para identificar la procedencia del mensaje y, el segundo, como un contador que va desde el 0 hasta el 199 (un mensaje único para los 200 mensajes). De esta forma, el segundo *byte* puede tomar valores desde el 0 hasta el C7 (199 en hexadecimal); por parte de los *headers*, estos siguen el orden mostrado en la Tabla 7.

Tabla 7: Tabla de *Headers* usados.

Header	Valor en Hex	Uso
E2R[1] (End device a Router)	FD	1. RP
R2C[1] (*Router* a Coordinador)	FC	1. RP
E12C(*End device* 1 a Coordinador)	FB	1. P2M
E22C(*End device* 2 a Coordinador)	FA	1. P2M
E32C(*End device* 3 a Coordinador)	F9	1. P2M
E2C(*End device* a Coordinador)	F8	1. P2P 2. RP 3. NM

Los radios cuentan con una amplia gama de valores que pueden ser configurados para tener redes más personalizadas, sin embargo, para fines de las pruebas, los radios se mantuvieron con valores por defecto salvo los mostrados en la Tabla 8.

Tabla 8. Configuraciones de los radios.

Parámetro[2]	Valor en ZB	Valor en DM
ID	123	2322
Preamble ID	NA	7
CE	*Enable*[1] (C), *Disable*[0] (R y ED's)	Ind. Msg. Coordinator[1] (C), *Std. Router*[0] (R y ED's)
DH	0 (C), 13A200 (R y ED'S)	0 (C), 13A200 (R y ED'S)
DL	FFFF (C), 41496C1F (R y ED'S)}	FFFF(C), 4183BA39 (R y ED'S)
NI	C, R, ED, ED2, ED3	C, R, ED, ED2, ED3
PL	Highest[4]	Highest[4]
PM	Enable[1]	NA
BD	9600[3]	9600[3]
AP	Transparent mode[0]	Transparent mode[0]
SM	Router[0]	Normal[0]

4.3. Escenarios

El estudio, en palabras simples, ocupa los radios Xbee con los diferentes protocolos bajo circunstancias similares, para observar el desempeño de estos ante unas condiciones rurales, donde el terreno es irregular. Con elementos que pueden ser significativos durante la transmisión de información, tales como: pasto, árboles, rocas, colinas y hasta basura.

Las pruebas responden a mediciones simplificadas de las condiciones usuales esperadas que puedan llegar a experimentar las redes inalámbricas IoT, avanzando en la dificultad y en la integración de más de 2 dispositivos. El trabajo tiene como alcance el representar las situaciones que puedan ocurrir durante la implementación de estas redes. A

1 Posible uso, debido a que su header no tiene relevancia para la presentación de resultados.

2 Revisar sección 2.2.1 de este trabajo.

continuación, los escenarios considerados son mostrados junto con aspectos de interés de cada medición:

1. RSSI(Indicador de intensidad de señal recibida):
   - Puntos de medición - cada 10 metros.
   - Elevación por punto de medición - 0,1,2 y 3 metros.
   - Interés - Intensidad de señal y paquetes transmitidos.
2. Punto a punto(P2P):
   - Puntos de medición - cada 10 metros (aprox).
   - Elevación por punto de medición - 0,1,2 y 3 metros.
   - Interés - Paquetes transmitidos, retraso y procesamiento.
3. Salto/Retransmisión (RP):
   - Puntos de medición - 3 mediciones de cada 10 metros antes de la máxima distancia aceptable (MDA) en espejo con un punto medio y una en espejo con la MDA a nivel del suelo.
   - Elevación por punto de medición - 1,2 y 3 metros y una a nivel del suelo.
   - Interés - Paquetes transmitidos, retraso y procesamiento.
4. Punto a multipunto (P2M):
   - Puntos de medición - 3 mediciones de cada 10 metros antes de la MDA y con la MDA a nivel del suelo.
   - Elevación por punto de medición - 1,2 y 3 metros y una a nivel del suelo.
   - Interés - Paquetes transmitidos, retraso y procesamiento.
5. Nodo Movil(NM):
   - Puntos de medición - 3 mediciones de cada 10 metros antes de la MDA y 3 con la MDA a nivel del suelo. Todas las mediciones con el transmisor en movimiento.
   - Elevación por punto de medición - 1,2 y 3 metros y una a nivel del suelo.
   - Interés - Paquetes transmitidos, procesamiento y velocidad constante del transmisor.

4.4. Desarrollo de los escenarios

4.4.1. RSSI

Este escenario ocupa los radios Xbee S2C y S3B junto con el *software* XCTU, el cual cuenta con la herramienta: "*Range Test*". La herramienta es empleada para transmitir 100 paquetes entre 2 radios del mismo protocolo, en la misma red y en el mismo canal; y obtener un PER (Taza de error de paquetes) junto con el RSSI de cada paquete recibido. La herramienta entonces introduce esta señal en una gráfica RSSI vs tiempo y, además, obtiene el PER en porcentaje con los paquetes entrantes.

La distancia entra como un eje con valores de 10 en 10 metros. Estos fueron medidos con la aplicación "*Spyglass*" en la posición donde eran tomadas las mediciones. A su vez, la posición era medida con un flexo-metro y colocando banderas discretas (como basura especifica o rocas designadas), para evitar llamar la atención durante los días de medición. Con Google Earth, la precisión de la ubicación de cada uno de los 10 metros de distancia fue aumentada.

En cada punto de 10 metros de medición las elevaciones de 0, 1, 2 y 3 metros serían incluidas para analizar el efecto que puede tener el terreno en la señal, la Figura 64 muestra un dibujo de las mediciones con respecto al escenario.

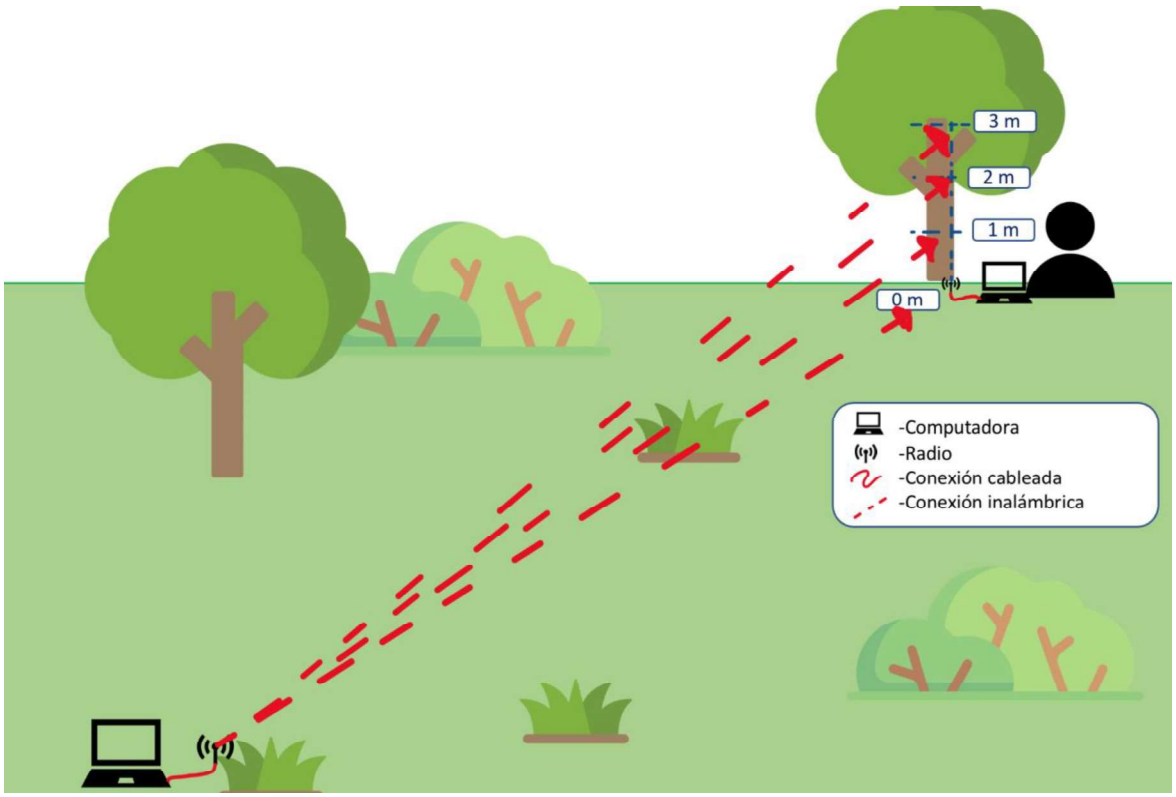

Figura 64: Dibujo alusivo a la toma de mediciones durante el escenario RSSI.

4.4.2. Punto a Punto (P2P)

Ocupando como apoyo los puntos del escenario anterior, dado que las mediciones de punto a punto son similares a las del escenario RSSI, este escenario busca encontrar la distancia máxima de para la transmisión de información. Ahora, los Arduinos son incorporados y el monitor serial muestra las lecturas en el *software* Arduino IDE.

Los Arduinos están programados con los programas del apéndice C (sección de Arduino para los escenarios P2P, RP y NM). El transmisor manda 200 paquetes con una diferencia de tiempo de 300 ms y posterior a 2 segundos de espera, repite la acción. El receptor espera al mensaje y en cuanto detecte un mensaje con valor menor que el anterior recibido, reinicia su cuenta de paquetes recibidos. La Figura 65 muestra un dibujo de lo que se pretende en las mediciones de este escenario.

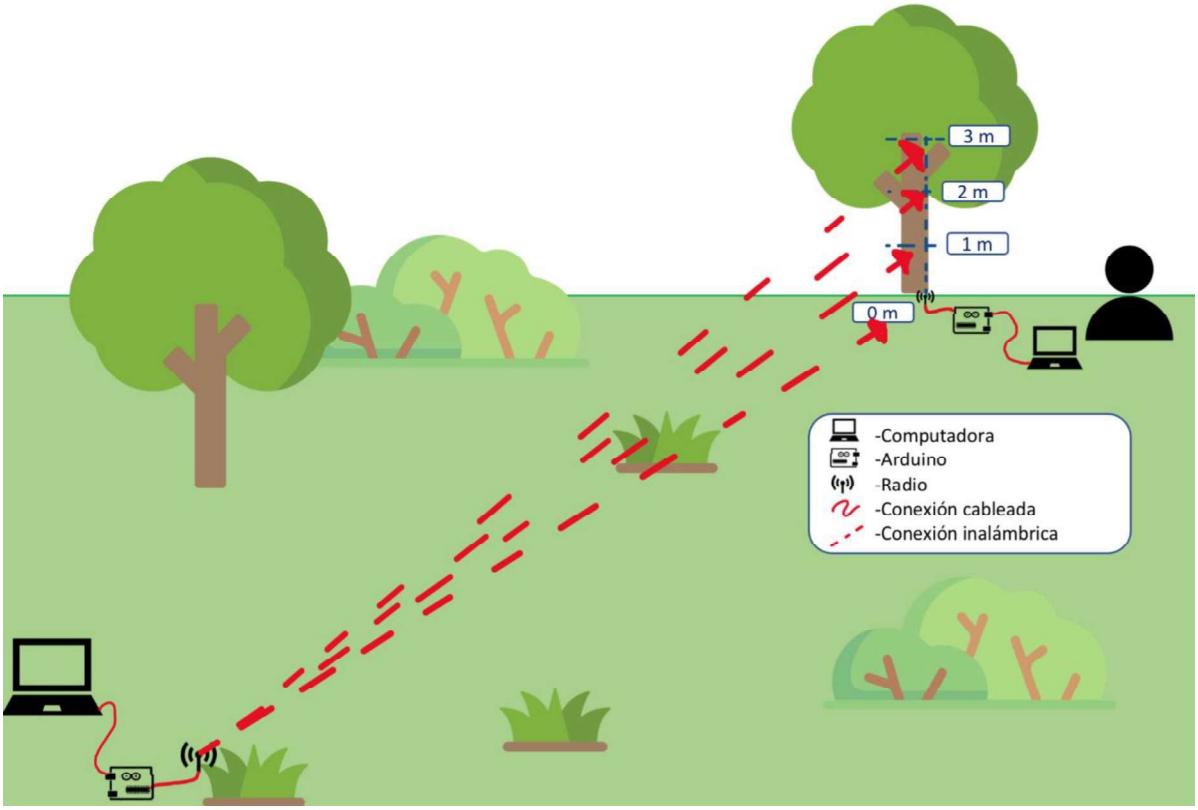

Figura 65: Dibujo alusivo a la toma de mediciones durante el escenario P2P.

4.4.3. Salto (RP)

Para la realización de este escenario, fue empleado un método similar a los escenarios RSSI y RP para medir el número de paquetes, pero agregando un radio extra para retransmitir la información recibida y aumentar el rango de comunicación.

La Figura 66 muestra la posición de los radios transmisor y receptor junto con el radio retransmisor (*router* o repetidor), con elevaciones de 1 a 3 metros en el repetidor. Otra medición será agregada con la elevación a nivel de piso, esto debido a la diferencia de rango de alcance a nivel del suelo con respecto a los demás niveles. Finalmente, los radios fueron configurados para que solo haya conexión con el coordinador (receptor) y el radio retransmisor solo sirvió para redireccionar los mensajes, sin procesar información alguna.

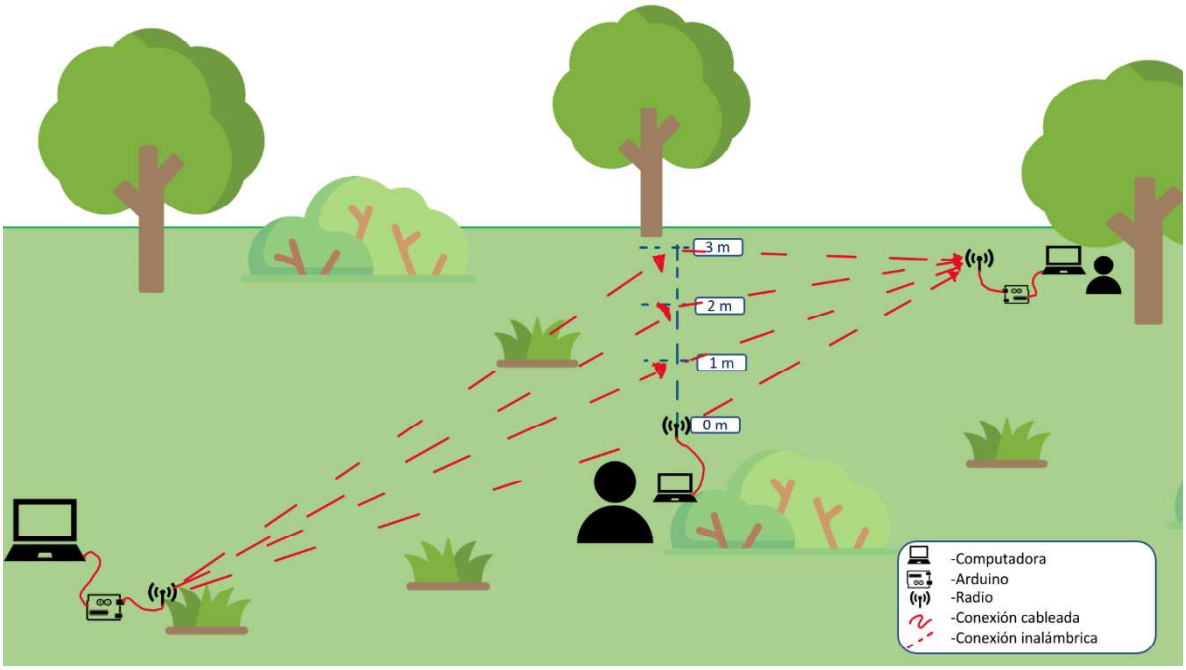

Figura 66: Dibujo alusivo a la toma de mediciones durante el escenario RP.

4.4.4. Punto a Multipunto (P2M)

Haciendo uso de 4 dispositivos: uno como coordinador y los otros 3 como *routers* (en el caso de Zigbee, *end device* en caso de los DigiMesh). Los *routers*, posicionados a la máxima distancia aceptable del coordinador, procuran evitar la conexión entre los *routers* y buscan bombardear al emisor/coordinador con mensajes a 300 ms por dispositivo, los programas cargados en Arduino están en el apéndice C (sección de Arduino para el escenario P2M). La Figura 67 muestra un dibujo aproximado del desarrollo de las mediciones bajo este escenario.

El escenario donde un transmisor tiene que dar información a 3 receptores a la par es poco probable y, por lo mismo, las mediciones de ese escenario no siguen el propósito de este trabajo.

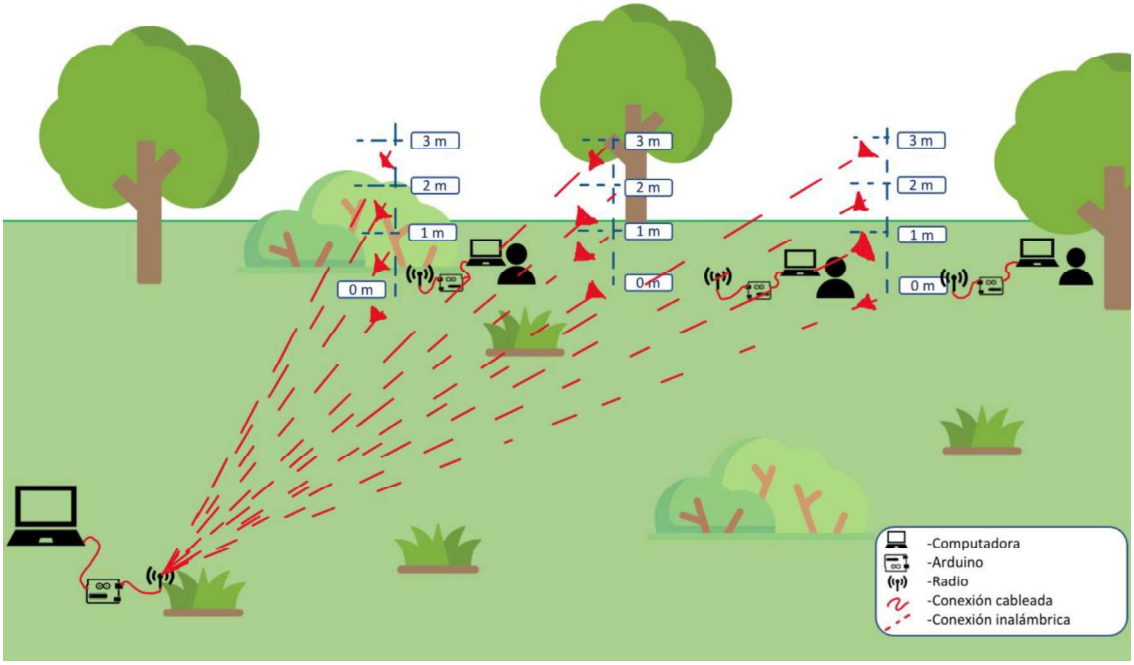

Figura 67: Dibujo alusivo a la toma de mediciones durante el escenario P2M.

4.4.5. Nodo Móvil (NM)

Con la última medición aceptable de P2P, las mediciones asimilan las tomadas durante los escenarios P2P y RP, teniendo 3 puntos de medición para las elevaciones 1, 2 y 3 y otros 3 puntos para la elevación 0.

Considerando un eje imaginario entre el transmisor y el receptor, las mediciones son realizadas con el transmisor en movimiento, trazando líneas imaginarias de 32 metros (aproximadamente) perpendiculares al eje anteriormente mencionado a una velocidad constante de 5.85 km/h con dificultad de mantener una velocidad constante mayor. Este escenario comparte los programas usados para los escenarios P2P y RP (Dibujo mostrado en la Figura 68).

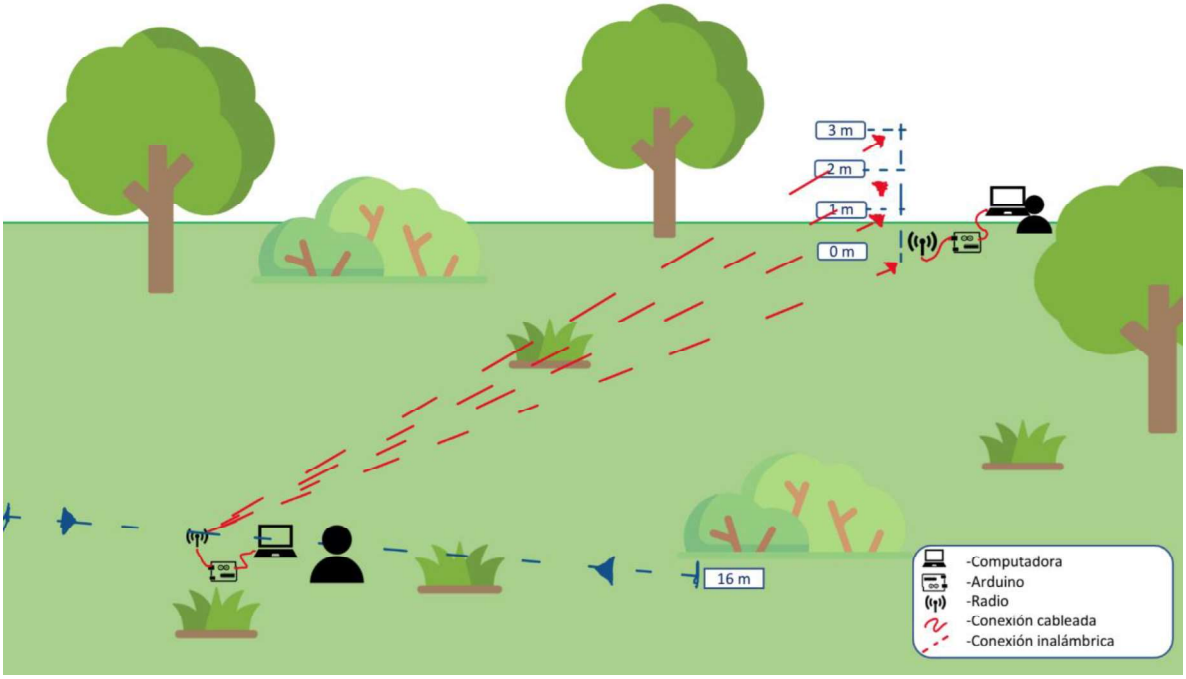

Figura 68: Dibujo alusivo a la toma de mediciones durante el escenario NM.

4.5. Ubicación de la zona

La Ubicación seleccionada consiste en una serie de terrenos cerca de la carretera a San Nicolás Tlamínca, en el municipio de Texcoco, Estado de México. A pesar de que los terrenos son del municipio o de particulares, las mediciones pueden llevarse con problemas moderados, dando un aproximado de 270 metros de terrenos irregulares con los elementos requeridos, mencionados anteriormente.

La Figura 69 y la Figura 70 muestran 2 fotos tomadas de la zona, mientras que la Figura 71 muestra la ubicación satelital empleadas en el momento en el que las mediciones fueron tomadas para los escenarios RSSI, P2P, RP y NM.

Figura 69: Fotografía 1.

Figura 70: Fotografía 2.

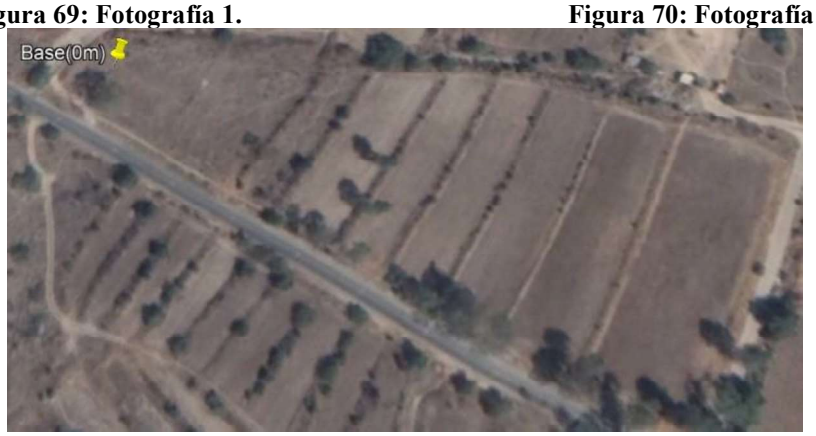

Figura 71: Imagen satelital de la Zona.

Las mediciones para el escenario P2M fueron tomadas 2 meses posteriores (por temporada de lluvias) lo que ocasionó que el lugar tuviera mayor humedad y vegetación, la Figura 72 y la Figura 73 son imágenes de la ubicación durante los 2 últimos escenarios mencionados.

Figura 72: Fotografía de la Zona tras las lluvias.

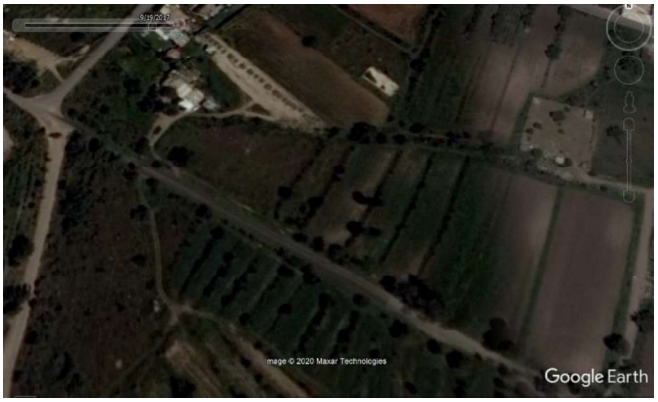

Figura 73: Imagen satelital tomada en el 2017 y que asemeja al estado del terreno después de la temporada de lluvias.

Capítulo 5. Resultados de los escenarios RSSI y P2P

5.1. Introducción

Tras realizar las mediciones, los resultados de cada escenario reciben un procesamiento con el programa del apéndice C: “Código en C++ para el procesamiento de información”. El programa retorna los elementos más relevantes durante la toma de mediciones en un formato *.csv* que pasa a formar tablas y gráficas en Excel. Este procesamiento, permite entender con mayor precisión lo que ha ocurrido durante la comunicación de estos dispositivos.

Este capítulo presenta con mayor detalle la forma de procesar datos, así como algunas consideraciones tomadas durante las mediciones. Agregando los resultados de los escenarios RSSI y P2P, respectivamente.

El formato de presentación de los resultados comprende de evidenciar los puntos de medición, seguido de una tabla con los valores numéricos y gráficas que ayudan a la comprensión de sus respectivas tablas. Al final, las gráficas contienen el análisis de lo que cada una indica.

5.2. Procesamiento de datos

La herramienta XCTU no puede exportar la información concisa a algún otro *software*, por lo que las capturas de pantalla fueron la única solución para rescatar la información (Figura 74). Con el uso del ojo humano la información fue traspasada a una tabla en Excel para obtener gráficas RSSI vs distancia en metros.

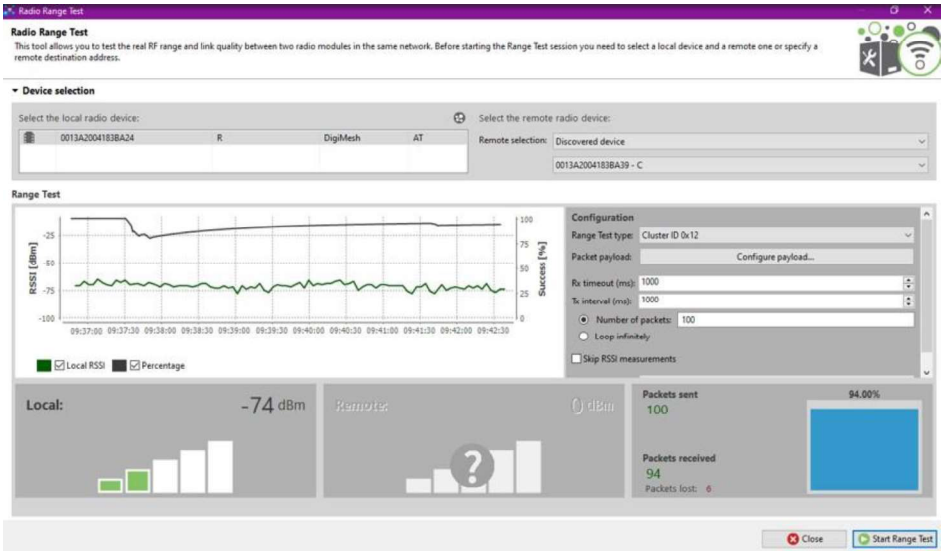

Figura 74: Ejemplo de resultados con la herramienta de XCTU para RSSI.

El monitor serial del *software* Arduino IDE tiene activado el acceso a la hora del sistema, por lo que muestra la hora del sistema en el que llega cada paquete en el formato: *horas:minutos:segundos.milisegundos* (Figura 75 y Figura 76). En el primer ejemplo, usado para los escenarios P2P, RP y NM, muestra la posibilidad de 3 tipos de entradas en el monitor.

La primera (y más abundante) muestra la entrada típica de un paquete con la hora de llegada en el formato mencionado, seguido de una flecha y el contador de paquetes del receptor o transmisor. La segunda muestra la hora de llegada, pero tras la flecha aparece un "NP" que significa el número de prueba, donde cada prueba acaba tras haber mandado los 200 paquetes. Finalmente, hay un "*head*" que consta de 2 números (0 y 1 en el ejemplo mencionado), esta entrada es añadida manualmente y sirve para reconocer que la distancia y la altura de la medición tomada respectivamente.

```
10:03:55.980 -> 192
10:03:56.317 -> 193
10:03:56.622 -> 194
10:03:56.957 -> 195
10:03:57.296 -> 196
10:03:57.634 -> 197
10:03:57.971 -> 198
10:04:00.300 -> 199
0 1
10:04:00.300 -> NP: 3
10:04:00.640 -> 0
10:04:00.976 -> 1
10:04:01.311 -> 2
10:04:01.648 -> 3
10:04:01.950 -> 4
10:04:02.286 -> 5
10:04:02.627 -> 6
10:04:02.968 -> 7
10:04:03.306 -> 8
10:04:03.644 -> 9
```

Figura 75: Ejemplo de resultados con el monitor serial para P2P, RP y NM.

```
10:27:57.315 -> ED2:188
10:27:57.588 -> ED3:136
10:27:57.621 -> ED2:189
10:27:57.860 -> ED3:137
10:27:57.927 -> ED2:190
10:27:58.167 -> ED3:138
10:27:58.235 -> ED2:191
10:27:58.474 -> ED3:139
10:27:58.508 -> ED2:192
10:27:58.745 -> ED1:193
10:27:58.745 -> NP1: 1
10:27:58.778 -> ED3:140
10:27:58.812 -> ED2:193
10:27:59.048 -> ED1:0
10:27:59.083 -> ED3:141
10:27:59.116 -> ED2:194
10:27:59.354 -> ED1:1
10:27:59.388 -> ED3:142
10:27:59.422 -> ED2:195
10:27:59.627 -> ED1:2
```

Figura 76: Ejemplo de resultados con el monitor serial para P2M.

Para el segundo ejemplo, el cual son las entradas en la recepción del escenario P2M, sigue la misma métrica en las posibles entradas del ejemplo anterior. La diferencia reside en que cada paquete esta diferenciado con un “ED” seguido del número del transmisor del cual proviene la información, al igual que la separación de la prueba.

Esta información, con las computadoras sincronizadas, otorga 2 documentos con formato ".txt" para los escenarios P2P y RP, 4 para el escenario P2M (solo 3 fueron recuperados) y un único documento para el escenario NM. Los documentos son introducidos al programa mostrado en el apéndice C, mismo que separa y organiza toda la información y cuya salida es un archivo en formato CSV que es exportado a Excel. Finalmente, Excel transforma la información a gráficas que describan los sucesos con mayor facilidad, con el fin de analizar de forma sencilla lo ocurrido durante las mediciones de los 2 protocolos.

5.2.1. Eventos Extra

Durante los resultados, los problemas de conexión son notorios, los que conllevan a los siguientes eventos extras:

- Falta de contador en los paquetes: Los programas de Arduino pueden mostrar que paquete está arribando en el radio, sin embargo, entre más elaborado resulta el código, más existe una descompensación al inicio de los paquetes. El resultado es una visualización de entradas como la mostrada en la Figura 77, la secuencia es complicada de predecir y alteraba las mediciones de una forma significativa. Arduino no manda más paquetes de los que debe, por lo que colocar un contador para que muestre cuantos paquetes llegan, en vez mostrar cada paquete en individual resolvió el problema y simplificó en gran medida las mediciones.

```
.9:10:38.678 -> 51
.9:10:38.678 -> NP: 1
.9:10:38.678 -> 13
.9:10:38.678 -> NP: 2
.9:10:38.678 -> 10
.9:10:38.678 -> NP: 3
.9:10:38.717 -> 52
.9:10:38.717 -> NP: 4
.9:10:38.717 -> 13
.9:10:38.717 -> NP: 5
.9:10:38.717 -> 10
.9:10:38.717 -> NP: 6
.9:10:38.717 -> 53
.9:10:38.717 -> NP: 7
.9:10:38.717 -> 13
```

Figura 77: Lectura de paquetes de Arduino sin contador.

- *Header* erróneo: El código distingue de donde proviene cada paquete, pero en caso de que el *header* no sea reconocido; éste manda un mensaje de error, la entrada entonces es similar a lo observado en la Figura 78.

```
10:13:53.037 -> ED2:57
10:13:53.037 -> ED2:58
10:13:53.037 -> ED2:59
10:13:53.037 -> ED2:60
10:13:53.037 -> ED2:61
10:13:53.084 -> errorerrorerrorerrorerrorerrorED2:62
10:13:53.084 -> ED2:63
10:13:53.084 -> ED2:64
10:13:53.084 -> ED2:65
10:13:53.084 -> ED2:66
10:13:53.084 -> ED2:67
10:13:53.084 -> ED2:68
10:13:53.084 -> ED2:69
10:13:53.084 -> ED2:70
10:13:53.084 -> ED2:71
10:13:53.084 -> ED2:72
10:13:53.084 -> ED2:73
```

Figura 78: Mensaje de error por paquetes con *header* erróneo.

- Perdida de conexión: Debido a diversos eventos, la conexión entre los radios puede cortarse, lo que resulta en entradas similares a las observadas en la Figura 79.

```
10:20:54.599 -> 197
10:20:56.912 -> 198
10:20:56.912 -> NP: 9
10:20:57.193 -> 0
10:20:57.521 -> 1
10:20:57.521 -> NP: 10
10:20:57.850 -> 0
10:20:58.131 -> 1
10:20:58.381 -> 2
10:20:58.709 -> 3
10:20:59.037 -> 4
10:20:59.037 -> NP: 11
10:20:59.366 -> 0
10:20:59.882 -> 1
```

Figura 79: Pérdida de paquetes por inestabilidad en la conexión.

- *Lag*: Un evento que se relaciona más con las limitantes del Arduino ocurre con el efecto "*lag*". Este ocurre cuando los paquetes son recibidos, pero el procesamiento es atrasado y al final todos los paquetes son mostrados con intervalos de tiempo no estipulados en el programa de Arduino. La entrada resulta similar a la mostrada en la Figura 80.

```
12:46:04.973 -> 192
12:46:05.427 -> 193
12:46:05.659 -> 194
12:46:13.095 -> 195
12:46:13.142 -> 196
12:46:13.142 -> 197
12:46:13.142 -> 198
12:46:13.142 -> NP: 7
12:46:13.195 -> 0
12:46:13.195 -> 1
12:46:13.195 -> 2
12:46:13.195 -> 3
12:46:13.242 -> 4
12:46:13.242 -> 5
12:46:13.295 -> 6
12:46:13.295 -> 7
12:46:13.295 -> 8
12:46:13.295 -> 9
12:46:13.342 -> 10
12:46:13.342 -> 11
12:46:13.627 -> 12
12:46:13.944 -> 13
12:46:14.299 -> 14
12:46:14.616 -> 15
```

Figura 80: Efecto *lag* durante las mediciones.

5.2.2. Consideraciones

1. El modo API de los radios XBees impide que el Arduino obtenga información del radio a través de sus pines. Con una biblioteca de Arduino es posible acceder a los datos, dicha biblioteca fue descartada debido a su exclusiva funcionalidad con el radio X2C bajo Zigbee, alterando la homogeneidad de los experimentos.

   A medida que el código en C de Arduino aumenta de complejidad, la transmisión no manda los datos en orden (Figura 77), por lo que el programa receptor no muestra el paquete entrante, si no que muestra el número de paquetes que ha recibido durante la prueba. Misma que cambia una vez que reciba un paquete 10 veces más pequeño que el paquete anterior recibido, debido a esto, el número de paquete en el receptor no forzosamente es el mismo que el transmisor ha mandado, sin embargo, esto propone una solución mejor que el problema supone.

2. Pruebas donde el número de paquetes sean menores a 50 serán descartados de las mediciones por no ofrecer suficiente información del desempeño del radio.

3. Las distancias son consideradas si la transmisión puede mandar más de 50 paquetes y/o el tiempo de transmisión no supera los 5 minutos sin recibir paquetes de forma continua.

4. Para la sincronización de datos TX y RX, el documento del receptor tiene prioridad, donde es registrada la primera prueba tomada en cuenta, cada prueba consta de un aproximado de 200 paquetes o 62 segundos. Posteriormente, 2 a 4 pruebas son tomadas (dependiendo del escenario) y 2 pruebas son ocupadas de guarda para continuar con la siguiente medición, ya sea en elevación o en distancia. Mediciones que duren más de las 4 pruebas por errores son consideradas con el respectivo tiempo extra, a menos que se descarten por el punto anterior.

   El documento de transmisión no tomará este tiempo extra, y se considerará como un tiempo de retraso mayor por fallos en transmisión, debido a la ambigüedad de los paquetes recibidos.

5. Los programas están hechos para detectar *headers* incorrectos con el mensaje: "Error" (Figura 78). Esto debido a que recibir ese mensaje involucra un error en la comunicación y el mensaje es descartado durante el procesamiento de información.

6. Debido al poco tiempo que requiere desplazarse en distancia, la segunda prueba de guarda puede ser considerada dentro de las mediciones para la elevación siguiente, pero esta consideración solo es empleada si el tiempo de *retraso* es mayor a los 62 segundos, pues simboliza que los paquetes en la recepción son, en su mayoría, pertenecientes a la prueba de transmisión anterior y cumplen con el propósito de esta.

5.3. RSSI

5.3.1. Puntos de medición

Los puntos correspondientes a este escenario fueron distanciados de 10 en 10 metros partiendo de la base y buscando no favorecer las conexiones con puntos altos, a su vez procurando una línea recta hasta que los protocolos dejaron de recibir mensajes de forma exitosa (bajo las condiciones de las consideraciones expuestas anteriormente). La Figura 81 y la Figura 82 muestran los puntos obtenidos para los 2 protocolos.

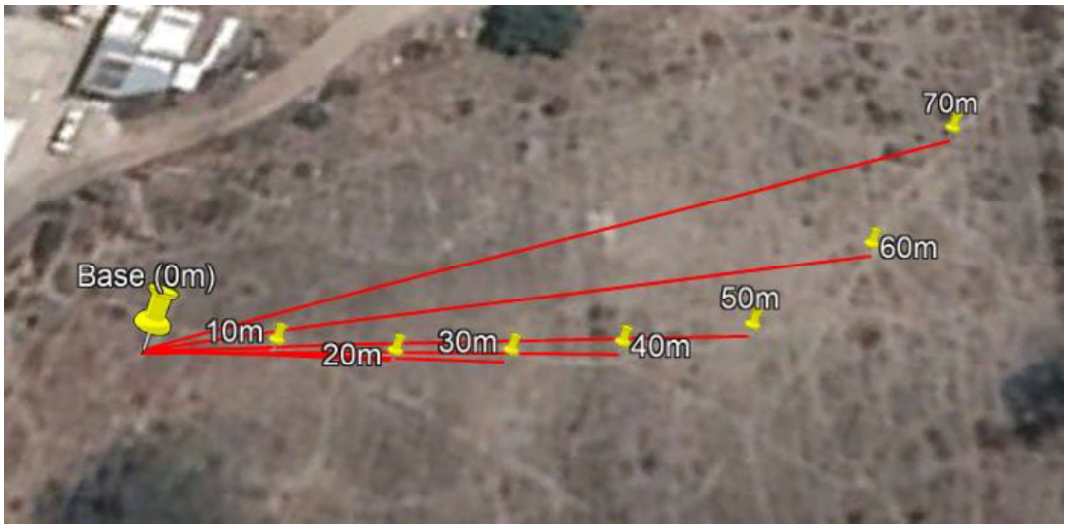

Figura 81: Puntos de mediciones usadas para Zigbee en RSSI.

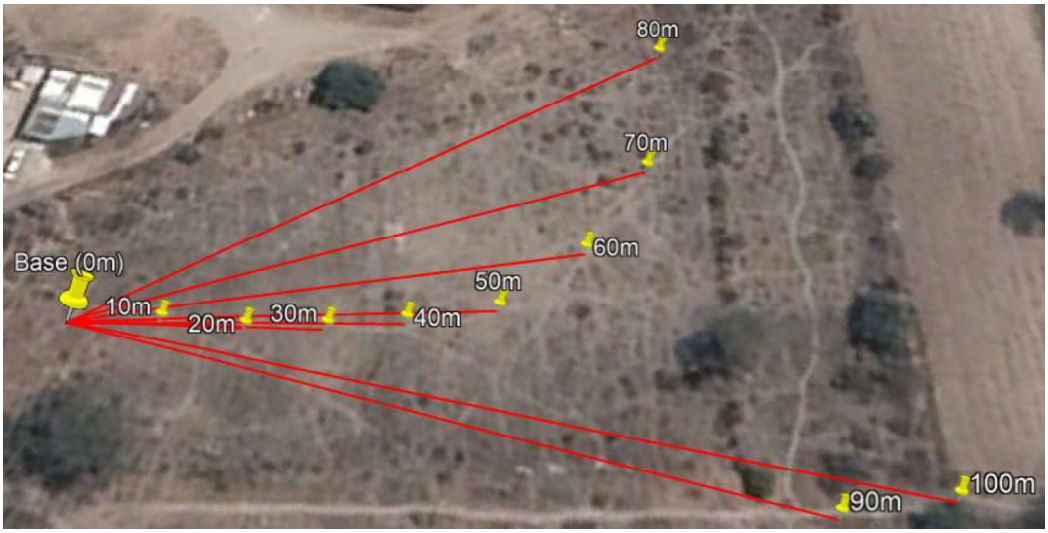

Figura 82: Puntos de mediciones usadas para DigiMesh en RSSI.

5.3.2. Resultados

La Tabla 9 y la Tabla 10 muestran, respectivamente, los resultados para los protocolos Zigbee y DigiMesh. Ambas tablas muestran, en la parte de superior con color azul, la serie de números que denotan las distancias medidas durante el escenario, desde la base a 0 metros hasta 100 metros de distancia. La columna azul de la izquierda divide los datos en 3 secciones: Los paquetes recibidos durante las mediciones, el PER resultante de esos paquetes y la potencia promedio obtenida por la herramienta XCTU en dBm. Delante de cada sección el nivel de elevación de las mediciones es mostrado para cada renglón.

Tabla 9: Resultados de las mediciones en el escenario RSSI bajo el protocolo Zigbee.

RSSI - ZB											
Distancia[m]		0	10	20	30	40	50	60	70	80	90	100
Paquetes	0m	100	97	91	18	92	58	1	0	0	0	0
	1m	100	97	100	48	37	79	56	20	0	0	0
	2m	100	100	97	81	98	62	12	0	0	0	0
	3m	100	100	97	83	78	73	26	6	0	0	0
PER	0m	0	0.03	0.09	0.82	0.08	0.42	0.99	1	1	1	1
	1m	0	0.03	0	0.52	0.63	0.21	0.44	0.8	1	1	1
	2m	0	0	0.03	0.19	0.02	0.38	0.88	1	1	1	1
	3m	0	0	0.03	0.17	0.22	0.27	0.74	0.94	1	1	1
PRx Prom [dBm]	0m	-23	-85	-94	-92	-91	-92	-97	-110	-110	-110	-110
	1m	-50	-83	-80	-86	-87	-89	-89	-93	-110	-110	-110
	2m	-55	-70	-83	-81	-84	-88	-90	-110	-110	-110	-110
	3m	-55	-66	-80	-85	-82	-85	-87	-97	-110	-110	-110

Tabla 10: Resultados de las mediciones en el escenario RSSI bajo el protocolo DigiMesh.

RSSI - DM												
Distancia[m]		0	10	20	30	40	50	60	70	80	90	100
Paquetes	0m	100	100	100	100	100	100	100	92	58	56	100
	1m	100	100	100	100	100	94	96	95	69	99	12
	2m	100	100	100	100	100	100	99	100	44	36	1
	3m	100	100	100	100	100	99	100	100	26	58	9
PER	0m	0	0	0	0	0	0	0	0.08	0.42	0.44	0
	1m	0	0	0	0	0	0.06	0.04	0.05	0.31	0.01	0.88
	2m	0	0	0	0	0	0	0.01	0	0.56	0.64	0.99
	3m	0	0	0	0	0	0.01	0	0	0.74	0.42	0.91
PRx Prom [dBm]	0m	-40	-54	-68	-70	-72	-78	-80	-82	-85	-87	-77
	1m	-40	-45	-60	-65	-65	-74	-75	-70	-76	-72	-77
	2m	-40	-40	-55	-58	-64	-60	-71	-68	-71	-73	-70
	3m	-40	-42	-52	-56	-62	-62	-64	-72	-70	-74	-76

La herramienta para obtener las mediciones de RSSI durante ambos protocolos resultó en beneficio por su simplicidad, dejando un sesgo de error para la potencia recibida. También hace falta recalcar, que los valores 0 en lo paquetes y sus consiguientes 1 en el PER y -110 en la potencia de ambas tablas, no fueron datos obtenidos por la herramienta, si no completados manualmente para terminar de mostrar el comportamiento. La razón se debe a que la herramienta solo muestra la potencia de un paquete recibido, siendo difícil obtener una potencia de un paquete que jamás llegó al receptor, el valor de -110 dBm aparece como el mínimo de potencia en la recepción en las especificaciones del XBee S3B.

La Figura 83, la Figura 84 y la Figura 85 muestran 3 gráficas; las cuales buscan representar de una manera más entendible, cada sección de los resultados de la Tabla 9, enfocándose en el protocolo Zigbee.

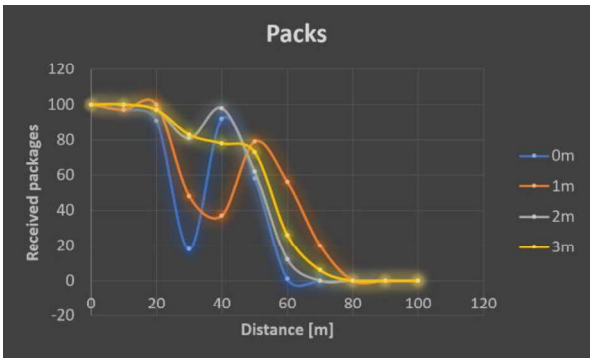

Figura 83: Paquetes recibidos en Zigbee durante el escenario RSSI.

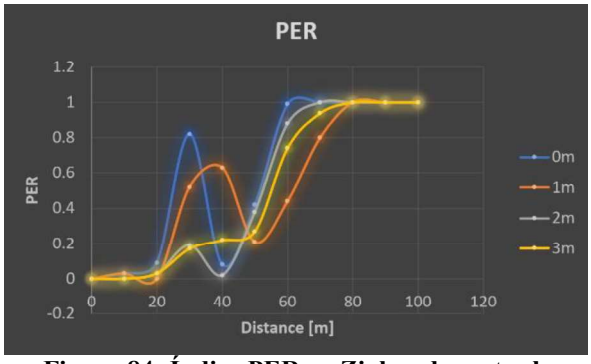

Figura 84: Índice PER en Zigbee durante el escenario RSSI.

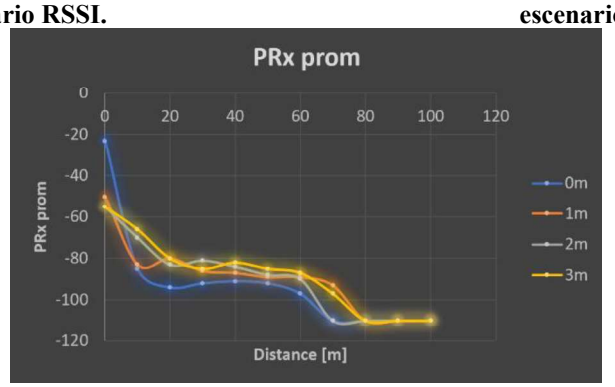

Figura 85: Potencia promedio en el receptor durante el escenario RSSI bajo Zigbee.

Lo primero a destacar, es que los radios bajo el protocolo Zigbee perdieron más de la mitad de los paquetes tras los primeros 60 metros, como la Figura 83 y la Figura 84 muestran. Agregando una variación a los 30 metros muy notable en bajas alturas, siendo causada por la vegetación de la zona. Estos sucesos delatan que el protocolo Zigbee no es muy robusto frente a muchas variaciones en los terrenos, lo que puede verse también en la potencia recibida (Figura 85), donde se pierden cerca de 20 dBm en el primer metro de separación; teniendo una caída de potencia más amortiguada hasta la pérdida total de información.

La elevación juega un papel importante para este protocolo, al no poder mantener la transmisión a elevaciones cercanas al suelo, pues depende de las irregularidades de este para la buena transmisión, recalcando que a 1 metro de elevación existió una inusual mejora en la recepción de los paquetes después de su caída a 30 metros, sin embargo, esta mejora no resulta de alta relevancia.

La Figura 86, la Figura 87 y la Figura 88 muestra de igual forma 3 gráficas, con el mismo propósito de las figuras anteriores de representar la información, ahora de la Tabla 10, siendo entonces orientadas a los resultados bajo el protocolo DigiMesh.

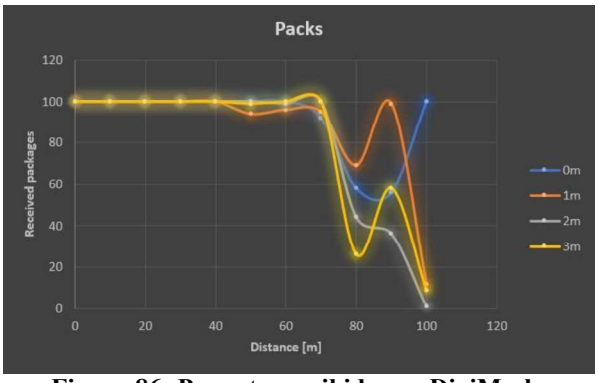

Figura 86: Paquetes recibidos en DigiMesh durante el escenario RSSI.

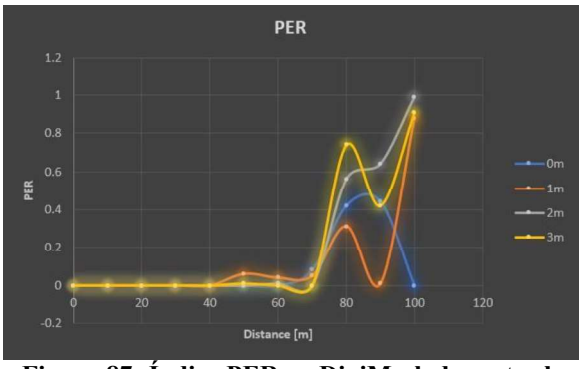

Figura 87: Índice PER en DigiMesh durante el escenario RSSI.

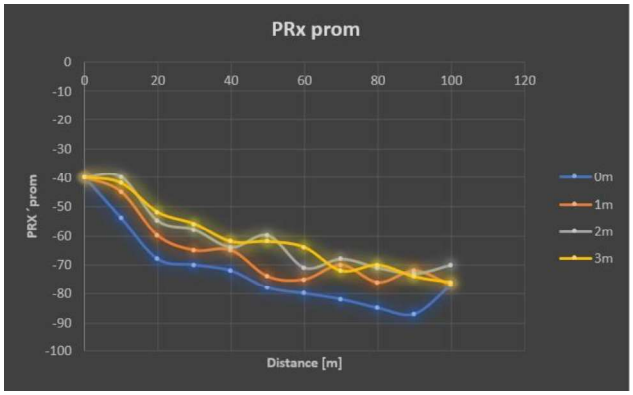

Figura 88: Potencia promedio en el receptor durante el escenario RSSI, bajo DigiMesh.

La Figura 86 y la Figura 87 para DigiMesh revelan llegar unos metros más lejos sin presentar variaciones extremas como ocurrió con el protocolo Zigbee, a los 80 metros hay una caída en todos los paquetes debido a la vegetación (nopales y arbustos) en la zona, lo que impidió la buena recepción de los demás datos y que incluso ocurrió un evento único. El evento es caracterizado como que el terreno asciende bastante a los 100 metros (Figura 174), por lo que los paquetes llegaron correctamente en ese punto y fue imposible replicar esa medición metros más adelante o a la redonda, sin violar el principio de no favorecer los puntos altos. La potencia bajo este protocolo también muestra no disminuir tanto como con Zigbee, la Figura 88 muestra un mayor amortiguamiento, el cual no llega a la potencia mínima recibida por el receptor bajo Zigbee (-97 dBm).

Es posible, por lo sucedido a los 100 metros y elevación 0 metros, que el protocolo DigiMesh llegara a mayor distancia, sin embargo, las caídas por la vegetación supondrían un problema para obtener la máxima distancia. Cabe destacar, que DigiMesh tiene afectaciones por vegetación más densa (nopales y arboles). Aunado a esto, el terreno tiene una elevación escalonada más pronunciada después de los 100 metros, por lo que existe una ineludible ventaja por parte de este terreno para las distancias posteriores.

5.4. Punto a Punto (P2P)

5.4.1. Puntos de medición

Los puntos de esta medición procuraron ser los mismos que las del escenario pasado. Debido a la extensión de la distancia lograda con los Arduinos, más puntos fueron agregados. Pasando los 100 metros, algunos terrenos pasaron a ser particulares, por lo que fue imposible continuar la secuencia de 10 metros y los puntos quedaron limitados a los bordes de los terrenos, ya que de esta forma no se invadía propiedad privada. A pesar de lo anterior, las elevaciones altas que favorecieran la comunicación entre los radios fueron evitadas, por lo que las conexiones fueron en las zonas bajas de los escalonados y no tras subir el escalón del terreno.

Los puntos resultantes son mostrados en las siguientes figuras, donde la Figura 89 muestra los puntos tomados para Zigbee y la Figura 90 los puntos de DigiMesh.

Figura 89: Puntos de mediciones usadas para Zigbee.

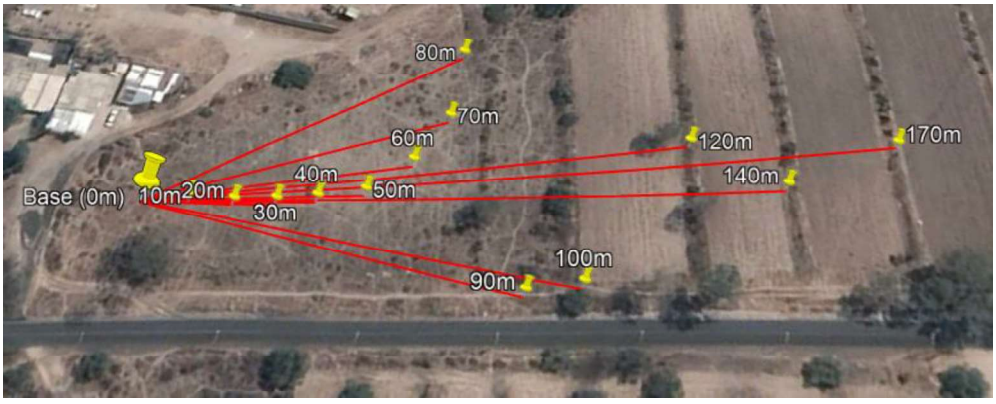

Figura 90: Puntos de mediciones usadas para DigiMesh.

5.4.2. Resultados

La Tabla 11 y la Tabla 12 muestran los resultados obtenidos durante el escenario P2P bajo los protocolos Zigbee y DigiMesh respectivamente. Los números del 0 a 120 y 0 a 160 en la franja superior azul corresponden a las distancias empleadas en las mediciones.

La columna azul en la izquierda, en ambas tablas, divide los datos en 5 secciones de interés: Los paquetes recibidos durante la medición, el PER obtenido con respecto a los paquetes, el tiempo de *retraso* (diferencia de tiempo desde que salió el mensaje del transmisor hasta que fue recibido por el receptor), el *throughput* (el número de bits de información mandada cada segundo) y el tiempo de procesamiento (cuando tardó en mandar/recibir desde el primer paquete hasta el último). Seguido de cada sección, aparecen las elevaciones de cada medición y en el caso del tiempo de procesamiento hay una franja extra para distinguir los datos de recepción de los de transmisión.

Tabla 11: Resultados de las mediciones en el escenario P2P bajo el protocolo Zigbee.

P2P - ZB													
Distancia[m]		**0**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**120**
Paquetes recibidos	**0m**	200	200	200	50	120							
	1m	200	200	199	188	200	182	135	177	198	185	181	162
	2m	200	200	192	200	200	199	200	200	109	181	176	139
	3m	200	200	200	200	200	200	190	200	200	180	199	76
PER	**0m**	0	0	0	0.75	0.4							
	1m	0	0	0.005	0.06	0	0.09	0.325	0.115	0.01	0.075	0.095	0.19
	2m	0	0	0.04	0	0	0.005	0	0	0.455	0.095	0.12	0.305
	3m	0	0	0	0	0	0	0.05	0	0	0.1	0.005	0.62
Tiempo de delay prom[s]	**0m**	0.0467	0.09775	0.10571	0.7839	118.25							
	1m	0.0432	0.06418	0.23971	7.95787	136.755	70.4955	31.2639	9.81869	3.108199	8.1324	15.1113	61.2932
	2m	0.041	0.04364	5.13987	0.04333	136.755	69.0802	2.68971	2.78536	94.70719	6.52901	7.45371	76.7344
	3m	0.0415	0.04225	0.04447	0.04268	136.77	68.2198	3.40282	2.67702	2.795143	5.6838	3.17213	128.902
Throughput [kbps]	**0m**	0.7013	0.70112	0.70095	0.25666	0.28023							
	1m	0.7011	0.70117	0.69666	0.71489	0.70113	0.64174	0.47451	0.648	0.694011	0.63695	0.68798	0.38512
	2m	0.7011	0.7011	0.6729	0.70104	0.70114	0.70123	0.70564	0.70101	0.190826	0.64464	0.61692	0.49091
	3m	0.7011	0.70101	0.70105	0.70116	0.70101	0.70111	0.63449	0.70113	0.701088	0.63044	0.6698	0.17138

Tiempo de procesamiento [min]	0 m	Rx	2.2816	2.28207	2.28262	1.55848	3.42575								
		Tx	2.2491	2.24875	2.24923	2.2485	2.24865								
	1 m	Rx	2.282	2.2819	2.2852	2.10383	2.28202	2.26883	2.27603	2.1852	2.282383	2.32358	2.1047	3.36522	
		Tx	2.2484	2.24872	2.24885	2.2489	2.2485	2.2489	2.2486	2.24857	2.24895	2.24838	2.24892	2.24883	
	2 m	Rx	2.2822	2.28213	2.28265	2.28232	2.28198	2.2703	2.26743	2.28242	4.5696	2.2462	2.28232	2.26517	
		Tx	2.2489	2.24873	2.24917	2.24853	2.24892	2.2484	2.24875	2.24897	2.24856	2.24858	2.24847	2.24865	
	3 m	Rx	2.2821	2.28242	2.2823	2.28193	2.28243	2.28208	2.39563	2.28203	2.282167	2.28412	2.37683	3.54758	
		Tx	2.2485	2.24917	2.24843	2.24855	2.2491	2.24842	2.24907	2.24925	2.24883	2.2488	2.24872	2.24875	

Tabla 12: Resultados de las mediciones durante el escenario P2P bajo el protocolo DigiMesh.

P2P - DM															
Distancia[m]		0	10	20	30	40	50	60	70	80	90	100	120	140	170
Paquetes recibidos	0 m	200	200	200	200	187	200	140	15						
	1 m	200	200	200	200	194	198	69	196	93	196	124	64	59	49
	2 m	200	200	200	200	198	199	200	199	192	166	140	132	75	123
	3 m	200	198	200	199	200	200	188	194	186	75	43	23	31	50
PER	0 m	0	0	0	0	0.065	0	0.3	0.925						
	1 m	0	0	0	0	0.03	0.01	0.655	0.02	0.535	0.02	0.38	0.68	0.705	0.755
	2 m	0	0	0	0	0.01	0.005	0	0.005	0.04	0.17	0.3	0.34	0.625	0.385
	3 m	0	0.01	0	0.005	0	0	0.06	0.03	0.07	0.625	0.785	0.885	0.845	0.75
Tiempo de delay prom[s]	0 m	0.0252	0.02254	0.02245	0.02263	34.7903	72.3096	7.6232	552.7493						
	1 m	0.0249	0.0252	0.02243	0.03336	10.9791	73.3884	18.7491	458.3238	14.0665	4.5552	111.9388	37.5190	68.7008	402.8683
	2 m	0.0247	0.0250	0.0249	0.02241	5.8724	73.0721	3.6843	141.7026	17.8560	18.3405	31.9117	19.2625	26.4681	82.3106
	3 m	0.0240	0.9779	0.0237	0.2815	3.6839	72.3231	19.6198	144.5007	13.4480	44.4478	96.7722	104.3361	59.8454	75.7743
Throughput [kbps]	0 m	0.7011	0.7012	0.7011	0.7010	0.6126	0.7012	0.7088	0.7556						
	1 m	0.7011	0.7011	0.7012	0.7010	0.7294	0.6942	0.4850	1.2865	0.7316	0.6954	0.1438	0.2277	0.2930	0.0191
	2 m	0.7012	0.7010	0.7010	0.7012	0.6941	0.6975	0.7012	0.6976	0.6725	0.5818	0.5030	0.4463	0.2567	0.2177
	3 m	0.7012	0.6942	0.7011	0.6975	0.7011	0.7012	0.6497	0.6800	0.5484	0.2629	0.0997	0.0556	0.1241	0.3513

Tiempo de procesamiento [min]	**0 m**	**Rx**	2.2822	2.2819	2.2822	2.2825	2.4420	2.2818	1.5802	0.1588							
		Tx	2.2488	2.2486	2.2490	2.2490	2.2486	2.2489	2.2487	2.2487							
	1 m	**Rx**	2.2820	2.2820	2.2818	2.2823	2.1278	2.2819	1.1382	1.2189	1.0170	2.2547	6.8975	2.2483	1.6107	20.5560	
		Tx	2.2492	2.2486	2.2489	2.2491	2.2485	2.2487	2.2493	2.2488	2.2484	2.2488	2.2488	2.2485	2.2491	2.2485	
	2 m	**Rx**	2.2820	2.2823	2.2823	2.2817	2.2820	2.2823	2.2820	2.2820	2.2839	2.2826	2.2266	2.2363	2.3371	4.5202	
		Tx	2.2490	2.2494	2.2486	2.2485	2.2488	2.2488	2.2486	2.2488	2.2488	2.2488	2.2485	2.2494	2.2485	2.2488	
	3 m	**Rx**	2.2819	2.2817	2.2821	2.2823	2.2821	2.2819	2.3149	2.2823	2.7136	2.2826	3.4487	3.3093	1.9991	1.1385	
		Tx	2.2486	2.2490	2.2489	2.2493	2.2488	2.2489	2.2487	2.2490	2.2487	2.2490	2.2489	2.2488	2.2488	2.2483	

La Figura 91 y la Figura 92 competen a una representación gráfica de las primeras 2 secciones de la Tabla 11, siendo estas, los paquetes recibidos y su consecuente PER, respectivamente.

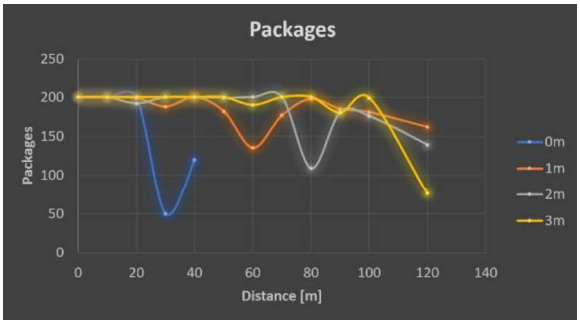

Figura 91: Paquetes recibidos en Zigbee durante el escenario P2P.

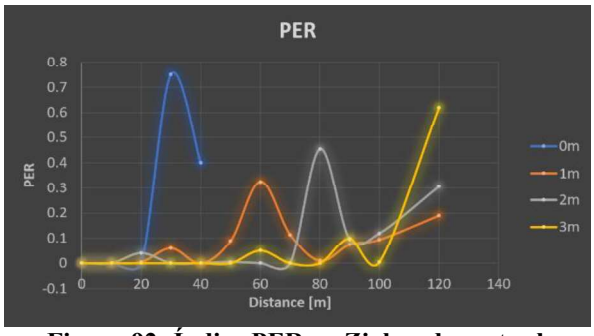

Figura 92: Índice PER en Zigbee durante el escenario P2P.

El aumento en la distancia mostrado en las figuras 91 y 92 bajo el protocolo Zigbee, de 60 metros en el escenario RSSI a los 120 metros obtenidos en este escenario, podría considerarse como algo discordante, sin embargo, los factores que alteran este resultado provienen de la forma en que las mediciones fueron tomadas. La herramienta de XCTU requiere conectarse a un dispositivo antes de empezar la prueba y cada paquete del cual no reciba respuesta, se considera como perdido. Al estar en una "prueba" los radios toman los reintentos para mandar (máximo 4 en Zigbee [26] y máximo 10 en DigiMesh [22], aunque fue configurado a 4); además de no necesitar una conexión predeterminada y eso sin mencionar las elevaciones del terreno las cuales ayudarían en cierta manera a continuar con LOS (3 metros de altura en la entena con respecto a la base a los 100 metros de distancia, como es apreciado en la altimetría de la Figura 174 y 7 metros de altura a los 120 metros de distancia, como la Figura 175 muestra).

La Figura 93 y la Figura 94 muestran los resultados correspondientes a las primeras 2 secciones de la Tabla 12, donde los paquetes recibidos y el consecuente PER son mostrados respectivamente usando el protocolo DigiMesh.

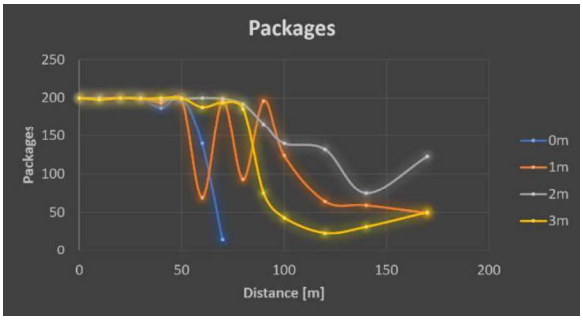

Figura 93: Paquetes recibidos en DigiMesh durante el escenario P2P.

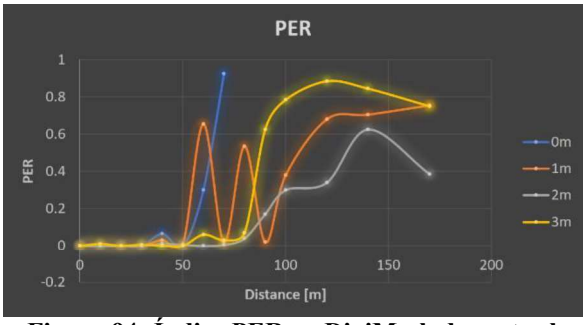

Figura 94: Índice PER en DigiMesh durante el escenario P2P.

Estas figuras señalan también un incremento de la distancia por el cambio de herramienta. Si bien, la Figura 90 muestra que la máxima distancia fue de 170 metros, la medición no fue incluida en los resultados por ser 8 paquetes en una elevación de 2 metros. El aumento de distancia en este protocolo no fue de tal magnitud como lo fue con el protocolo Zigbee. Además, el protocolo Zigbee muestra mayor estabilidad frente al protocolo DigiMesh; donde DigiMesh mostró relativa estabilidad solo con una elevación de 2 metros.

La Figura 95 y la Figura 96 muestran los resultados en términos del retardo (*Delay*) punto a punto para los protocolos Zigbee y DigiMesh respectivamente. Ambas figuras muestran un aumento en el retardo a los 40 y 50 metros. El retardo en DigiMesh no es tan pronunciado en los primeros metros, pero lo compensa con su retraso a los 70 metros. Ambos picos indican una pérdida de comunicación que después es restablecida con DigiMesh de una forma tardía.

En general, los tiempos de retraso son relativamente bajos, siendo el protocolo Zigbee el que presenta una mayor rapidez en la reconexión, sin embargo, no es algo que haga mucha diferencia. El caso de los tiempos para el *Delay* con DigiMesh presenta una conexión perdida y recuperada constantemente.

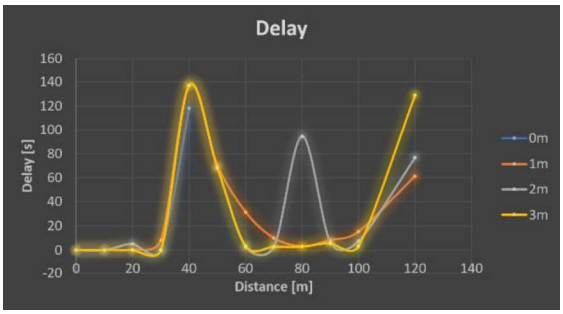

Figura 95. *Delay* de los paquetes en Zigbee durante el escenario P2P.

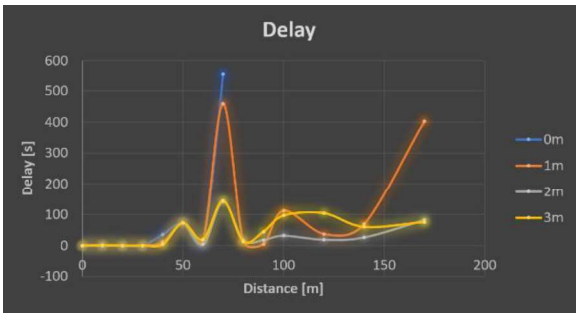

Figura 96. *Delay* de los paquetes en DigiMesh durante el escenario P2P.

Los cálculos de retardo tienen un ligero fallo, mismo que resulta obvio al considerar que los retardos en DigiMesh son de más de 120 segundos (que significaría estar 2 minutos

esperando por un mensaje) y los 400 segundos indican una espera media de poco más de 6 minutos por mensaje, lo que resulta ilógico. Debido a la forma de operar de los mensajes, y con motivo de la simpleza, resulta complicado saber qué paquete es el que llega, dado que el número reflejado es del número de paquetes recibidos con un contador. El contador entonces solo tiene un cambio cuando llega un paquete que resulte ser 10 veces menor al último mandado. De esta forma, tanto el contador cuando indica 0 paquetes como cuando 20 paquetes es menor a cuando indica 200 paquetes recibidos.

El fallo aumenta con algo que en este trabajo tomará el nombre de "*lag*", este evento ocurre cuando el receptor no recibe ningún paquete y posteriormente tiene lecturas de una cantidad determinada de paquetes con el mismo tiempo de llegada. Estos 2 desperfectos sumados entorpecen la obtención precisa de la información, lo que radicó en el cambio de cómo se consideraban los paquetes para los resultados debido a la ambigüedad existente. La lógica inicial es que no pueden existir paquetes que lleguen antes de que fueran transmitidos.

Esto, junto con el formato de pruebas de 200 paquetes, planteó una solución aceptable: enumerar las pruebas y anotar el número de pruebas en la recepción que eran consideradas en los resultados, tomar las mediciones con sus respectivas elevaciones y pruebas de guarda. A la hora de comparar los datos de transmisión y recepción, solo el tiempo de inicio de la primera prueba enumerada es considerado y comparado con la prueba que más se acerque en tiempo y que sea menor al número de la prueba en la recepción. Siguiendo el formato de las elevaciones con las bandas de guarda (elevación de menor a mayor con separación de 2 bandas de guarda por elevación), y al terminar la medición, ocupar las bandas de guarda en el transmisor para que los tiempos iniciales de las pruebas vuelvan a ser mínimos y repetir el proceso. Esta forma de trabajo llevó una manera más imparcial para los protocolos, con las posibles consecuencias de un *delay* algo alejado de lo real, pero que dictamina la inestabilidad de las transmisiones en ambos protocolos.

Siguiendo este nuevo orden, la Figura 97 y la Figura 98 muestran las gráficas del *Throughput* observado en Zigbee y en DigiMesh, respectivamente. Estas gráficas facilitan la comprensión de lo acontecido durante las mediciones.

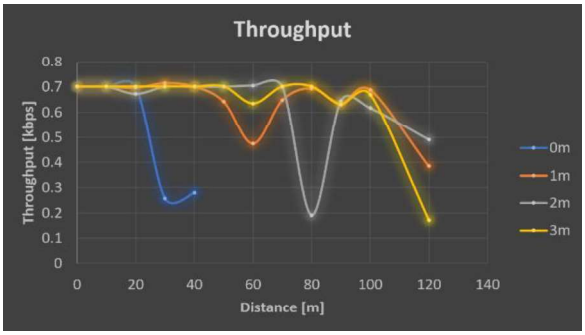

Figura 97: *Troughput* de los paquetes en Zigbee durante el escenario P2P.

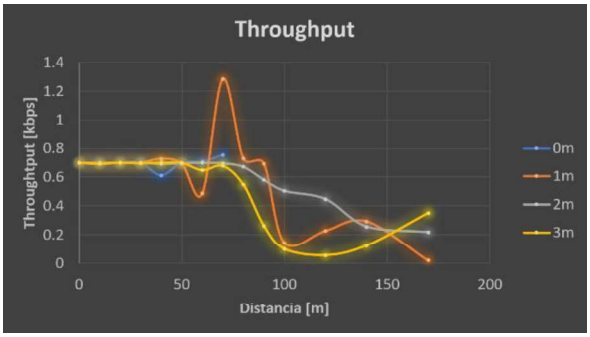

Figura 98: *Troughput* de los paquetes en DigiMesh durante el escenario P2P.

El *throughput*, en ambos casos, muestra una mayor claridad con la relación con los paquetes recibidos en el tiempo que tomó en recopilarlos. Debido a la falta de información, los bits del frame de los protocolos fueron ignorados, considerando solo los bits de información usados en las mediciones.

De igual forma, a los 40 y 50 metros en la Figura 97, muestra el error que puede existir por la forma de procesar los resultados; ya que, a pesar de que el retraso en esas distancias fue muy alto, los paquetes de toda la prueba fueron recibidos correctamente y en el tiempo promedio. Lo acontecido indicaría un corrimiento en la información que no afectó a todas las mediciones debido a la compensación con las pruebas de guarda. La Figura 98, muestra de igual manera que DigiMesh no presentó corrimientos tan marcados en la información como fue con Zigbee, aunque el bajo *throughput* resulta tan evidente como el retraso constante de sus paquetes.

La Figura 99, la Figura 100, la Figura 101 y la Figura 102; muestran la comparación de las últimas 2 secciones de las tablas. Las figuras de la izquierda (Figuras 99 y 101) corresponden al protocolo Zigbee y a la derecha las correspondientes al protocolo DigiMesh (Figuras 100 y 102).

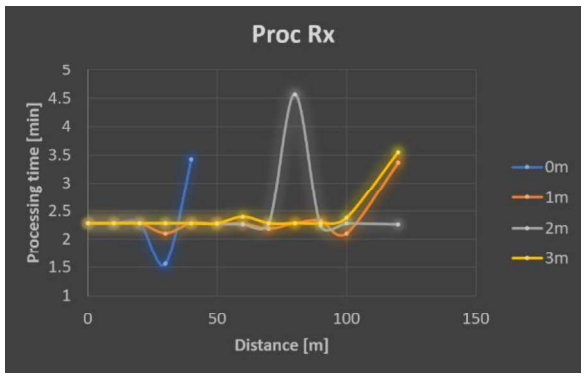

Figura 99: Tiempo de procesamiento en el receptor para Zigbee durante el escenario P2P.

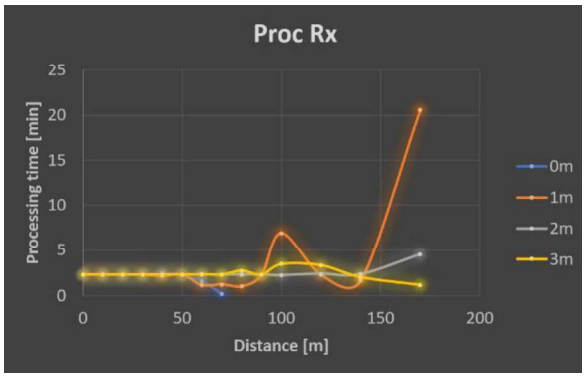

Figura 100: Tiempo de procesamiento en el receptor para DigiMesh durante el escenario P2P.

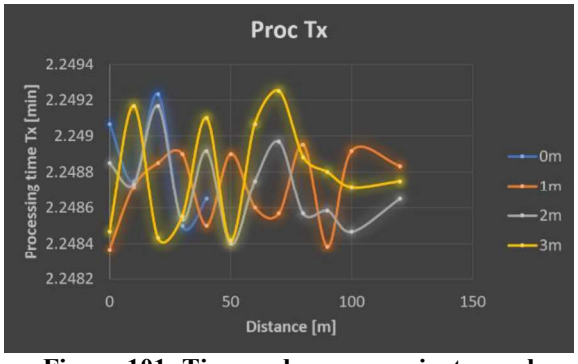

Figura 101: Tiempo de procesamiento en el transmisor para Zigbee durante el escenario P2P.

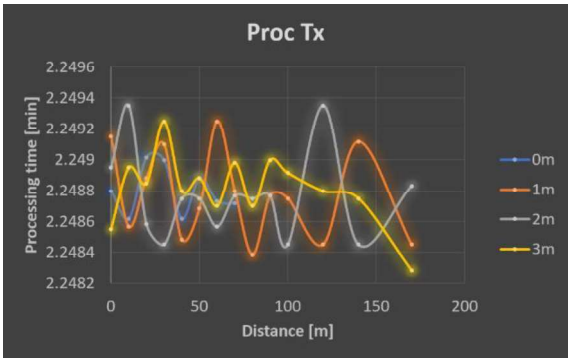

Figura 102: Tiempo de procesamiento en el transmisor para DigiMesh durante el escenario P2P.

La Figura 99 y la Figura 100 muestran una cierta estabilidad en los tiempos de procesamiento. Valores ajenos al promedio de minutos indican pruebas incompletas y refuerzan las conclusiones del resultado obtenido en el *throughput* de sus respectivos protocolos.

El caso de la Figura 101 muestra las variaciones que se tiene en la transmisión, que, a pesar de ser tener variaciones mínimas, no deberían existir debido a que la transmisión es programada con tiempos iguales y fijos en el Arduino. La Figura 102 corrobora que estas variaciones son del Arduino propiamente y que las mismas oscilan entre unas cuantas décimas de milisegundos, con un promedio del tiempo programado en el Arduino (300 ms).

Capítulo 6. Resultados de los escenarios RP, P2M y NM

6.1. Introducción

El presente capítulo, funge como continuación del capítulo anterior, mostrando los resultados procesados de los escenarios RP, P2M y NM; respectivamente. El formato no sufre alteración con respecto a lo establecido en el capítulo anterior.

6.2. Salto (RP)

6.2.1. Puntos de medición

Para este escenario, los puntos de medición tomaron como referencia la última distancia aceptable por el escenario P2P. Debido a la diferencia de paquetes recibidos en las elevaciones, las mediciones estarán dividas en 2 partes. La primera parte contendrá la MDA de la elevación 0 (Figura 103 para el protocolo Zigbee y Figura 104 para el protocolo DigiMesh), mientras que la segunda parte comprenderá a las elevaciones de 1, 2 y 3 metros (Figura 105).

Un repetidor está colocado en la MDA de cada elevación con respecto del emisor, mientras que el receptor a una distancia similar a la del emisor con el repetidor (dentro de lo posible debido a los terrenos privados). La distancia máxima de comunicación posible fue prioridad en este escenario, por lo que las mediciones abarcan las zonas altas del terreno. Las diferentes elevaciones están perpetradas únicamente por el repetidor. Tanto el emisor como el receptor estuvieron a nivel del suelo.

Figura 103: Rutas usadas para elevación 0 bajo el protocolo ZB.

Figura 104: Rutas usadas para elevación 0 bajo el protocolo DM.

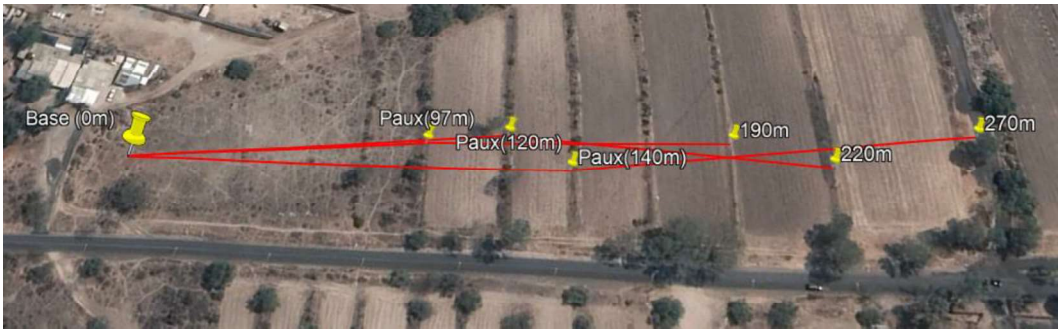

Figura 105. Imágenes satelitales pertinentes a las mediciones del escenario RP, elevaciones 1, 2 y 3.

La conexión entre los radios alcanzo los 270 metros de distancia (durante la medición en elevaciones 1, 2 y 3 metros), pasando esta distancia, se ubica un hogar con terreno de cultivo activo, por lo que el límite de los puntos fue de los 270 metros. Las mediciones corresponden a los últimos 3 puntos de mediciones aceptables del escenario anterior, agregando 3 puntos de la nueva distancia del transmisor.

6.2.2. Resultados

La Tabla 13 y Tabla 14 la muestran los resultados obtenidos durante el escenario RP bajo los protocolos Zigbee y DigiMesh respectivamente. Debido a que el interés de este escenario está enfocado al estudio del máximo rango de comunicación, las distancias tomadas fueron a 60 metros para el protocolo Zigbee y a 120 metros para el protocolo DigiMesh, únicamente para la elevación a nivel del suelo. Para las demás elevaciones, las distancias ocupadas fueron 190, 220 y 270 metros exclusivamente. Las distancias están mostradas en el renglón superior de tablas.

La columna izquierda divide la información en 5 secciones de interés, siendo estos: los pquetes recibidos durante la medición, el PER obtenido respecto a los paquetes enviados (max. 200), el tiempo de retraso promedio de los paquetes, el *throughput* de información durante las mediciones y el tiempo de procesamiento de receptor y transmisor. Seguido de cada sección, aparece una franja que denota las elevaciones durante cada medición. Para el tiempo de procesamiento una franja extra es agregada para diferenciar la recepción de la transmisión.

Tabla 13: Resultados obtenidos durante las mediciones del escenario RP bajo Zigbee.

RP - ZB						
Distancia[m]			**60**	**190**	**220**	**270**
Paquetes recibidos	**0m**		128			
	1m		176	176	200	111
	2m		200	200	189	144
	3m		200	200	105	182
PER	**0m**		0.36			
	1m		0.12	0.12	0	0.445
	2m		0	0	0.055	0.28
	3m		0	0	0.475	0.09
Tiempo de delay prom[s]	**0m**		80.1266			
	1m		129.0172	54.7656	42.5919	23.3987
	2m		0.3399	42.8275	129.7232	27.8664
	3m		0.2399	42.8953	94.6943	10.4210
Throughput[kbps]	**0m**		0.3307			
	1m		0.3386	0.7157	0.7758	0.4304
	2m		0.7756	0.7671	0.3087	0.5652
	3m		0.7753	0.7839	0.4134	0.6940
Tiempo de procesamiento [min]	**0m**	**Rx**	3.0965			
		Tx	2.0295			
	1m	**Rx**	4.1583	1.9673	2.0625	2.0631
		Tx	2.0293	2.0296	2.0294	2.0296
	2m	**Rx**	2.0628	2.0858	4.8981	2.0383

		Tx	2.0297	2.0298	2.0297	2.0291
	3m	Rx	2.0636	2.0410	2.0318	2.0981
		Tx	2.0294	2.0298	2.0294	2.0290

Tabla 14: Resultado de las mediciones durante el escenario RP bajo el protocolo DigiMesh.

RP - DM						
Distancia[m]			**120**	**190**	**220**	**270**
Paquetes recibidos	**0m**		199			
	1m		200	200	199	199
	2m		200	200	188	198
	3m		200	198	199	197
PER	**0m**		0.005			
	1m		0	0	0.005	0.005
	2m		0	0	0.06	0.01
	3m		0	0.01	0.005	0.015
Tiempo de *delay* prom[s]	**0m**		0.3454			
	1m		0.5483	0.2071	0.6862	2.1365
	2m		0.6959	0.9145	20.8169	2.6099
	3m		0.0981	2.6902	1.1893	5.9276
Throughput[kbps]	**0m**		0.7711			
	1m		0.7757	0.7757	0.7424	0.7626
	2m		0.8442	0.7750	0.7534	0.7686
	3m		0.7756	0.7680	0.7462	0.7659
Tiempo de procesamiento [min]	**0m**	**Rx**	2.0646			
		Tx	2.0298			
	1m	**Rx**	2.0628	2.0627	2.1445	2.0876
		Tx	2.0289	2.0294	2.0294	2.0289
	2m	**Rx**	1.8954	2.0644	1.9964	2.0610
		Tx	2.0293	2.0290	2.0295	2.0289
	3m	**Rx**	2.0629	2.0624	2.1336	2.0578
		Tx	2.0296	2.0298	2.0292	2.0294

La Figura 106 y la Figura 107 muestran los resultados correspondientes a las primeras 2 secciones de la Tabla 13 para el protocolo Zigbee (paquetes recibidos y su PER). Por su parte, la Figura 108 y la Figura 109, muestran otras 2 gráficas; correspondientes a las mismas secciones, ahora de la

Tabla 14 para el protocolo DigiMesh, y con el mismo orden. Estas figuras fueron agrupadas con la finalidad de permitir la comparación del desempeño de ambos protocolos durante este escenario.

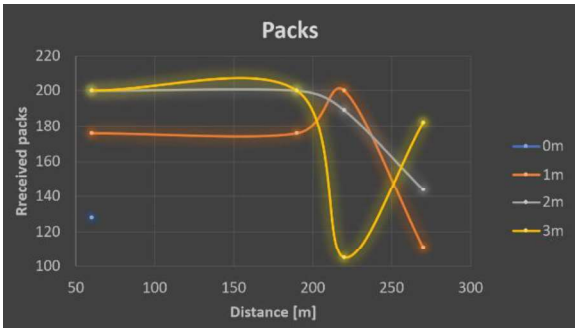

Figura 106: Paquetes recibidos en Zigbee durante el escenario RP.

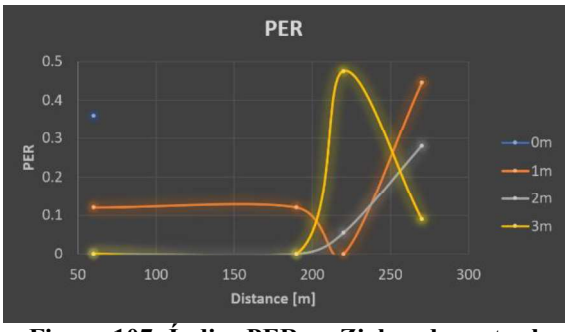

Figura 107: Índice PER en Zigbee durante el escenario RP.

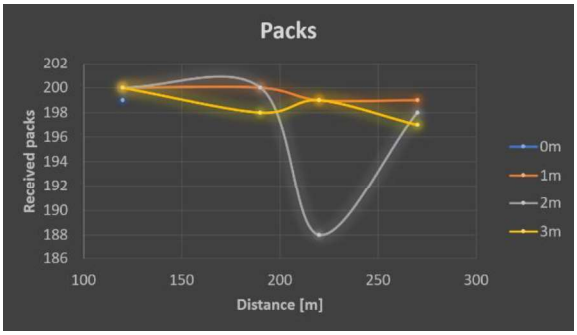

Figura 108: Paquetes recibidos en DigiMesh durante el escenario RP.

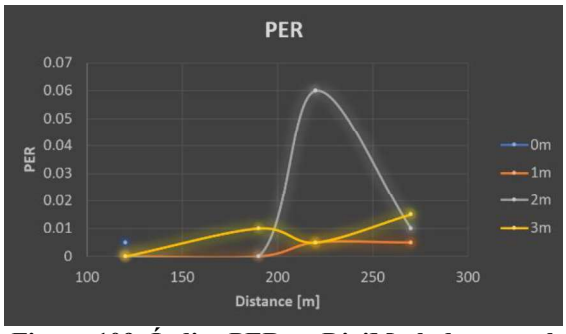

Figura 109: Índice PER en DigiMesh durante el escenario RP.

Existe una diferencia considerable entre la máxima distancia a 0 metros de elevación y la máxima distancia en las demás elevaciones, lo cual supuso una forma diferente de llevar a cabo las mediciones de este escenario. Dado que el mayor interés reside en obtener el rendimiento de la comunicación a la máxima distancia con un repetidor, las mediciones están centradas en las elevaciones de 1, 2 y 3 metros; considerando la elevación a 0 metros en un único caso de medición. Partiendo de este punto, las figuras 106 y 107 muestran una diferencia entre los protocolos a nivel del suelo, donde el problema de conexión fue más notable en el protocolo Zigbee; sin embargo, sería incorrecto no considerar la ventaja del protocolo DigiMesh por la altura inevitable tras pasar los 100 metros de distancia (ver Figura 175 del Anexo A, página XVII).

Los resultados del protocolo Zigbee con elevaciones a 1, 2 y 3 metros distan, en materia del desempeño, de los obtenidos a 60 metros. Lo que resulta sorpresivo es la máxima distancia lograda bajo este protocolo (270 metros), comparando con DigiMesh, el cual también alcanzó la misma distancia. Es palpable una mayor estabilidad por parte de este último protocolo, quien muestra que la distancia máxima para dicho protocolo es mayor a la alcanzada durante la presente tesis; cuando con el protocolo Zigbee, da a entender que no podría prevalecer por muchos metros más.

La elevación del terreno (ver Figura 179 del Anexo A, página XVII) entra en beneficio a ambos protocolos y los resultados para las elevaciones a 3 metros indican que existe una pérdida de potencia en las señales recibidas debido a la multi-propagación de las

ramas y absorción de las señales de RF ocasionada por la vegetación de la zona. Finalmente, es de importancia recalcar que la distancia de las mediciones no pudo exceder los 270 metros debido a la existencia de terrenos privados intransitables.

Las siguientes 2 gráficas presentan la misma sección de *Delay* obtenido en los protocolos Zigbee (Figura 110) y DigiMesh (Figura 111). De esta forma resulta de mayor facilidad la comparación de estos protocolos durante los acontecimientos del escenario.

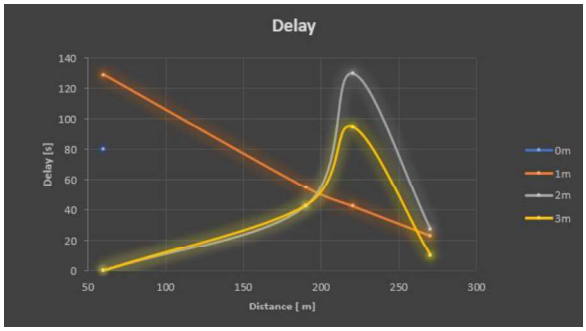

Figura 110: *Delay* de los paquetes con Zigbee durante el escenario RP.

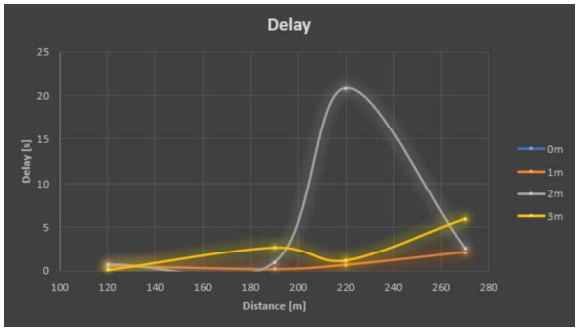

Figura 111: *Delay* de los paquetes con DigiMesh. durante el escenario RP

Lo primero a recalcar aparece con el protocolo Zigbee, donde a 60 metros, muestra posibles corrimientos de información por la inestabilidad del protocolo y un problema específico a 220 metros de distancia; con una curva que supera los 2 minutos de retraso a una elevación de 2 metros, cuando a un metro de elevación el retraso es de 40 s. Es probable, que dicho problema ocurriera debido a un corte de información que fue retomado después, con un número de paquete muy cercano al perdido. Esto ocasionó el corrimiento de esa medición y su consecuente retraso, al ser imposible ocupar las pruebas de guarda para compensarlo. Sin embargo, dicho problema no afectó al retraso en la última medición a 270 metros.

Por parte del protocolo DigiMesh, el retraso mejoró las expectativas en comparación con el desempeño mostrado en el escenario anterior, siendo este, un indicio de la estabilidad en la comunicación con el uso de un repetidor. Aun así, es necesario recalcar el corrimiento de información que obtuvo a los 220 metros, similar a Zigbee, que señalaría una obstrucción debido a la vegetación de la zona a una elevación de 2 metros, pero que no causo un impacto importante.

De igual forma que en las figuras anteriores, las siguientes figuras buscan la comparación de los resultados de las tablas para Zigbee y DigiMesh, ahora con la sección de Throughput. Siendo la Figura 112, la correspondiente con los resultados bajo el protocolo Zigbee, mientras que la Figura 113 corresponde a los resultados bajo el protocolo DigiMesh.

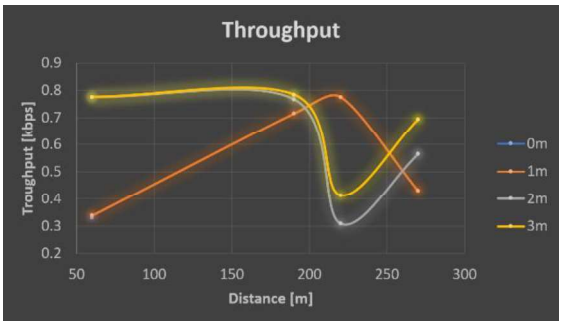

Figura 112: Throughput de los paquetes en Zigbee durante el escenario RP.

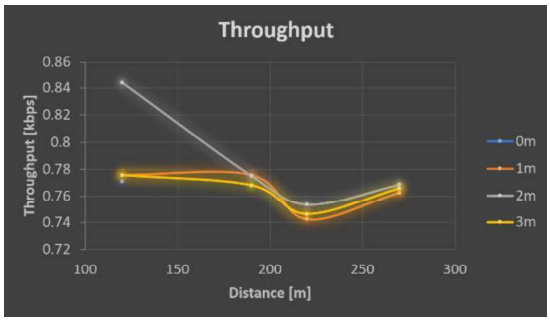

Figura 113: Throughput de los paquetes DigiMesh durante el escenario RP.

El *throughput* mostrado para Zigbee en la Figura 112 muestra las pérdidas de información a 220 metros, así como la pérdida de información a los 60 metros; lo que también apunta a descartar el corrimiento de información y considerar pérdidas importantes de información. Mientras tanto, el *throughput* mostrado en la Figura 113 para DigiMesh sigue mostrando pocas variaciones ante un valor promedio, mismo que refuerza la idea de una mayor distancia para este protocolo de la que fue posible medirse durante este trabajo.

Finalmente, son comparados los tiempos de procesamiento en ambos protocolos. La Figura 114 y la Figura 116, a la izquierda, revelan los tiempos de procesamiento en la recepción y en la transmisión para Zigbee. A la derecha, la Figura 115 y la Figura 117 muestran los tiempos de procesamiento con el mismo orden para DigiMesh.

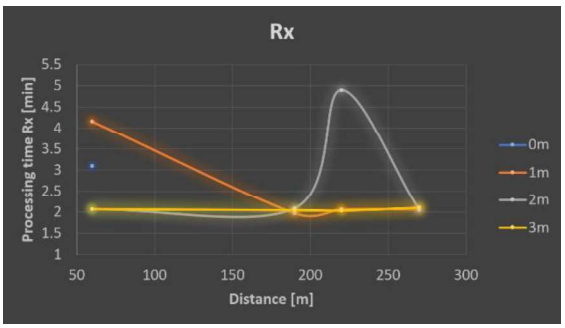

Figura 114: Tiempo de procesamiento en el receptor en Zigbee.

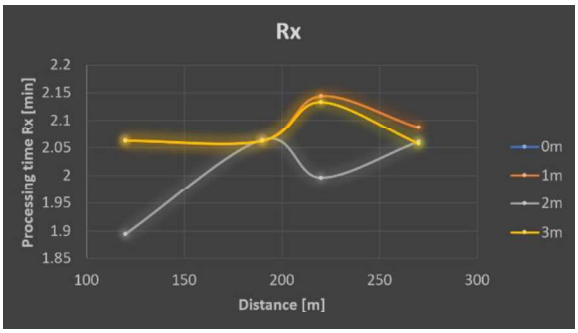

Figura 115: Tiempo de procesamiento en el receptor en DigiMesh.

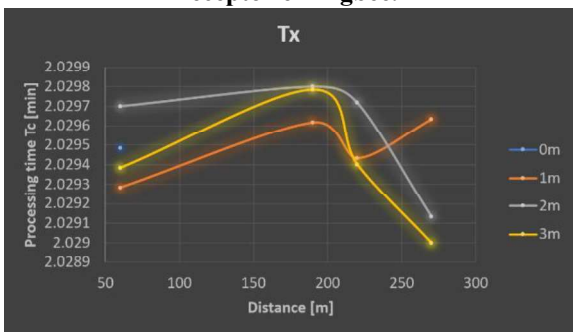

Figura 116: Tiempo de procesamiento en el transmisor en Zigbee.

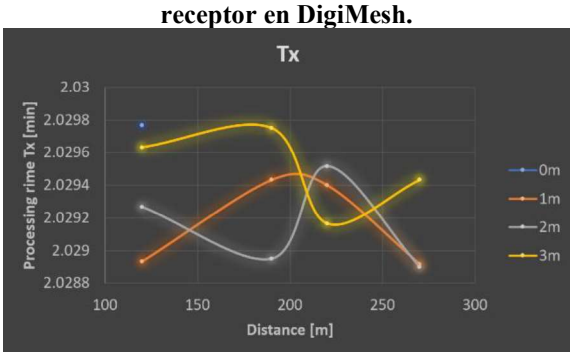

Figura 117: Tiempo de procesamiento en el transmisor en DigiMesh.

La Figura 114 refleja, con mayor claridad, los sucesos ocurridos durante las mediciones con Zigbee, donde a 60 metros de distancia con 0 y 1 metro de elevación, aparece un tiempo mayor al promedio en la recepción. Este tiempo extra, indica una pérdida en los paquetes y que los mismos fueron llenados con paquetes correspondientes a otras pruebas o con paquetes que fueron mandados con cierto *lag*. Esto mismo ocurre a los 220 metros, donde se observa la gran pérdida de información a los 2 metros de elevación, aclarando que a los 3 metros no hubo una pérdida de paquetes que ocasionara el retraso, si no que fue un corrimiento de información. La pérdida de información ocurrió entonces en medio de la transmisión de los paquetes de la prueba y fue recuperada antes de que terminara la misma.

El protocolo DigiMesh, en la Figura 115, revela menores variaciones respecto a Zigbee, donde destacan valores menores al promedio. Estos valores indican una pérdida de información con respecto a un par de paquetes a los 220 metros; aunque, al ser tan pocos, resulta más adecuado señalar el *lag* a favor de este protocolo con lo visto a los 120 metros. El *lag* sería a favor, ya que el *throughput* se dispara con un PER de 0 y un tiempo menor recepción, con ligeros retrasos a los 220 metros con las elevaciones de 1 y 3 metros. Estos resultados destacan una mayor interferencia a una distancia en específico, 220 metros que se le es atribuida a árboles y/o vegetación seca que estaban en los terrenos, siendo DigiMesh el protocolo que logró una mejor mitigación.

La Figura 116 y la Figura 117 solo muestran variaciones ocurridas con milisegundos de diferencia, muy probablemente alteraciones ocasionadas por los Arduinos programados, por lo que es una variación fuera del control. Aunque el promedio no altera a las mediciones en los escenarios.

6.3. Punto Multipunto (P2M)

6.3.1. Puntos de medición

Debido al incremento de vegetación en la zona, nuevos puntos de máxima distancia del escenario P2P fueron considerados junto con otros 2 nuevos puntos de medición a una distancia similar con respecto del receptor. Con la reducción de la distancia, resulta posible tomar las mediciones a todas las elevaciones de interés (0, 1, 2 y 3 metros), lo cual tiene relevancia en este escenario, pues ocurre solo por aprovechar los puntos altos en los terrenos escalonados. Los puntos son los mostrados en la Figura 118.

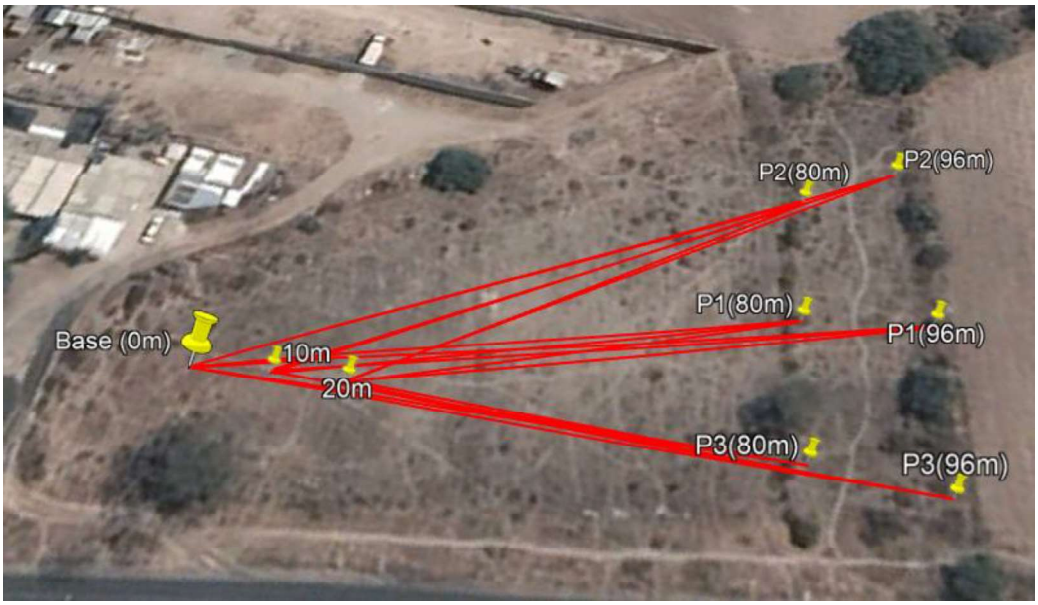

Figura 118: Imágenes satelitales pertinentes a las mediciones del escenario P2M para ambos protocolos.

6.3.2. Resultados

Siguiendo la búsqueda de la comparación entre el desempeño de ambos protocolos, las tablas son agrupadas por los protocolos, por lo que inicialmente la Tabla 15 y la Tabla 16 muestran los resultados del escenario P2M bajo el protocolo Zigbee en los transmisores 1 y 2 respectivamente. Posterior a ellas, la Tabla 17 y la Tabla 18 muestran los resultados bajo el protocolo DigiMesh con el mismo orden de los transmisores. La explicación de los sucesos en dichas tablas es seccionada, usando gráficas, y abarcada más adelante en el mismo capítulo.

La información del transmisor 3 no pudo ser obtenida debido a limitaciones técnicas, sin embargo, su permanencia en las mediciones se mantuvo ya que formaba parte de la información recibida en el receptor

Todas las tablas de esta sección siguen la misma descripción, donde el renglón superior azul muestra las distancias tomadas a 80, 70 y 60 metros para Zigbee y 96, 86 y 76 metros para DigiMesh. La técnica de medición fue contraria de los escenarios anteriores, acercando el receptor (en vez de alejar al transmisor) y manteniendo fijos los transmisores.

La columna izquierda azul divide la información en 5 secciones de interés, siendo estas: los paquetes recibidos, el PER respecto a los paquetes, el tiempo retraso que tiene al ser enviada la información, la cantidad de información en bits transmitida por segundo y el tiempo que tomo cada dispositivo en mandar/recibir desde el primer paquete hasta el último en un determinado número de paquetes. Seguido de cada sección aparece una franja que comprende a las elevaciones de cada medición y para el caso del tiempo de procesamiento hay una franja extra para diferenciar el tiempo en el receptor y en el transmisor

Tabla 15: Resultados de las mediciones del dispositivo 1 durante el escenario P2M bajo Zigbee.

P2M - ZB - 1					
Distancia[m]			80	70	60
Paquetes recibidos	0m		177	200	199
	1m		200	200	193
	2m		199	200	190
	3m		200	200	200
PER	0m		0.115	0	0.005
	1m		0	0	0.035
	2m		0.005	0	0.05
	3m		0	0	0
Tiempo de delay prom[s]	0m		4.7543	0.0750	0.3757
	1m		0.0425	0.2062	1.1947
	2m		0.7256	0.2795	9.9445
	3m		0.0395	0.0460	0.4149
Throughput[kbps]	0m		0.6770	0.7748	0.7714
	1m		0.7748	0.7747	0.7475
	2m		0.7708	0.7748	0.7361
	3m		0.7748	0.7747	0.7744
Tiempo de procesamiento [min]	0m	Rx	4.1830	4.1300	4.1278
		Tx	4.0970	4.0973	4.0972
	1m	Rx	4.1301	4.1306	4.1311
		Tx	4.0970	4.0965	4.0969
	2m	Rx	4.1306	4.1301	4.1301
		Tx	4.0971	4.0973	4.0975
	3m	Rx	4.1300	4.1305	4.1321
		Tx	4.0974	4.0975	4.0970

Tabla 16: Resultados de las mediciones del dispositivo 2 durante el escenario P2M bajo Zigbee.

P2M – ZB - 2				
Distancia[m]		80	70	60
Paquetes recibidos	0m	170	200	200
	1m	200	189	118
	2m	199	200	200
	3m	200	200	192
PER	0m	0.15	0	0
	1m	0	0.055	0.41
	2m	0.005	0	0
	3m	0	0	0.04

Tiempo de delay prom[s]	**0m**		41.8715	0.0638	124.1270
	1m		0.0649	12.8052	199.9818
	2m		0.4513	0.3694	124.3080
	3m		0.0500	0.0351	133.8418
Throughput[kbps]	**0m**		0.6202	0.7748	0.7748
	1m		0.7742	0.7409	0.3053
	2m		0.7708	0.7747	0.7748
	3m		0.7747	0.7747	0.7438
Tiempo de procesamiento [min]	**0m**	**Rx**	4.3855	4.1302	4.1302
		Tx	4.0970	4.0971	4.0974
	1m	**Rx**	4.1334	4.0813	6.1836
		Tx	4.0969	4.0968	4.0971
	2m	**Rx**	4.1306	4.1309	4.1303
		Tx	4.0972	4.0972	4.0969
	3m	**Rx**	4.1304	4.1307	4.1302
		Tx	4.0973	4.0976	4.0972

Tabla 17: Resultados de las mediciones del dispositivo 1 durante el escenario P2M bajo DigiMesh.

P2M - DM - 1					
Distancia[m]			**96**	**86**	**76**
Paquetes recibidos	**0m**		190	195	175
	1m		172	188	193
	2m		119	187	192
	3m		162	194	172
PER	**0m**		0.05	0.025	0.125
	1m		0.14	0.06	0.035
	2m		0.405	0.065	0.04
	3m		0.19	0.03	0.14
Tiempo de delay prom[s]	**0m**		169.8531	135.2724	19.0132
	1m		23.5529	40.9373	14.2378
	2m		42.1782	144.7975	15.2584
	3m		131.2246	138.0444	37.1253
Throughput[kbps]	**0m**		0.7516	0.7590	0.6770
	1m		0.8453	0.7328	0.7477
	2m		0.6663	0.4137	0.7438
	3m		0.7593	0.7471	0.6811
Tiempo de procesamiento [min]	**0m**	**Rx**	4.0450	4.1109	4.1356
		Tx	4.0933	4.0970	4.0948

	1m	Rx	3.2558	4.1048	4.1299
		Tx	4.0985	4.1016	4.0938
	2m	Rx	2.8576	7.2321	4.1303
		Tx	4.0988	4.0945	4.1035
	3m	Rx	3.4135	4.1547	4.0408
		Tx	4.0979	4.0966	4.0961

Tabla 18: Resultados de las mediciones del dispositivo 2 durante el escenario P2M bajo DigiMesh.

P2M - DM – 2					
Distancia[m]			96	86	76
Paquetes recibidos	0m		93	145	145
	1m		191	168	195
	2m		136	135	152
	3m		191	191	185
PER	0m		0.535	0.275	0.275
	1m		0.045	0.16	0.025
	2m		0.32	0.325	0.24
	3m		0.045	0.045	0.075
Tiempo de delay prom[s]	0m		4.5411	56.8331	3.4336
	1m		3.1622	2.9276	4.6287
	2m		6.3807	134.9166	43.9071
	3m		32.8683	61.6038	13.8786
Throughput[kbps]	0m		0.7210	0.7491	0.8067
	1m		0.7388	0.7372	0.7602
	2m		0.7332	0.8749	0.5889
	3m		0.5918	0.7534	0.6801
Tiempo de procesamiento [min]	0m	Rx	2.0638	3.0970	2.8759
		Tx	4.0933	4.0970	4.0948
	1m	Rx	4.1363	3.6461	4.1041
		Tx	4.0985	4.1016	4.0938
	2m	Rx	2.9679	2.4688	4.1300
		Tx	4.0988	4.0945	4.1035
	3m	Rx	5.1638	4.0564	4.3524
		Tx	4.0979	4.0966	4.0961

La disminución de la distancia fue ocasionada por 2 factores. El primero fue el aumento de vegetación en la zona debido a la temporada de lluvias, lo que sugiere un evento muy perjudicial la existencia de estas comunicaciones. El segundo, el factor de lo reducido del terreno, ya que las limitaciones de los terrenos privados solo permitían las

mediciones en las líneas entre terrenos e imposibilitó tomar un distanciamiento apropiado entre los 3 transmisores, al igual que su distancia con respecto del receptor. A pesar de lo anterior, el escenario volvió a reflejar una leve ventaja por parte del protocolo DigiMesh al lograr una distancia extra de cobertura de 16 metros extra la distancia que alcanzó el protocolo Zigbee.

Siguiendo el esquema de comparación, las siguientes figuras muestran 4 gráficas. La Figura 119 y Figura 120 responden a los resultados del dispositivo 1 bajo los protocolos Zigbee y DigiMesh, respectivamente. La Figura 121 y la Figura 122, por su parte, contienen los resultados del dispositivo 2 bajo los protocolos Zigbee y DigiMesh.

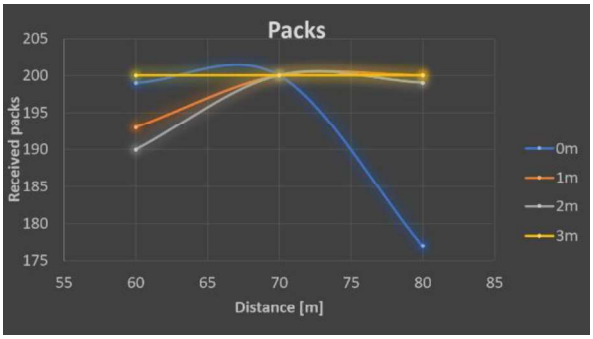

Figura 119: Paquetes recibidos en el dispositivo 1 bajo el protocolo Zigbee durante el escenario P2M.

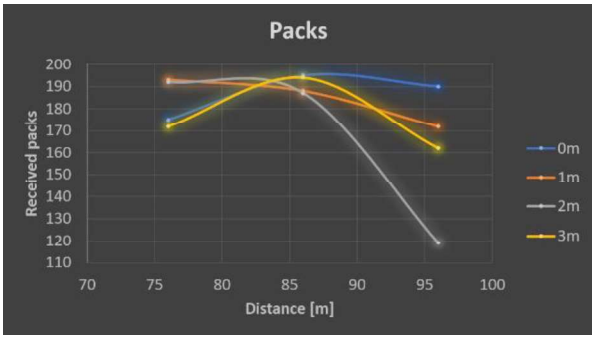

Figura 120: Paquetes recibidos en el dispositivo 1 bajo el protocolo DigiMesh durante el escenario P2M.

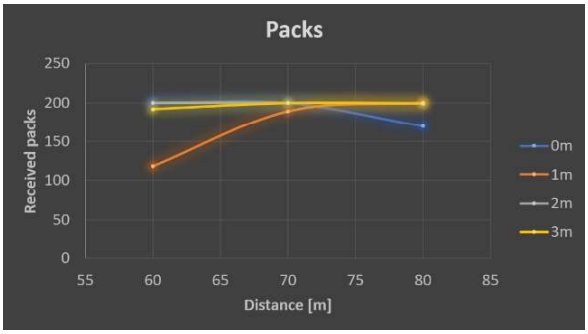

Figura 121: Paquetes recibidos en el dispositivo 2 bajo el protocolo Zigbee durante el escenario P2M.

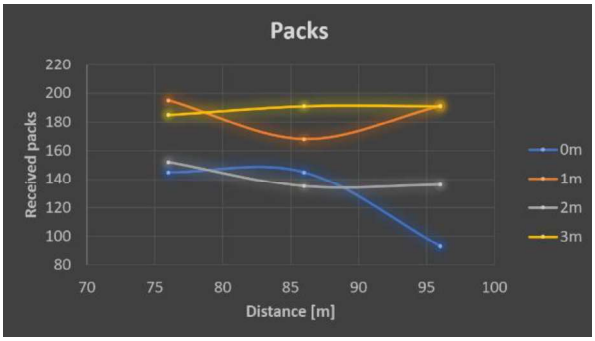

Figura 122: Paquetes recibidos en el dispositivo 2 bajo el protocolo DigiMesh durante el escenario P2M.

Aunado a las gráficas anteriores, la Figura 123 y Figura 124 muestran el consecuente PER, obtenido a partir de los paquetes recibidos en el dispositivo 1 durante las mediciones con los protocolos Zigbee y DigiMesh, respectivamente. La Figura 125 y la Figura 126 exponen el PER de los paquetes recibidos en el dispositivo 2 bajo los protocolos Zigbee y DigiMesh, con el mismo orden.

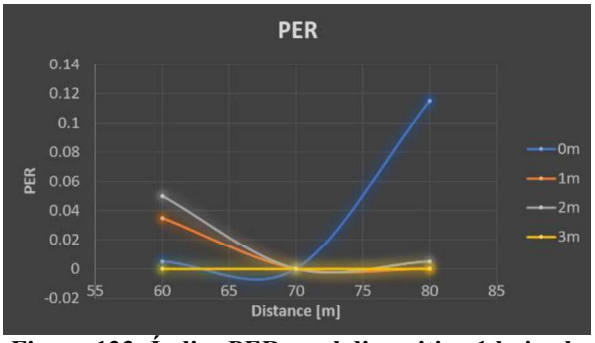

Figura 123: Índice PER en el dispositivo 1 bajo el protocolo Zigbee durante el escenario P2M.

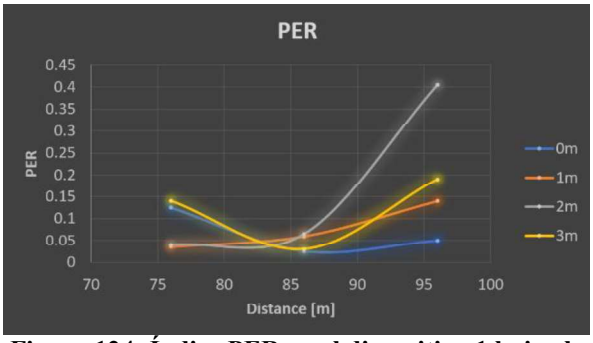

Figura 124: Índice PER en el dispositivo 1 bajo el protocolo DigiMesh durante el escenario P2M.

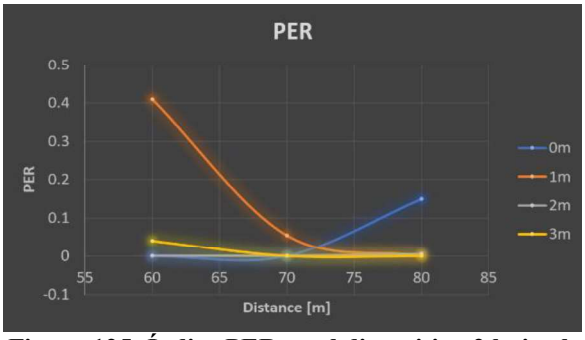

Figura 125: Índice PER en el dispositivo 2 bajo el protocolo Zigbee durante el escenario P2M.

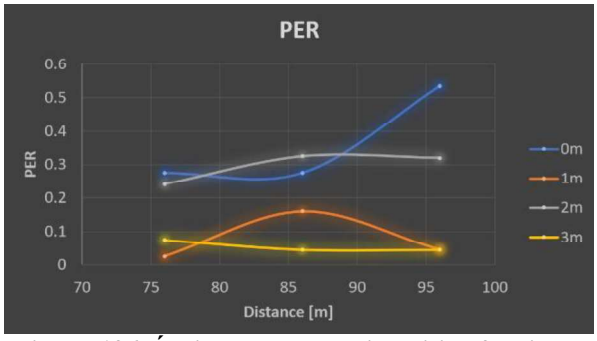

Figura 126: Índice PER en el dispositivo 2 bajo el protocolo DigiMesh durante el escenario P2M.

La Figura 119 y la Figura 121 muestran los resultados correspondientes a los paquetes recibidos por 2 de los 3 dispositivos empleados en las mediciones bajo el protocolo Zigbee (P1 y P2 a 80 metros de distancia en la Figura 118). P1 muestra decremento en la recepción de paquetes a los 0 metros de elevación y 80 metros de distancia, en conjunto con ligeros pérdidas de paquetes a 1 y 2 metros de elevación para los 60 metros de distancia. Estas pérdidas, en comparación con las del escenario P2P, son menores debido a que se empleó otro punto para los 80 metros y cuya altura es superior a la empleada para el escenario P2P (Figura 171, considerando un error por estar al borde de esa elevación, es probable que la elevación más precisa sea la mostrada en la Figura 173 para 96 metros de distancia, Apéndice A, página XVII).

De lado de DigiMesh, la Figura 120 y la Figura 122 muestras fuertes caídas al nivel del suelo, donde los paquetes llegan con errores en el header, por lo que el programa de recepción recibía mensajes como los mostrados en la Figura 78. De igual forma, P1 presenta mejores resultados en distancias posteriores, mientras que P2 tuvo problemas para mantener la información constante siendo probable que sea por un tunal encontrado que suponía un obstáculo para LOS (Figura 189) mismo que no presentó problemas con Zigbee, ya que la distancia reducida impidió que afectara la conexión.

Las siguientes gráficas muestran una comparativa en la sección de Delay, la Figura 127 y la Figura 128 muestran el retraso obtenido por los dispositivos 1 y 2 bajo el protocolo Zigbee, respectivamente. En la Figura 129 y en la Figura 130, los dispositivos muestran el retraso obtenido por los mismos dispositivos bajo el protocolo DigiMesh, siguiendo el mismo orden.

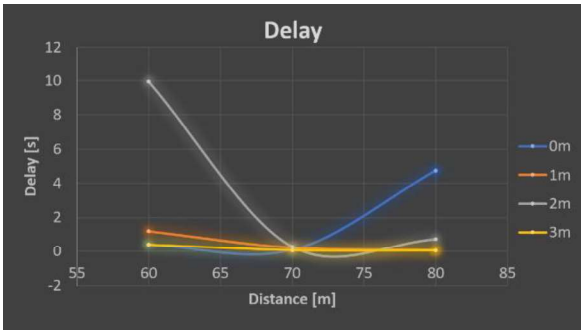

Figura 127: Retraso de los paquetes en el dispositivo 1 bajo el protocolo Zigbee durante el escenario P2M.

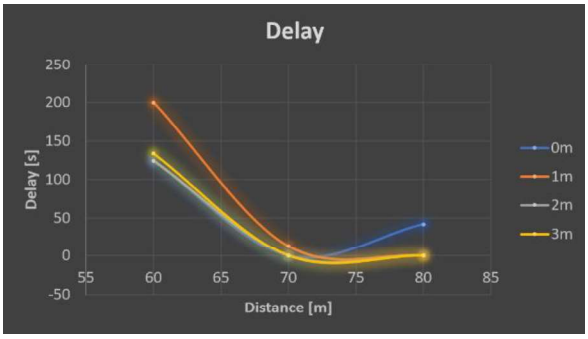

Figura 128: Retraso de los paquetes en el dispositivo 2 bajo el protocolo Zigbee durante el escenario P2M

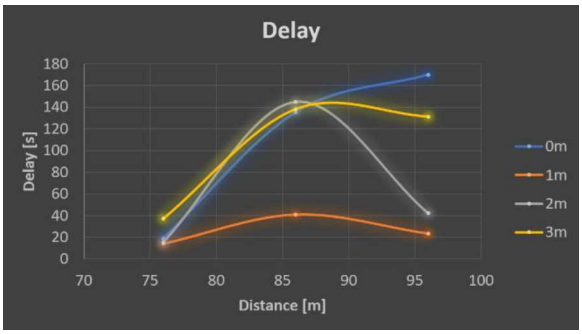

Figura 129: Retraso de los paquetes en el dispositivo 1 bajo el protocolo DigiMesh durante el escenario P2M..

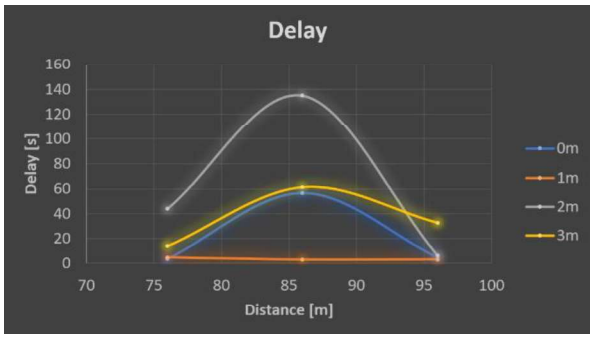

Figura 130: Retraso de los paquetes en el dispositivo 2 bajo el protocolo DigiMesh durante el escenario P2M.

Si bien, pareciera que la vegetación pudo no tener un gran impacto en la parte de paquetes recibidos bajo Zigbee, la Figura 127 y la Figura 128 muestran que P2 tuvo un problema de retraso muy significativo a los 60 metros, que también concuerda con la mayor pérdida de paquetes por parte del transmisor a 1 metro. Esto aclarara que existió la falta de conexión y que terminó en un corrimiento de información en las elevaciones consecuentes (2 y 3 metros). P1 muestra un ligero retraso a la misma distancia, sin embargo, estas pérdidas no reflejaron una inestabilidad muy fuerte en la comunicación, por lo que la parte abundante de vegetación. A pesar de no ser un obstáculo directo, representó una diferencia con respecto a la estabilidad en la comunicación.

El protocolo DigiMesh, en la Figura 129 y la Figura 130, presenta una inestabilidad impresionante por parte del dispositivo P1, donde si bien la perdida de paquetes no fue digna de mención en contra, los retrasos mostrados a 96 y a 76 metros de distancia indican

múltiples perdidas de señal. Teniendo, como consecuencia, posibles corrimientos de información en las elevaciones 0 y 3 metros para 0 metros de distancia y elevaciones 0, 1 y 3 metros para 10 metros de distancia; ambos en el dispositivo P1. El dispositivo P2 mostró también retrasos significativos, que; excluyendo a los 76 metros y elevación de 2 metros, es prácticamente imposible un corrimiento de información. Este comportamiento indica una relativa mayor estabilidad.

Los resultados en la sección de Throughput de las tablas aparecen en el siguiente compendio de gráficas. La Figura 131 y la Figura 132 exponen los resultados en los dispositivos 1 y 2, ambos bajo el protocolo Zigbee. Por su parte, la Figura 133 y la Figura 134 revelan los resultados en los dispositivos 1 y 2, ahora bajo el protocolo DigiMesh.

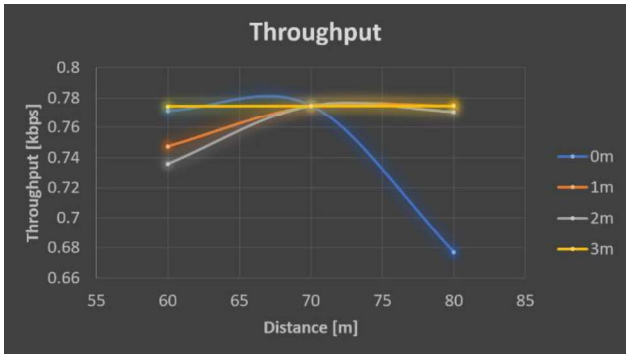

Figura 131: *Throughput* de los paquetes del dispositivo 1 bajo el protocolo Zigbee durante el escenario P2M.

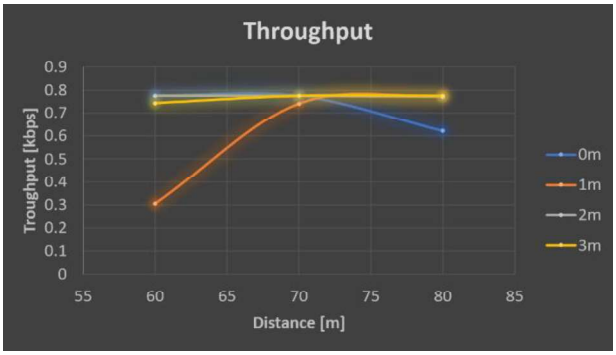

Figura 132: *Throughput* de los paquetes del dispositivo 2 bajo el protocolo Zigbee durante el escenario P2M..

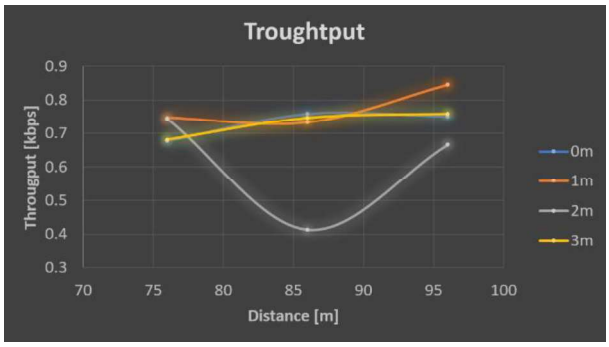

Figura 133: *Throughput* de los paquetes del dispositivo 1 bajo el protocolo DigiMesh durante el escenario P2M

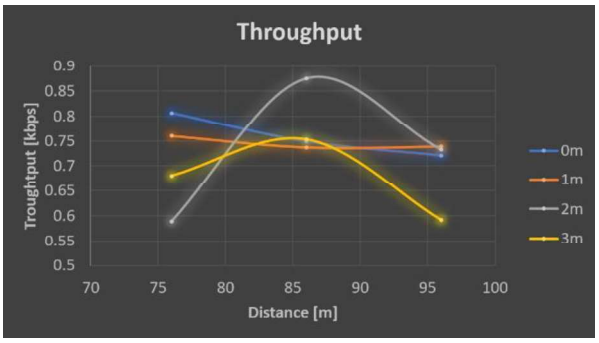

Figura 134: *Throughput* de los paquetes del dispositivo 2 bajo el protocolo DigiMesh durante el escenario P2M.

Los *troughputs* mostrados en la Figura 131 y Figura 132 revelan bajas en la capacidad de recepción no muy fuertes para el dispositivo P1 bajo el protocolo Zigbee. P2 en cambio, muestra no solo una ligera caída a la misma distancia que P1, si no también, una pérdida de información más destacable a los 60 metros de distancia. El resultado puede indicar una pérdida importante de información por desconexión a 1 metro de elevación, siendo que las siguientes elevaciones (2 y 3 metros) existe un evidente corrimiento de información.

La Figura 133 y la Figura 134 muestran corrimientos de información a distancias de 76 y 86 metros para el dispositivo P1 bajo el protocolo Zigbee, con elevaciones del transmisor a 0 y 3 metros y 0, 1 y 3 metros; respectivamente. Se observa en estas gráficas una constante pérdida de conexión a 0 metros de elevación y corrimientos debido a pérdida de información a 2 metros de elevación. El *throughput* en el dispositivo P2 bajo el protocolo DigiMesh aclara que las pérdidas de conexión fueron determinantes, o lo que es lo mismo, no hubo una buena reconexión una vez que la misma se perdía. También es palpable que el *lag* a los 10 metros de distancia con 2 metros de elevación terminó beneficiando la medición del *throughput* a esa distancia.

El siguiente compendio de 4 gráficas muestran los resultados en los tiempos de procesamiento en el receptor, La Figura 135 y la Figura 136 muestran esta sección en los dispositivos 1 y 2 bajo el protocolo Zigbee, mientras a que con el mismo orden, la Figura 137 y la Figura 138 muestran los resultados bajo el protocolo DigiMesh.

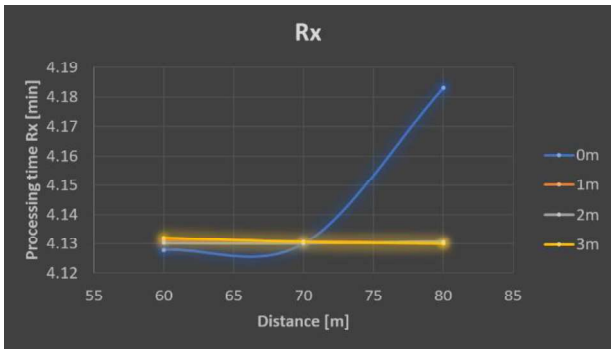

Figura 135: Tiempo de procesamiento en el receptor para el dispositivo 1 bajo el protocolo Zigbee durante el escenario P2M.

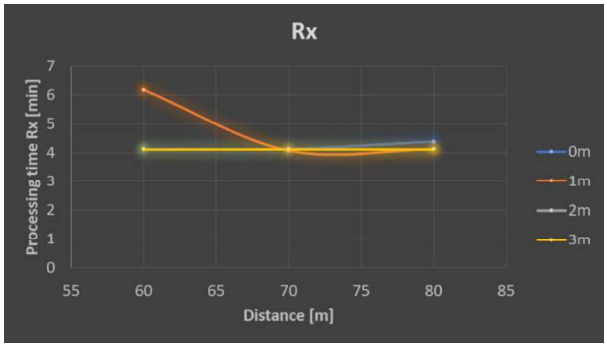

Figura 136: Tiempo de procesamiento en el receptor para el dispositivo 2 bajo el protocolo Zigbee. durante el escenario P2M.

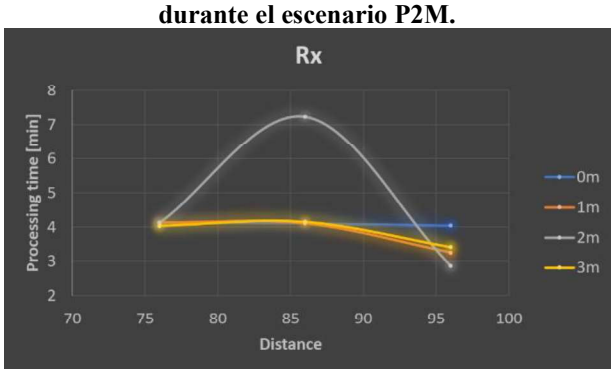

Figura 137: Tiempo de procesamiento en el receptor para el dispositivo 1 bajo el protocolo DigiMesh durante el escenario P2M.

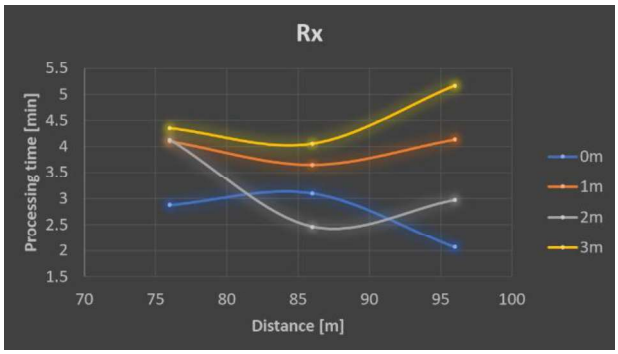

Figura 138: Tiempo de procesamiento en el receptor para el dispositivo 2 bajo el protocolo DigiMesh durante el escenario P2M.

Finalmente, las últimas 4 gráficas tienen la misma función que las anteriores. La Figura 139 y la Figura 140 para los dispositivos 1 y 2 la sección del procesamiento del tiempo en la transmisión. Por su parte, la Figura 141 y la Figura 142, presentan la misma sección para los mismos dispositivos, solo bajo el protocolo DigiMesh.

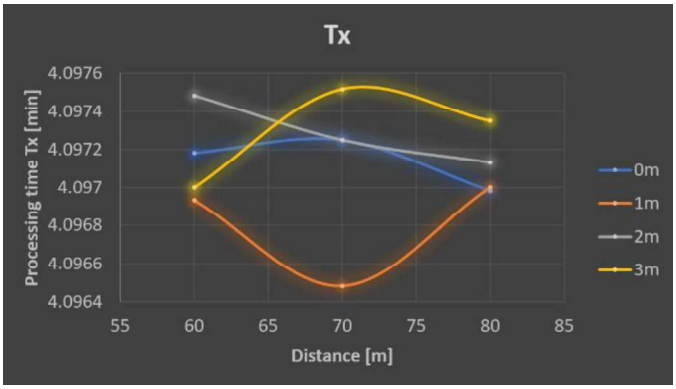

Figura 139: Tiempo de procesamiento en el transmisor 1 bajo el protocolo Zigbee durante el escenario P2M.

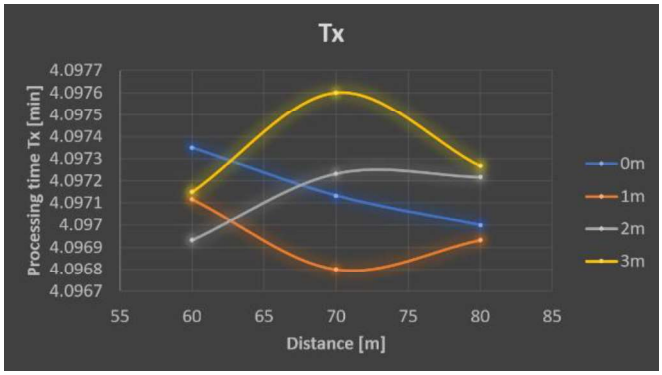

Figura 140: Tiempo de procesamiento en el transmisor 2 bajo el protocolo Zigbee durante el escenario P2M..

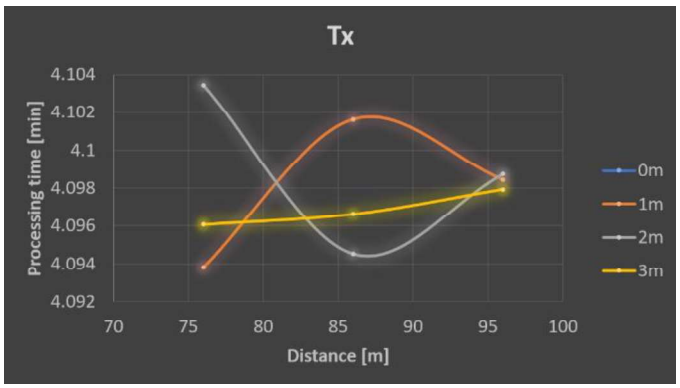

Figura 141: Tiempo de procesamiento en el transmisor 1 bajo el protocolo DigiMesh durante el escenario P2M

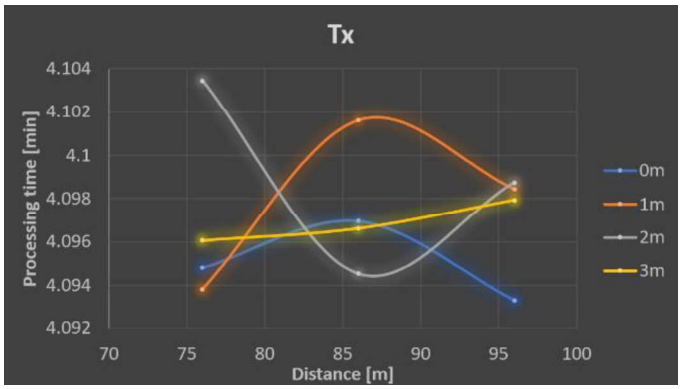

Figura 142: Tiempo de procesamiento en el transmisor 2 bajo el protocolo DigiMesh durante el escenario P2M.

En materia de tiempos de procesamiento en los receptores, la Figura 135 y la Figura 136 muestran gráficas para Zigbee, donde el dispositivo P1 no tiene retrasos en su procesamiento salvo a los 80 metros de distancia y elevación de 0 metros, mismo que concuerda con las de un PER superior en las perdidas. El dispositivo P2 no presenta retardos de renombre hasta los 60 metros donde a elevación de 1 metro existen retrasos en el procesamiento importantes que concuerdan con la caída de paquetes y los retrasos en la transmisión.

Para el protocolo DigiMesh, la Figura 137 y la Figura 138 revelan, en el dispositivo P1, tiempos concordantes con la perdida de paquetes y, a 86 metros de distancia, un retraso importante de información correspondiente a múltiples desconexiones con reconexiones exitosas. P2 muestra retardos consecutivos que, en conjunción con los datos del *troughput*, revelan una serie de reconexiones con *lags*, que terminan beneficiando la conexión en ciertos puntos; pero delatan la inestabilidad de la calidad del enlace.

Cabe destacar que los tiempos promedio aumentaron a 4 minutos aproximadamente en promedio debido que las mediciones fueron de 4 pruebas en cada elevación, teniendo un total de 800 mensajes mandados en 4 minutos. Los paquetes y el PER fueron normalizados a 200 paquetes para no discrepar de los límites de los escenarios anteriores.

Para el tiempo de procesamiento en la transmisión, la Figura 139 y la Figura 140 para Zigbee, al igual que la Figura 141 y la Figura 142 muestran variaciones ocasionadas por Arduino, las cuales no pueden ser controladas pero que, al ser de decenas de milisegundos, es posible posicionarlas como carentes de importancia.

6.4. Nodo Móvil (NM)

6.4.1. Puntos de medición

Con el receptor colocado en la máxima distancia aceptable (MDA) de los resultados en el escenario P2P y usando el terreno alto de la zona, el transmisor recorrió líneas perpendiculares a la recta imaginaría de la distancia entre el transmisor y el receptor. Debido a que la MDA a nivel del suelo tiene una diferencia considerable con respecto a la MDA con la antena en las alturas 1, 2 y 3 metros; 2 MDAs serán consideradas por cada protocolo.

La Figura 143 y la Figura 144 muestran los puntos de medición considerados, con la MDA a nivel del suelo para el protocolo Zigbee y el protocolo DigiMesh, respectivamente. La Figura 145 y la Figura 146 muestran los puntos de medición considerados, con la MDAs obtenidas con las antenas a las alturas de 1, 2 y 3 metros sobre el suelo; para el protocolo Zigbee y para el protocolo DigiMesh, respectivamente.

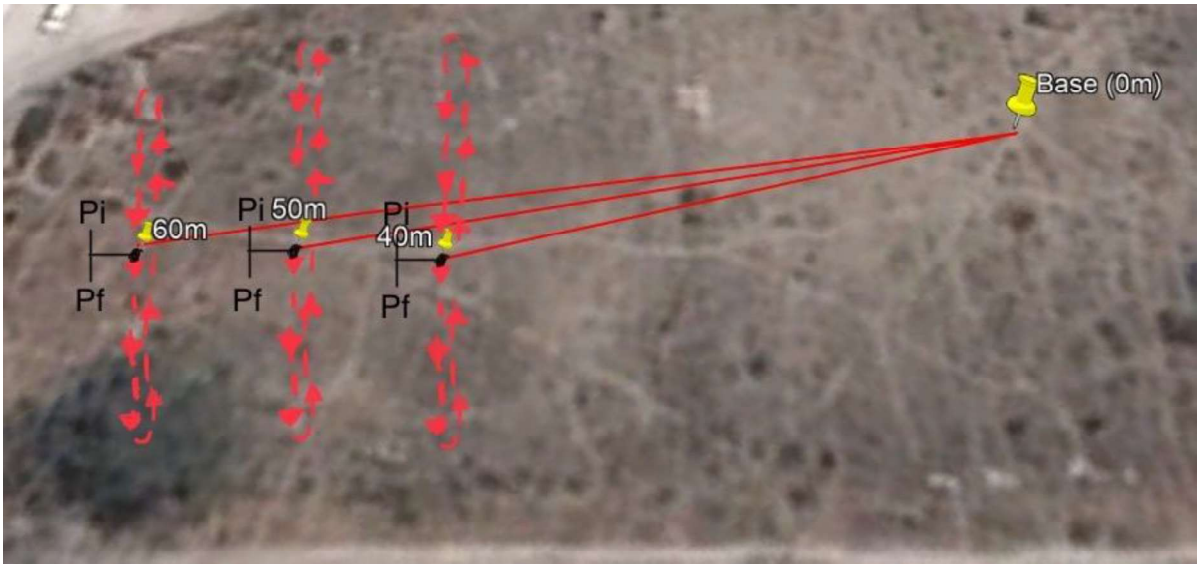

Figura 143: Puntos de mediciones usadas en el escenario NM, elevación 0 bajo el protocolo Zigbee.

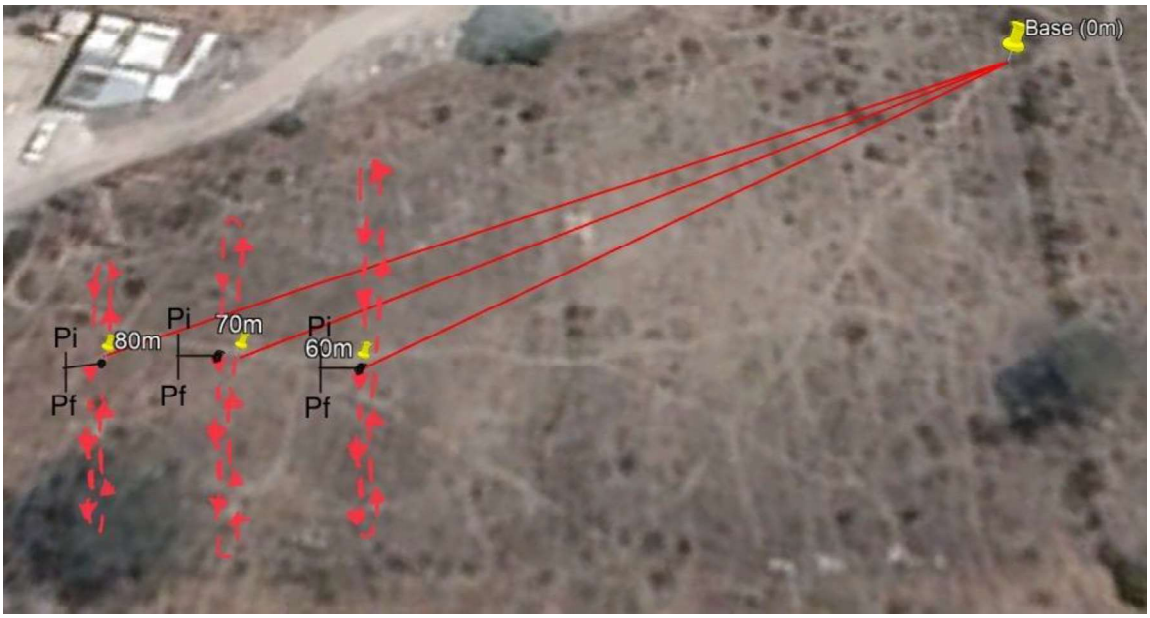

Figura 144: Puntos de mediciones usadas en el escenario NM, elevación 0 bajo el protocolo DigiMesh.

Figura 145: Puntos de mediciones usadas en el escenario NM, elevaciones 1, 2 y 3; bajo el protocolo Zigbee.

Figura 146: Puntos de mediciones usadas en el escenario NM, elevaciones 1, 2 y 3; bajo el protocolo DigiMesh.

La línea perpendicular es representada en estas figuras como una línea punteada, con flechas de color rojo que representar el movimiento del nodo. Las leyendas Pi y Pf, corresponden a los puntos de inicio y puntos finales recorridos por el transmisor, lo que conlleva a que el punto inicial es el mismo que el punto final.

El transmisor recorre, con velocidad constante, la línea punteada mostrada en las figuras. Esta línea punteada, tiene una longitud aproximada de 32 metros de longitud. Ya que el software de Arduino tiene una salida con el tiempo exacto en que el receptor recibió un paquete, es posible seccionar los 32 metros de la línea punteada en 8 secciones de 4 metros. De esta forma, conociendo el tiempo de llegada de un paquete, es posible estimar la posición aproximada del transmisor cuando envió dicho paquete y, usando gráficas de tipo histogramas, es posible observar el comportamiento de la comunicación entre el transmisor y el receptor en movimiento y poder analizar el desempeño de ambos protocolos con nodos en movimiento.

6.4.2. Resultados

La Tabla 19 y la Tabla 20 muestran los resultados obtenidos en el escenario NM bajo los protocolos Zigbee y DigiMesh respectivamente, cuyo escenario consta de mediciones con el receptor fijo a 60 metros (para una altura de la antena de 0 metros) y a 120 metros de distancia (para el resto de las alturas de la antena) en el caso del protocolo Zigbee. Para el caso del protocolo DigiMesh, el receptor permanece a 80 metros de distancia (para la altura de la antena de 0 metros) y a 140 metros de distancia (para resto de las alturas de la antena).

Los renglones azules superiores muestran las distancias, el primer renglón, contiene las distancias con el receptor en la MDA de las alturas de 1, 2 y 3 metros sobre el nivel del suelo; y, por lo tanto, tiene las distancias de mayor longitud. El segundo renglón, contiene las distancias usadas con el receptor en la MDA a nivel del suelo, por lo que este renglón cuenta con distancias de menor longitud. El orden de los renglones de las distancias es el mismo para ambas tablas.

En ambas tablas también, la columna izquierda azul muestra la información dividida en 7 secciones de interés:

- Paquetes recibidos, donde, posterior a la franja que muestra las elevaciones empleadas, cada renglón corresponde a las elevaciones en las que fueron tomadas y a la distancia indicada de separación respecto al transmisor.
- PER, el cual se obtuvo con el número de paquetes recibidos, la distribución de los datos es igual a la sección anterior.
- La sección del Throughput sigue el mismo formato de las secciones de Paquetes Recibidos y de PER.

- La sección del tiempo de procesamiento sigue el mismo formato que las secciones de Paquetes recibidos, PER y Throughput. En donde destaca que al ser imposible rescatar la información del transmisor, esta sección solo cuenta con la información del procesamiento en el receptor.
- El bloque de paquetes por sección fue elaborado dividiendo la línea perpendicular (mencionada en el desarrollo y mostrada en las rutas de este mismo escenario) de 32 metros en 8 secciones de 4 metros cada uno. Con el transmisor recorriendo la línea mencionada, el receptor recibiría un número determinado de paquetes en lo que el transmisor pasa por cada sección, lo que representa el estado de transmisión de información durante cada momento del trayecto.
    - La subsección denominada: "Altura de la antena a 0 m" muestra, como su nombre lo indica, los paquetes recibidos cuando la altura del receptor yace al nivel del suelo, por lo que sigue el renglón de las distancias menores.
    - La subsección llamada: "Distancia 100 metros" (para el protocolo Zigbee) o "Distancia 120" (para el protocolo DigiMesh), muestra los resultados a la distancia mencionada en el nombre de la sección. Estos resultados se apoyan en el renglón de las Alturas del receptor.
    - La subsección llamada: "Distancia 90 metros" (para el protocolo Zigbee) o "Distancia 110" (para el protocolo DigiMesh), muestra los resultados a la distancia mencionada en el nombre de la sección. Estos resultados se apoyan en el renglón de las Alturas del receptor.
    - Finalmente, la subsección llamada: "Distancia 80 metros" (para el protocolo Zigbee) o "Distancia 100" (para el protocolo DigiMesh), muestra los resultados a la distancia mencionada en el nombre de la sección. Estos resultados se apoyan en el renglón de las Alturas del receptor.

Tabla 19: Resultados de las mediciones en el escenario NM bajo Zigbee.

NM - ZB						
Distancia[m]				**100**	**90**	**80**
			60	**50**	**40**	
Paquetes Recibidos	**Altura del receptor [m]**	0m	164	177	200	
		1m		92	198	169
		2m		192	200	200
		3m		200	199	199
PER	**Altura del receptor [m]**	0m	0.18	0.115	0	
		1m		0.54	0.01	0.155
		2m		0.04	0	0
		3m		0	0.005	0.005
Tiempo de Procesamiento [min]	**Altura del receptor [m]**	**0m**	4.1273	4.1348	4.1300	
		1m		3.7817	4.1440	4.1306
		2m		4.1299	4.1302	4.1302
		3m		4.1303	4.0955	4.1305
Throughput [kbps]	**Altura del receptor [m]**	**0m**	0.6358	0.6849	0.7748	
		1m		0.3892	0.7645	0.6546
		2m		0.7438	0.7748	0.7748
		3m		0.7748	0.7774	0.7709

Altura del receptor para sección: Paquetes por sección				1m	2m	3m
Paquetes por sección	Altura del receptor a 0 m	30 -- 32	64	75	90	
		24 -- 28	68	84	107	
		20 -- 24	66	82	91	
		16 -- 20	83	61	92	
		0 -- 4	89	108	122	
		4 -- 8	101	84	91	
		8 -- 12	81	114	96	
		12 -- 16	104	103	111	
	Distancia 100 metros	30 -- 32		40	109	93
		24 -- 28		34	78	95
		20 -- 24		37	86	99
		16 -- 20		40	89	99
		0 -- 4		61	103	96
		4 -- 8		57	110	108
		8 -- 12		50	98	99
		12 -- 16		50	97	111
	Distancia 90 metros	30 -- 32		95	95	98
		24 -- 28		89	96	94
		20 -- 24		103	96	96
		16 -- 20		92	93	97
		0 -- 4		107	108	103
		4 -- 8		98	101	105
		8 -- 12		108	103	106
		12 -- 16		103	108	100
	Distancia 80 metros	30 -- 32		93	92	94
		24 -- 28		91	102	91
		20 -- 24		89	101	98
		16 -- 20		82	79	98
		0 -- 4		89	117	102
		4 -- 8		65	103	103
		8 -- 12		101	100	113
		12 -- 16		68	106	98

Tabla 20. Resultados de las mediciones en el escenario NM bajo DigiMesh.

NM - DM						
Distancia[m]				120	110	100
			80	70	60	
Paquetes Recibidos	Altura de la antena [m]	0m	149	191	197	
		1m		126	159	148
		2m		138	175	196
		3m		181	191	199
PER	Altura de la antena [m]	0m	0.255	0.045	0.015	
		1m		0.37	0.205	0.26
		2m		0.31	0.125	0.02
		3m		0.095	0.045	0.005
Tiempo de Procesamiento [min]	Elevaciones [m]	0m	4.1273	4.1348	4.1300	
		1m		3.7817	4.1440	4.1306
		2m		4.1299	4.1302	4.1302

		3m		4.1303	4.0955	4.1305
Throughput [kbps]	Elevaciones [m]	0m	0.6358	0.6849	0.7748	
		1m		0.3892	0.7645	0.6546
		2m		0.7438	0.7748	0.7748
		3m		0.7748	0.7774	0.7709
Altura de la antena para sección: Paquetes por sección				1m	2m	3m
Paquetes por sección	Altura de la antena a 0 m	30 -- 32	84	203	98	
		24 -- 28	123	86	68	
		20 -- 24	50	91	107	
		16 -- 20	94	78	72	
		0 -- 4	91	85	129	
		4 -- 8	51	98	131	
		8 -- 12	50	43	117	
		12 -- 16	53	82	66	
	Distancia 120 metros	30 -- 32		123	58	44
		24 -- 28		66	57	45
		20 -- 24		57	122	46
		16 -- 20		59	61	52
		0 -- 4		36	53	245
		4 -- 8		37	54	199
		8 -- 12		44	67	49
		12 -- 16		82	82	46
	Distancia 110 metros	30 -- 32		52	64	60
		24 -- 28		129	85	29
		20 -- 24		95	92	48
		16 -- 20		56	93	54
		0 -- 4		75	88	237
		4 -- 8		60	72	83
		8 -- 12		54	117	208
		12 -- 16		116	92	45
	Distancia 100 metros	30 -- 32		69	98	99
		24 -- 28		73	94	111
		20 -- 24		72	88	99
		16 -- 20		74	137	90
		0 -- 4		77	112	132
		4 -- 8		99	88	97
		8 -- 12		71	82	73
		12 -- 16		59	85	98

Las mediciones de este escenario fueron realizadas antes del periodo de lluvias, por lo que las distancias alcanzadas durante este escenario fueron mayores al escenario P2M, siendo estas: 60 y 80 metros para la elevación a 0 metros, junto con 120 y 140 con las alturas del receptor (1, 2 y 3 metros) para los protocolos Zigbee y DigiMesh, respectivamente.

Las siguientes figuras muestran 4 gráficas, la Figura 147 y Figura 148, las cuales representan los Paquetes recibidos bajo los protocolos Zigbee y DigiMesh,

respectivamente. Mientras que la Figura 149 y la Figura 150 muestran el consecuente PER de las gráficas anteriores, con el mismo orden.

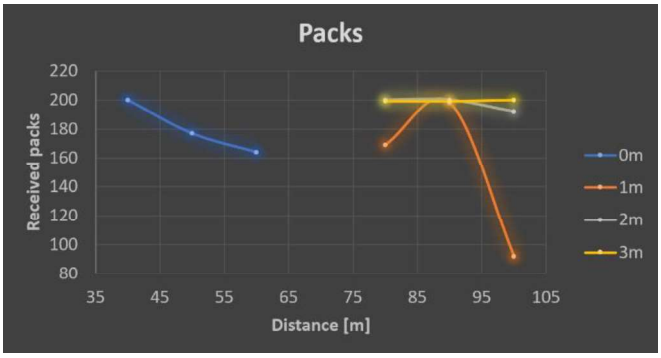

Figura 147: Paquetes recibidos bajo Zigbee durante el escenario NM.

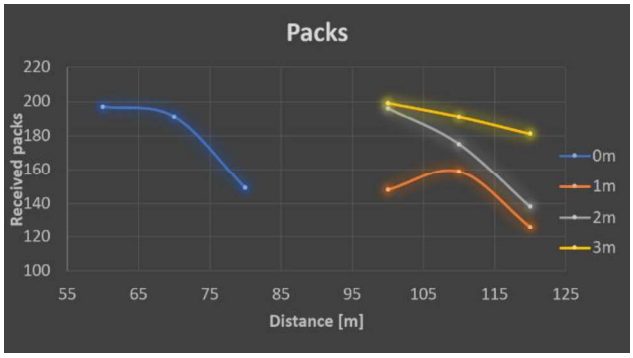

Figura 148: Paquetes recibidos bajo DigiMesh durante el escenario NM.

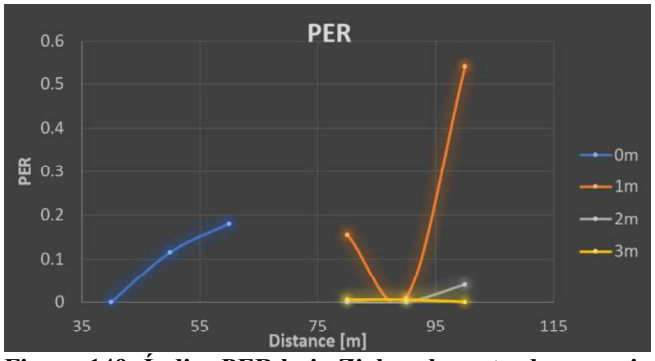

Figura 149: Índice PER bajo Zigbee durante el escenario NM.

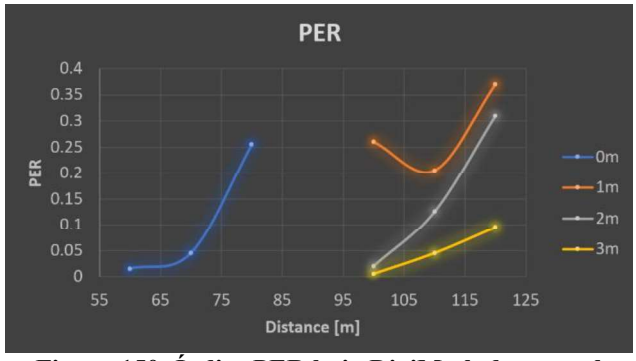

Figura 150: Índice PER bajo DigiMesh durante el escenario NM.

La Figura 147 y la Figura 149 muestran una caída de paquetes a los 80 metros con Zigbee, que muestra ser muy ligera en comparación con lo visto en el escenario P2P. Un evento sorpresivo, ocurre a los 100 metros, donde a 1 metro de elevación hubo una caía de más del 50% de los paquetes y en el cual no es posible intuir un posible retraso debido a la falta de tiempos por parte del transmisor. Aunado a que no hubo más indicios en los siguientes puntos de medición que revelaran algún patrón.

El protocolo DigiMesh presentó comportamientos similares, pero en los resultados hay un mayor número de paquetes perdidos en las siguientes elevaciones. Durante las mediciones fue imposible mantener completamente la elevación a 0 metros en el transmisor debido al movimiento necesario para estas mediciones, por lo que fue necesario considerar una elevación de 50 cm para que fuera posible este escenario. Esto puede ser un factor de mejora ante las mediciones a 0 metros. Sin embargo, persiste una pérdida de paquetes considerable, por lo que es posible que esto se deba a las pérdidas por el movimiento, mismo que sugeriría un problema mayor el mantener la conexión para el protocolo DigiMesh en comparación con el protocolo Zigbee.

La sección de Throughput es representada en las siguientes 2 gráficas, donde a la izquierda, la Figura 151 muestra el *throughput* obtenido con el protocolo Zigbee; y la Figura 152, revela los resultados para el protocolo DigiMesh.

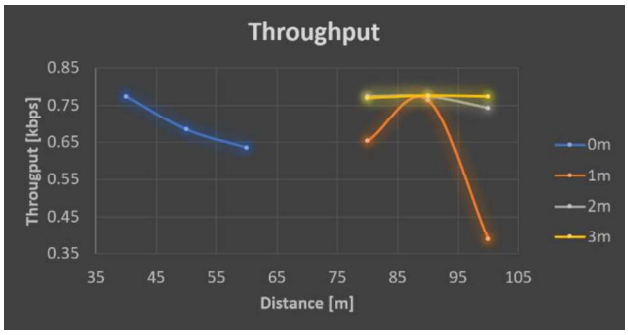

Figura 151: *Throughput* en Zigbee durante el escenario NM.

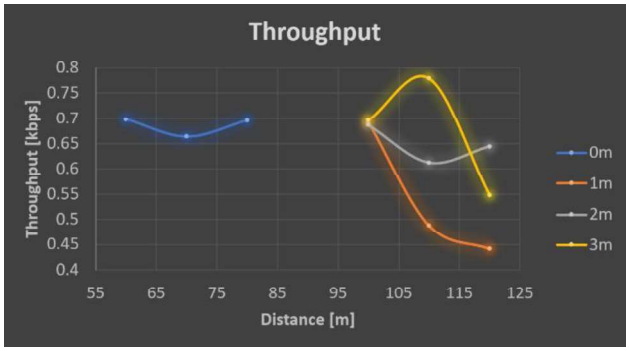

Figura 152: *Throughput* en DigiMesh durante el escenario NM.

En la Figura 151 y en la Figura 152, Zigbee muestra un comportamiento muy similar al de la perdida de los paquetes, lo que sugiere que no hubo retrasos muy importantes y, además, que en la mayoría de los paquetes perdidos hubo reconexiones sin muchos conflictos. Los retrasos mencionados en DigiMesh toman mayor sustento al ver el comportamiento del *throughput*, pues el efecto *lag* puede verse con un *throughput* mayor al esperado, mismo que denota y reafirma la inestabilidad en la conexión por parte de DigiMesh.

Finalmente, el siguiente compendio de gráficas muestra los tiempos de procesamiento en la recepción, siendo estos los únicos posibles de obtener debido a la naturaleza del escenario. La Figura 153 muestra los resultados bajo el protocolo Zigbee, mientras que la Figura 154 muestra los resultados ahora bajo el protocolo DigiMesh.

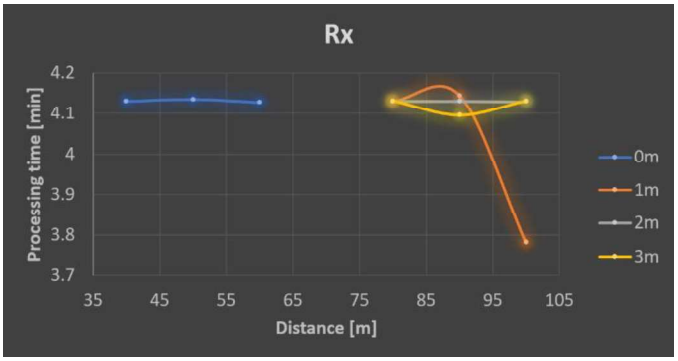

Figura 153: Tiempo de procesamiento en el receptor en Zigbee durante el escenario NM.

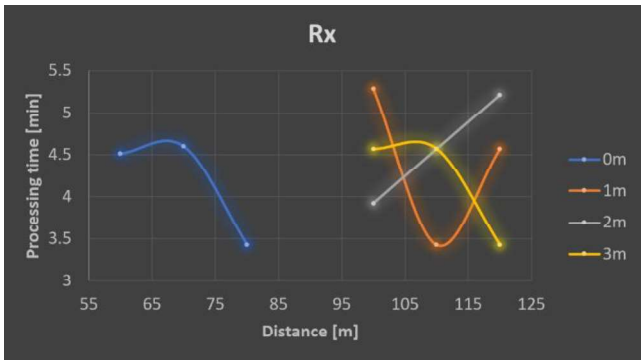

Figura 154: Tiempo de procesamiento en el receptor en DigiMesh durante el escenario NM.

El protocolo Zigbee no presenta muchas variaciones en la gráfica de la Figura 153, con la excepción del bajo tiempo de procesamiento a 100 metros de distancia y con la altura de la antena a 1 metro. Esta variación indica una caída de paquetes sin posible reconexión, lo mismo ocurre a 40 metros y 3 de elevación.

La Figura 154 demuestra, con mayor finalidad, la inestabilidad de los tiempos en DigiMesh, con tiempos de 5 minutos y otros cercanos a los 3 minutos, sabiendo que el promedio de tiempo de procesamiento debería estar alrededor de los 4.1 minutos. La inestabilidad presentada sugiere varios efectos *lag*, así como posibles retrasos importantes en la información.

Debido a que el interés inicial de las mediciones con la altura de la antena a 0 metros es probar el comportamiento a nivel del suelo, tomar pruebas posteriores con elevaciones superiores carece de sentido. La falta de mediciones a otras alturas es compensada con mediciones a diferentes distancias del receptor; teniendo, por consiguiente, gráficas que muestran un aproximado de paquetes que fueron recibidos mientras el transmisor se encontraba desplazándose por rectas imaginarias de una longitud promedio de 32 metros.

Con la finalidad de brindar una representación más clara del apartado de Paquetes por Sección, donde los paquetes son representados por medio de un Histograma. Estas representaciones muestran el número de paquetes obtenidos en las diferentes distancias. Teniendo en común la altura de la antena a nivel del suelo (0 metros). Las rectas de la trayectoria están divididas en 8 segmentos de 4 metros cada uno, comparándose con el número de paquetes aproximado recibido mientras el transmisor pasaba por cada uno de los segmentos.

Cabe destacar que todas las mediciones fueron realizadas con 4 pruebas por medición, siendo entonces un total de 800 paquetes por distancia en este caso y por elevación en gráficas posteriores, siendo normalizada a 200 paquetes al igual que en el escenario P2M. La Figura 155 muestra la gráfica para el protocolo Zigbee y, la Figura 156, para el protocolo DigiMesh.

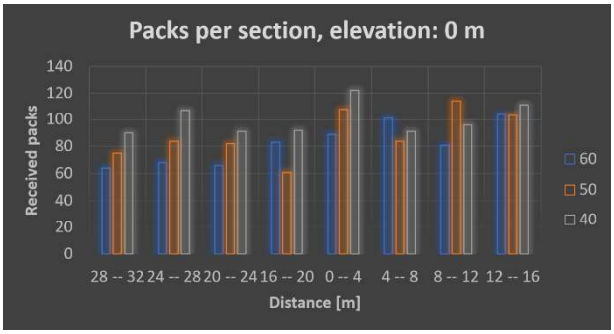

Figura 155: Paquetes por sección, elevación: 0 m en Zigbee durante el escenario NM.

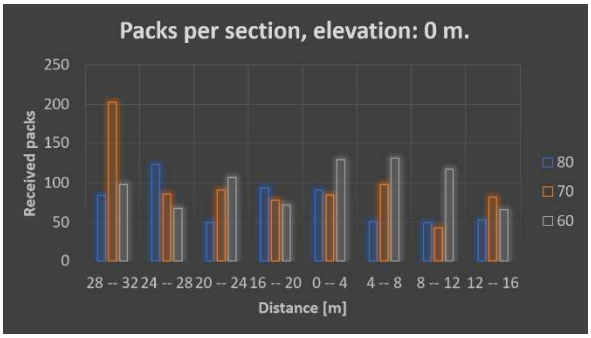

Figura 156: Paquetes por sección, elevación: 0 m en DigiMesh durante el escenario NM.

En la Figura 155, el protocolo Zigbee muestra tener numerosas perdidas a 60 metros, pero que mejoran paulatinamente a 50 y a 40 metros de distancia. Destacan varias perdidas en segmentos de 16 a 32 metros, donde la vegetación era un poco más densa a distancias 0 y 10 metros y donde resaltan el segmento de 12 a 16 metros, ya que no hubo muchas variaciones de paquetes perdidos.

En el protocolo DigiMesh existe, igual que en el protocolo Zigbee, un ligero decremento de paquetes en los segmentos de 8 a 16 metros, sin embargo, dado a los paquetes son muy irregulares a 0 metros, resulta complicado determinar si la vegetación influyo a una distancia de 80 metros. A 70 metros de distancia, un pico de más de 200 paquetes aparece en el segmento 28 a 32, lo que puede indicar un efecto de *lag*, mismo que causa que varios paquetes aparentemente son recibidos a la misma hora y, debido a varios de estos efectos, ese segmento puede que represente un buen punto de reconexión. En general, el protocolo Zigbee muestra mayor estabilidad con respecto al movimiento, sin embargo, es pertinente recalcar que la distancia entre radios es menor en comparación con el protocolo DigiMesh. Esto sugiere, que la estabilidad puede ser un gran impedimento a la hora de considerar grandes distancias de comunicaciones entre 2 dispositivos; sobre todo, con antenas a nivel del suelo.

Cambiando la forma de relacionar los histogramas, se muestra una representación donde ahora la relación es la distancia (dependiendo de cada protocolo), y las variaciones son respecto a las elevaciones tomadas en el mismo punto. La Figura 157 contiene los resultados bajo el protocolo Zigbee y la Figura 158 contiene los resultados para el protocolo DigiMesh.

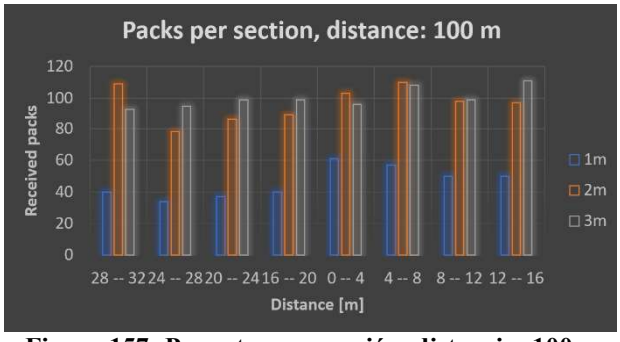

Figura 157: Paquetes por sección, distancia: 100m para Zigbee durante el escenario NM.

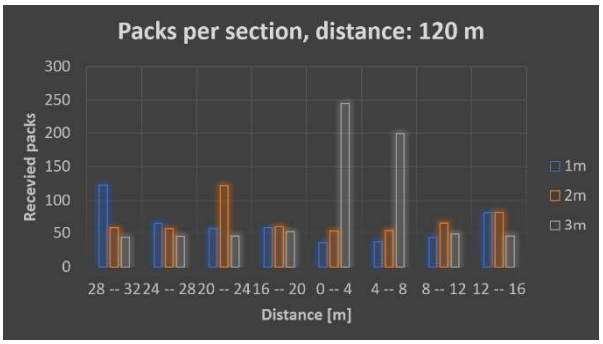

Figura 158: Paquetes por sección, distancia: 120 m para DigiMesh durante el escenario NM.

Las barras de la Figura 157 y la Figura 158 indican los paquetes recibidos con la antena en alturas de 1 a 3 metros, donde los receptores se hallan a 100 metros para el protocolo Zigbee y a 120 metros para el protocolo DigiMesh.

El protocolo Zigbee presenta una reducción en la recepción de paquetes muy notable durante las mediaciones con la antena a una altura de 1 metro, en comparación con las elevaciones subsecuentes. A pesar de que no hay un patrón reconocible, existe un

decremento mayor de paquetes en todos los segmentos, desde los 20 a los 32 metros, pero al no existir mucha diferencia, es complicado decir si la vegetación o el movimiento tuvieran un factor determinante; en las elevaciones consecuentes, la recepción de paquetes mejoró.

El protocolo DigiMesh, por su parte, mantiene cierta irregularidad en sus paquetes, con picos de entrega de paquetes que sugieren uno o varios efectos de *lag*, siendo un error tomarlos como puntos de "buena conexión" pues no son coherentes entre ellos. Esto reflejaría tener problemas importantes en la comunicación durante movimientos a largas distancias entre los disposisitivos.

De forma similar al esquema anterior, la Figura 159 y la Figura 160 muestran, los histogramas para el protocolo Zigbee y para el protocolo DigiMesh, respectivamente.

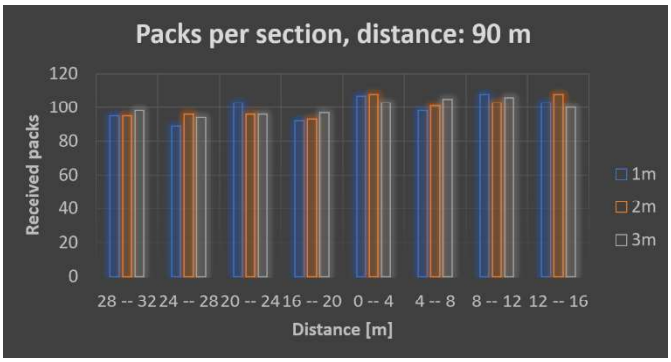

Figura 159: Paquetes por sección, distancia: 90 m para el protocolo Zigbee durante el escenario NM.

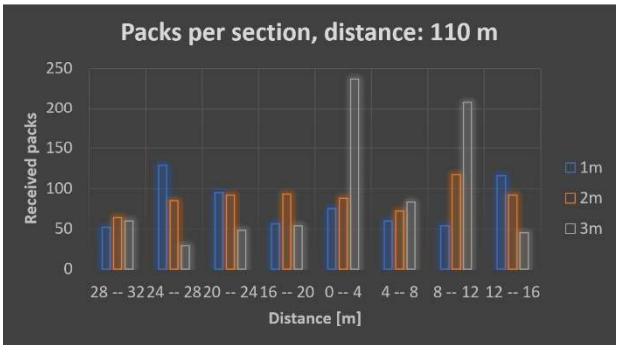

Figura 160: Paquetes por sección, distancia: 110 m para el protocolo DigiMesh durante el escenario NM.

La Figura 159 muestra que el protocolo Zigbee obtuvo una mejora significativa en todas las alturas y sin retrasos de ninguna índole. El protocolo DigiMesh también presenta una mejora, mostrado en la Figura 160, con el defecto de aun tener algunos paquetes perdidos y un *lag* muy marcado a los 3 metros de altura en la antena en segmentos de 0 a 4 metros y 8 a 12. Así como también, presenta una ligera reducción de paquetes recibidos con la antena a una altura de 1 metro, en los segmentos de 4 a 8 metros y 28 a 32 metros. La inestabilidad en la desconexión es visible a pesar de la mejora en los paquetes.

Terminando con la serie de histogramas de la sección Paquetes por sección, la Figura 161 y la Figura 162 muestran los histogramas correspondientes a una distancia de 80 metros para el protocolo Zigbee y una distancia de 100 metros para el protocolo DigiMesh, respectivamente.

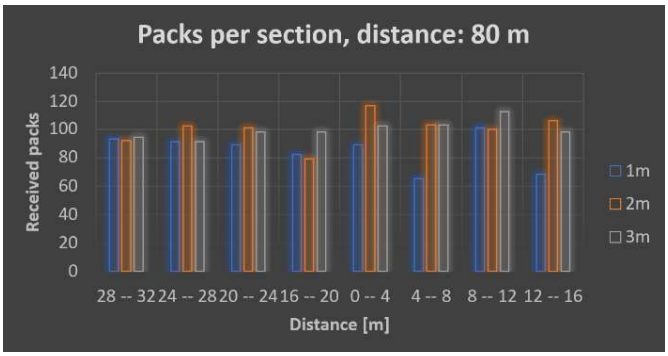

Figura 161: Paquetes por sección, distancia: 80 m para Zigbee durante el escenario NM.

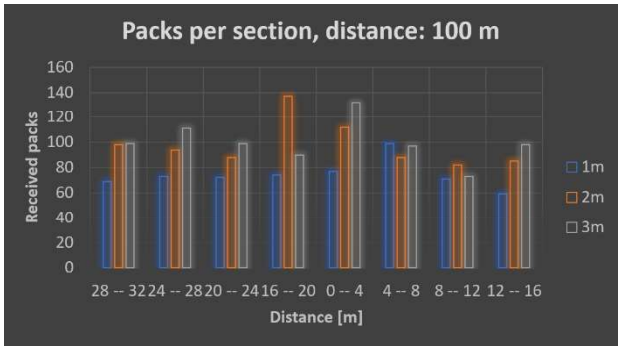

Figura 162: Paquetes por sección, distancia: 100 m para DigiMesh durante el escenario NM.

A 80 metros de distancia con el protocolo Zigbee, una ligera e inesperada caída de paquetes ocurre a 1 metro de elevación, donde en los segmentos 4 a 8 y 12 a 16 resultan aparecer las primeras caídas. Ya que el comportamiento es irregular, resulta complicado denotar alguna obstrucción en la señal específica, siendo más probable un problema con efecto de *lag*, por lo observado en el segmento 8 a 12.

Con DigiMesh, la mejora a elevaciones 2 y 3 metros es evidente y la inestabilidad baja un poco en todas las elevaciones, manteniendo un pico a 2 metros de elevación y en un segmento de 16 a 20 metros y otro a una altura de antena de 3 metros en el segmento de 0 a 4 metros; con la antena a 1 metro de altura, no se logró obtener todos los paquetes, aunque la inestabilidad en la conexión mejoró notablemente.

Capítulo 7. Conclusiones

7.1. Discusiones Finales

7.1.1. RSSI

Ocupar la herramienta proporcionada por el software XCTU tiene la ventaja de la simplicidad, cuyo precio es el cambiar las condiciones de las mediciones con respecto al uso del Arduino, lo cual es permisible considerando el propósito de dicha medición.

Durante este escenario, ocurre una ventaja preponderante de parte de DigiMesh sobre Zigbee en materia de alcance con una diferencia corta de 20 metros. DigiMesh mostró mayor estabilidad ante las irregularidades en el suelo y con vegetación como arbustos, matorrales y pasto. Zigbee, por su parte, mostró una pérdida de potencia más determinante, sin embargo, considerando las frecuencias a las que operan (2.4 GHz para Zigbee y 915 MHz para DigiMesh) es predecible la ventaja en el alcance de DigiMesh. Este último, demostró que el alcance podría ser todavía mayor, sin embargo, la vegetación más densa como tunales o arboles de la zona producen una interferencia muy complicada de superar. Finalmente, hay que considerar que hacer el test con la herramienta ocasiona una diferencia muy notable ya que no hay reintentos de mensaje.

7.1.2. Punto a Punto (P2P)

La existencia de los reintentos de transmisión dio pie, por parte de ambos protocolos, de aumentar el rango de alcance de la transmisión. Si bien, DigiMesh vuelve a destacar con un mayor alcance, lo cierto es que también presenta mayor inestabilidad en la conexión, con retrasos muy notables en la información. Lo más notable en DigiMesh es la pérdida de información a nivel del suelo, misma que no fue considerada a distancias posteriores a los 100 metros debido a la constante elevación del terreno. Esta elevación, indica mejores valores que podrían mal interpretarse en caso de implementar las comunicaciones con estos dispositivos.

El Arduino influyó negativamente, tanto con los errores de la transmisión de la Figura 77, las variaciones en la transmisión al igual que su participación en los efectos de *lag* y, por tanto, en los retrasos.

El rango, bajo el protocolo Zigbee, fue igual al visto en el artículo de Khalifeh [15], lo cual hace sentido al saber que el radio era de la misma serie que el empleado. Sin embargo, DigiMesh mejoró considerablemente con respecto a lo analizado en el mismo artículo, lo que puede atribuírsele al uso de una serie diferente de radios usado, aunado al uso del protocolo en la frecuencia de los 915 MHz. En materia del *throughput*, el trabajo no buscaba una transmisión de muchos datos; pero es palpable que DigiMesh, a pesar de su mayor rango, presenta un *thoughput* más inestable y con una tendencia a disminuir después de los 100 metros.

7.1.3. Salto (RP)

El uso de un repetidor mostró un predecible aumento en el rango de alcance dentro de las limitaciones del terreno. Si bien, es indiscutible el buen rango por parte de los dispositivos, es incorrecto exponer algún valor específico, pues el constante escalonado ascendente de los terrenos juega un papel fundamental para las mediciones. Lo mostrado, considerando la altura, denota de nuevo una ventaja por parte del protocolo DigiMesh, cuyo comportamiento fue más estable e incluso dio indicios de que los a 270 metros de distancia no era un límite. El protocolo Zigbee mostró tener mayores problemas, donde es probable que la vegetación de la zona realmente es una oposición a la estabilidad en la conexión. Ocupar un repetidor sin Arduino proporcionó mejores resultados en algunos puntos que incluso en el escenario P2P, lo que supone una mayor intervención con el procesamiento de Arduino de lo esperado, quizá por la interacción con la computadora al mostrar los datos en el monitor. Luego, lo visto en la Figura 77, aclara una cierta limitación por parte de Arduino, sin mencionar la capacidad para mandar los mensajes con menor tiempo de 300 ms.

7.1.4. Punto a multipunto (P2M)

Si bien el tiempo de espera, a la hora de mandar mensajes, por parte de Arduino era de 1 100 ms por paquete en conexión P2P con algunos pocos errores; a la hora de tratar con P2M, los errores eran tales que las mediciones a 0 metros de distancia entre radios no tenían sentido en cuanto a paquetes perdidos. Para que las mediciones no se vieran completamente afectadas, el tiempo de espera entre mensajes programado en Arduino fue aumentado a 300 ms, donde no presentó problemas en ninguna prueba inicial (antes de realizar las mediciones competentes a este trabajo). Dentro de esta consideración, junto con el incremento de la vegetación y sin posibilidad de conocer la información del tercer transmisor; DigiMesh volvió a demostrar mayor alcance con mayores problemas de inestabilidad en la conexión. Aun así, ambos protocolos mostraron muchos problemas con

la conexión ante el incremento de pasto, arbustos y con la humedad. Zigbee tuvo resultados más estables, sin embargo, la posición de los dispositivos fue ventajosa por sobre los de DigiMesh al estar al borde de donde se hallaba toda la vegetación. Es considerable la diferencia de estabilidad en relación con los 16 metros de alcance extra que tenía DigiMesh.

El escenario mostró de igual forma, la ventaja que adquieren las conexiones tras elevar los radios, pues si bien, en el escenario P2P, ambos protocolos quedaban sin conexión a los pocos metros. Tras aumentar su altura con el escalonado, el aumento de rango fue considerable, siendo complicado determinar un límite fijo para alguna implementación, ya que dependerá en gran medida del medio en el que se encuentre el sistema de telecomunicaciones al ser implementado.

7.1.5. Nodo Móvil (NM)

Durante las mediciones a nivel del suelo, el transmisor estaba en contante movimiento a una distancia superior a los 30 cm. Este suceso eliminó muchas de las interferencias ocasionadas por las irregularidades del suelo y parte del pasto, con lo que los resultados de la conexión fueron visiblemente mejor para el protocolo Zigbee que en sus escenarios pasados. El protocolo DigiMesh mantiene constante su ventaja de distancia, sin embargo, el protocolo Zigbee supone tener mayor robustez durante el movimiento. El protocolo DigiMesh tendría que reducir su ventaja de alcance frente al protocolo Zigbee para que los resultados con el protocolo DigiMesh recuperen prosperidad. Por lo mismo, las gráficas del protocolo Zigbee distinguen un patrón de interferencia orientado a una susceptibilidad muy alta ante la vegetación. La altura también juega un papel fundamental por la inclinación en beneficio a los dispositivos pues dificulta los obstáculos durante LOS, misma que aumenta al considerar la elevación por parte del transmisor.

7.2. Trabajo futuro

El trabajo contó con limitantes en diversos aspectos, siendo el primero, el uso de Arduino Leonardo, pues al ser un dispositivo orientado a la educación, es probable que el microcontrolador y el reloj de 16 MHz sea de baja gama para comunicaciones que buscan un constante y elevado *throughput*. Sin embargo, no se puede descartar los retrasos ocurridos de forma natural, debido a la comunicación serial con la computadora. Los problemas vistos en las gráficas de Delay van intrínsicamente ligados con el *lag* ocasionado por la placa de Arduino. Es recomendado, entonces, utilizar placas de mayor capacidad de procesamiento para implementaciones más profesionales, como los Raspberry Pi o superiores.

Las condiciones del terreno limitaron en gran medida la capacidad de alcance máximo de las mediciones para el escenario RP, al igual que la separación entre dispositivos para el escenario P2M y rectas cortas para el escenario NM. Una ubicación con mayores dimensiones brindaría más información acerca del comportamiento de los radios ante los escenarios ya mencionados.

En este trabajo fueron ejercidos diferentes escenarios individuales donde los radios, bajo diferentes protocolos, mostraron su desempeño y capacidad de transmisión ante condiciones rurales, los cuales muy rara vez son sinónimos de la palabra "ideal". El trabajo no abarcó el medir el comportamiento de los radios y protocolos bajo la unión de los diversos escenarios, donde redes de dispositivos enfrenten la jerarquía de sus nodos ya sea estáticamente o con movimiento; donde los dispositivos están forzados a usar múltiples conexiones y saltos para la gestión de la información. En estas redes tipo “Mesh”, es posible agregar, además, otros factores necesarios para la comunicación, como la encriptación o la capacidad de los dispositivos para "dormir"; lo que tampoco fue abarcado durante el estudio.

7.3. Conclusiones finales

A pesar de las particularidades de Arduino y las limitaciones por el terreno, los dispositivos y protocolos presentaron ciertos comportamientos que se repetían constantemente. Si bien no es errado pensar que estos protocolos están pensados para un entorno más urbano y en distancias cortas como hogares y empresas, lo cierto es que cada uno mostró tener ventaja ante ciertas situaciones.

El protocolo Zigbee demostró dificultades para los obstáculos a nivel del suelo y vegetación de casi todo tipo, con marcadas ventajas en conexiones punto multipunto y para conexiones con dispositivos en movimiento. Por su parte, el protocolo DigiMesh presentó menores problemas ante obstáculos a nivel del suelo (a una distancia limitada, pero mayor al protocolo Zigbee) y vegetación no muy densa como pasto o arbustos, presentando mayores problemas con árboles y tunales. Así como también, ligeras inestabilidades en conexiones punto multipunto y fue bastante más inestable ante movimiento en los dispositivos. Existe una ventaja teórica del protocolo DigiMesh con respecto al protocolo Zigbee, que no formó parte del alcance de este trabajo; pero resulta recalcable, la cual es que la jerarquía del protocolo DigiMesh da pie a que casi todos los dispositivos entren en modo "sueño", lo que resulta importante cuando la forma de energizar a estos radios resulta limitada.

Si bien, la innovación en las telecomunicaciones esta inclinada al uso masivo en información con tecnologías como 5G, *streaming* y IoT; lo cierto es que sus primeras implementaciones tienen un mercado orientado a lo urbano. Quizá tome algo de tiempo para realización de tecnologías que se concentren en redes independientes en entornos

rurales con beneficios en gestión de cultivos, o prevenir a comunidades de riesgos de cualquier índole. Este trabajo, entonces, presenta carencias que denotan futuros nichos para mercados u operaciones gubernamentales, buscando mantener un mundo más conectado y la preponderancia de la seguridad a medida que la tecnología mejore, siempre en consideración del beneficio de los seres vivos.

Capítulo 8. Referencias

[1] The Competitive Intelligence Unit, "Telecomunicaciones Móviles al 2T-2021: En Trayectoria de Plena Recuperación," *The CIU*, Aug. 02, 2021. https://www.theciu.com/publicaciones-2/2021/8/2/telecomunicaciones-mviles-al-2t-2021-en-trayectoria-de-plena-recuperacin (accessed Aug. 24, 2021).

[2] INEGI, "TIC's en hogares," 2018. https://www.inegi.org.mx/temas/ticshogares/ (accessed Jan. 16, 2020).

[3] Oxford, "wearable_1 noun - Definition, pictures, pronunciation and usage notes | Oxford Advanced Learner's Dictionary at OxfordLearnersDictionaries.com." https://www.oxfordlearnersdictionaries.com/us/definition/english/wearable_1?q=Wearable (accessed Jan. 16, 2020).

[4] Expanción, "El consumo de wearables crece en México," 2016. https://expansion.mx/tecnologia/2016/07/05/el-consumo-de-wearables-crece-en-mexico (accessed Jan. 16, 2020).

[5] Unión Internacional de las Telcomunicaciones., *Manual de telecomunicaciones de emergencia.* Ginebra: UIT, 2005.

[6] U. I. de T. ITU, "Descripción General de Internet de los Objetos Y.2060- Y.4000," *Y.2060 Y.4000*, p. 20, 2016, [Online]. Available: http://handle.itu.int/11.1002/1000/11559.

[7] M. Rouse, "What is internet of things (IoT)? - Definition from WhatIs.com," 2016. https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT (accessed Jan. 16, 2020).

[8] The Carnegie Mellon University Computer Science Deparment Coke Machine, "The 'Only' Coke Machine on the Internet." https://www.cs.cmu.edu/~coke/history_long.txt (accessed Jan. 16, 2020).

[9] R. S. Raji, "Smart networks for control," *IEEE Spectr.*, vol. 31, no. 6, pp. 49–55, Jun. 1994, doi: 10.1109/6.284793.

[10] Jason Pontin, "ETC: Bill Joy's Six Webs - MIT Technology Review," 2005. https://www.technologyreview.com/s/404694/etc-bill-joys-six-webs/ (accessed Jan. 16, 2020).

[11] K. Ashton, "In the real world, things matter more than ideas. That 'Internet of Things' Thing," 2009. Accessed: Jan. 16, 2020. [Online]. Available: http://www.rfidjournal.com/articles/view?4986.

[12] Y. Omura, Abhijit Mallik, and N. Matsuo, *MOS Devices for Low-Voltage and Low-Energy Applications*. Singapore: IEEE PRESS, 2017.

[13] J. Steward, "The Ultimate List of Internet of Things Statistics for 2021," *findstack*, 2021. https://findstack.com/internet-of-things-statistics/.

[14] X. Li and X. Lu, “Design of a ZigBee wireless sensor network node for aquaculture monitoring,” *2016 2nd IEEE Int. Conf. Comput. Commun. ICCC 2016 - Proc.*, pp. 2179–2182, 2017, doi: 10.1109/CompComm.2016.7925086.

[15] A. Khalifeh, H. Salah, S. Alouneh, A. Al-Assaf, and K. Darabkh, “Performance Evaluation of DigiMesh and ZigBee Wireless Mesh Networks,” *2018 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2018*. 2018, doi: 10.1109/WiSPNET.2018.8538620.

[16] I. A. Zualkernan, F. A. Aloul, V. Sakkia, H. Al Noman, S. Sowdagar, and O. Al Hammadi, “An IoT-based emergency evacuation system,” *Proc. - 2019 IEEE Int. Conf. Internet Things Intell. Syst. IoTaIS 2019*, pp. 62–66, 2019, doi: 10.1109/IoTaIS47347.2019.8980381.

[17] H. Yuliandoko and A. Rohman, “Flooding detection system based on water monitoring and zigbee mesh protocol,” *2019 4th Int. Conf. Inf. Technol. Inf. Syst. Electr. Eng. ICITISEE 2019*, vol. 6, pp. 385–390, 2019, doi: 10.1109/ICITISEE48480.2019.9003928.

[18] EWIN, “EWIN,” 2020. http://ewin.corporativostr.com/ (accessed Jan. 23, 2020).

[19] IEEE, *802.15.4-2020 - IEEE Standard for Low-Rate Wireless Networks*. IEEE, 2020.

[20] Zigbee Alliance, “Zigbee Specification,” 2015. Accessed: Jun. 28, 2021. [Online]. Available: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf.

[21] Digi International, “About Digi,” 2020. https://www.digi.com/about-digi (accessed Mar. 09, 2020).

[22] Digi International, “XBee®-PRO 900HP/XSC RF Modules S3 and S3B User Guide,” 2020. Accessed: Mar. 09, 2020. [Online]. Available: www.digi.com/howtobuy/terms.

[23] Arduino, “What is Arduino? - Introduction,” 2020. https://www.arduino.cc/en/guide/introduction (accessed Mar. 09, 2020).

[24] “Arduino Leonardo with Headers | Arduino Official Store.” https://store.arduino.cc/usa/leonardo (accessed Jun. 11, 2020).

[25] Mercado libre, “Digi Xbee Peo Digimesh S3b 900mhz Antena Ufi,” 2021. https://articulo.mercadolibre.com.mx/MLM-853410382-digi-xbee-pro-digimesh-s3b-900mhz-antena-ufl-_JM?matt_tool=13246067&matt_word=&matt_source=google&matt_campaign_id=15698047819&matt_ad_group_id=134599571754&matt_match_type=&matt_network=g&matt_device=c&m.

[26] Digi International, “XBee®/XBee-PRO S2C Zigbee® RF Module User Guide,” 2020. Accessed: Jun. 10, 2020. [Online]. Available: www.digi.com/howtobuy/terms.

[27] “Firmware Definition.” https://techterms.com/definition/firmware (accessed Jun. 09, 2020).

[28] buyhatke, “Digi XBee Module S2C 802.15.4 2mW with Wire Antenna XB24CZ7WIT-004,” 2021. Digi XBee Module S2C 802.15.4 2mW with Wire Antenna XB24CZ7WIT-004.

[29] Digi Internationals, “XBee & XBee-PRO OEM RF Module Antenna Considerations,” 2012. Accessed: Jun. 14, 2020. [Online]. Available: www.digi.com.

[30] Taoglass, “SPECIFICATION Patent Granted FXP74 Black Diamond 2.4GHz Band Antenna,” 2020.

[31] Taoglas, “FXP290.07.0100A,” 2020.

[32] Digi International, “XCTU Configuration and Test Utility Software User Guide,” 2020. Accessed: Jun. 28, 2020. [Online]. Available: www.digi.com/howtobuy/terms.

Apéndice A:

Este apéndice muestra todas las elevaciones del terreno, mostrado por la herramienta Google Earth.

Figura 163: Elevación en la ruta de 0 a 10 metros.

Figura 164: Elevación en la ruta de 0 a 20 metros.

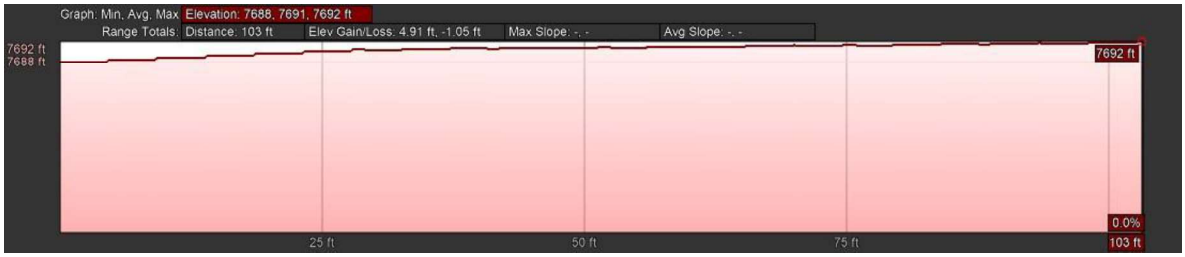

Figura 165: Elevación en la ruta de 0 a 30 metros.

Figura 166: Elevación en la ruta de 0 a 40 metros.

Figura 167: Elevación en la ruta de 0 a 50 metros.

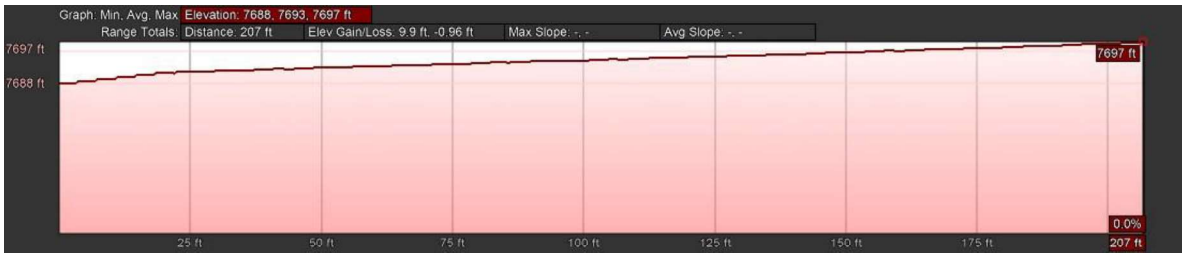

Figura 168: Elevación en la ruta de 0 a 60 metros.

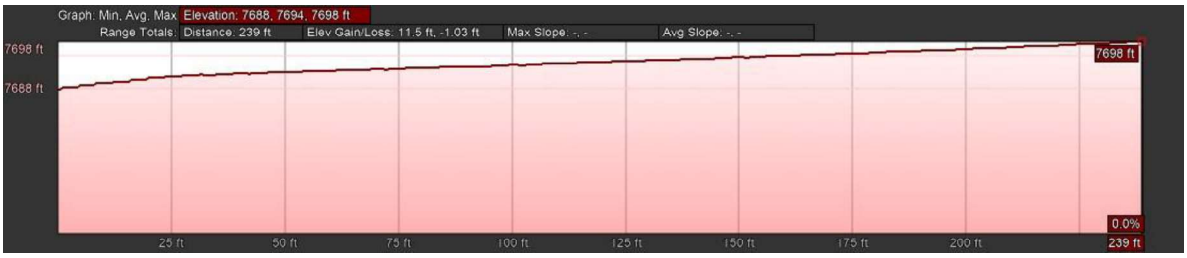

Figura 169: Elevación en la ruta de 0 a 70 metros.

Figura 170: Elevación en la ruta de 0 a 80 metros (Punto usado en los escenarios RSSI y P2P).

Figura 171: Elevación en la ruta de 0 a 80 metros (punto usado en los escenarios RP, P2M Y NM).

Figura 172: Elevación en la ruta de 0 a 90 metros.

Figura 173: Elevación en la ruta de 0 a 96 metros.

Figura 174: Elevación en la ruta de 0 a 100 metros.

Figura 175: Elevación en la ruta de 0 a 120 metros.

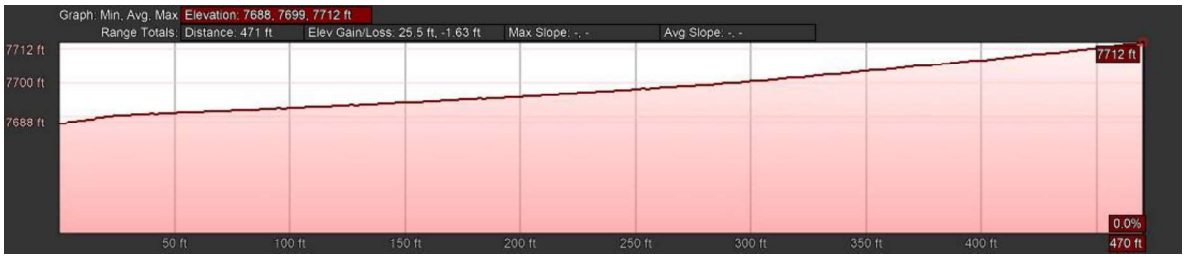

Figura 176: Elevación en la ruta de 0 a 140 metros.

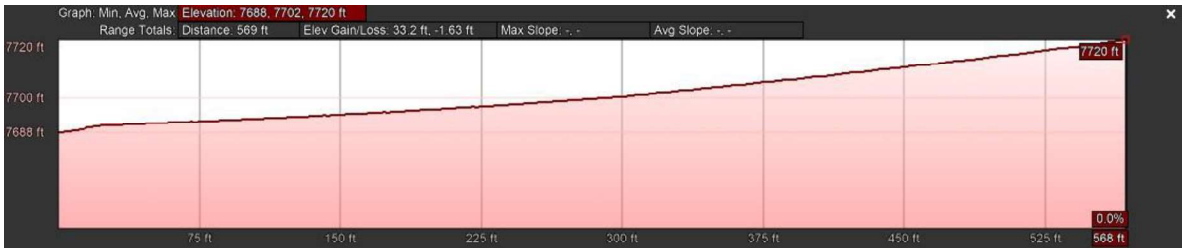

Figura 177: Elevación en la ruta de 0 a 170 metros.

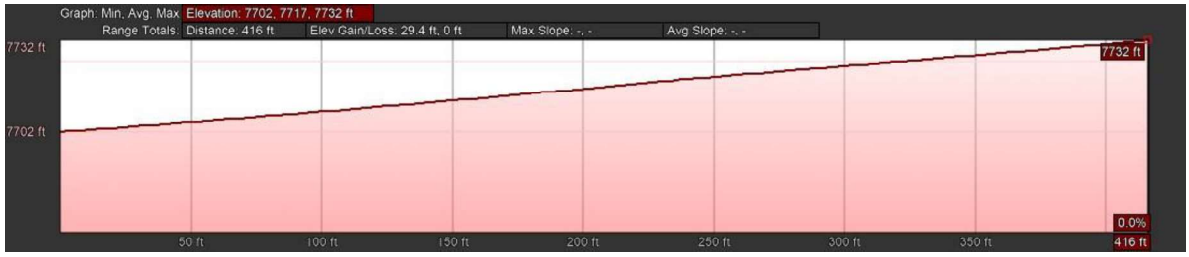

Figura 178: Elevación en la ruta de 100 a 220 metros.

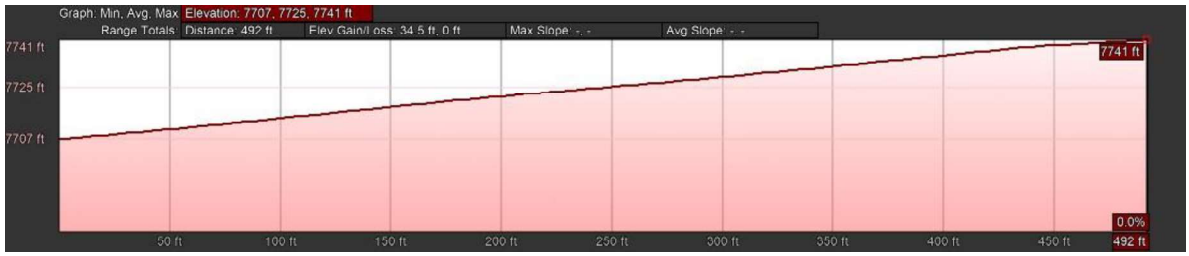

Figura 179: Elevación en la ruta de 120 a 270 metros.

Apéndice B:

Este apéndice está compuesto de fotografías tomadas en los puntos de medición de este trabajo.

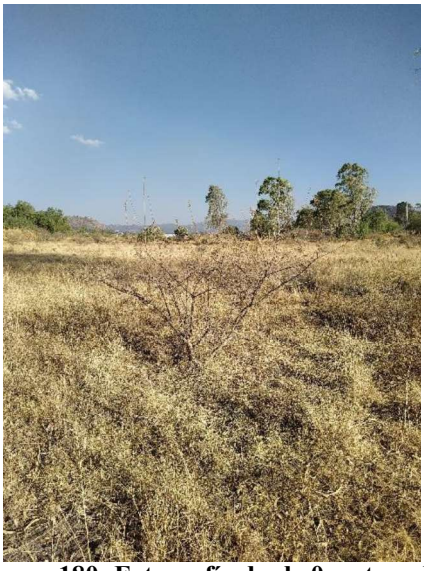

Figura 180: Fotografía desde 0 metros del transmisor.

Figura 181: Fotografía desde 10 metros del transmisor.

Figura 182: Fotografía desde 20 metros del transmisor.

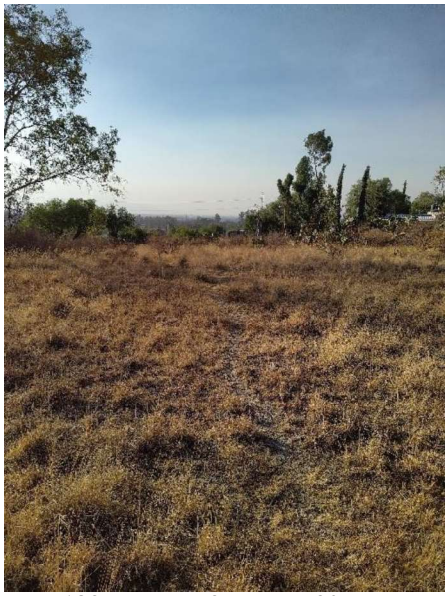

Figura 183: Fotografía desde 30 metros del transmisor.

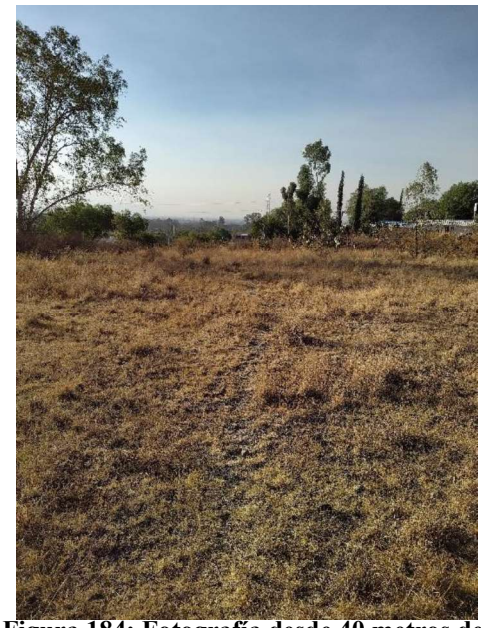

Figura 184: Fotografía desde 40 metros del transmisor.

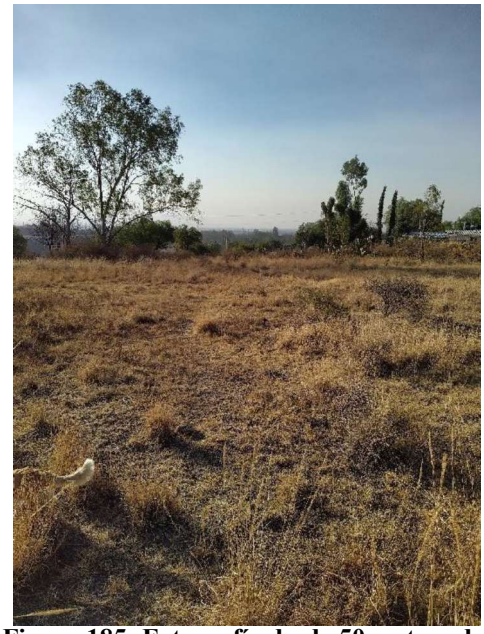

Figura 185: Fotografía desde 50 metros del transmisor.

Figura 186: Fotografía desde 60 metros del transmisor.

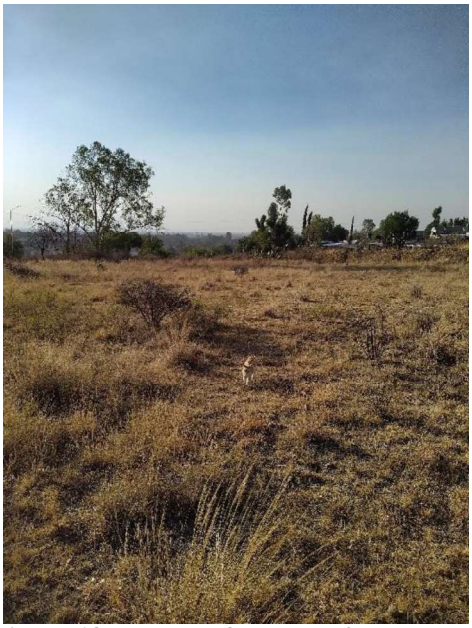

Figura 187: Fotografía desde 70 metros del transmisor.

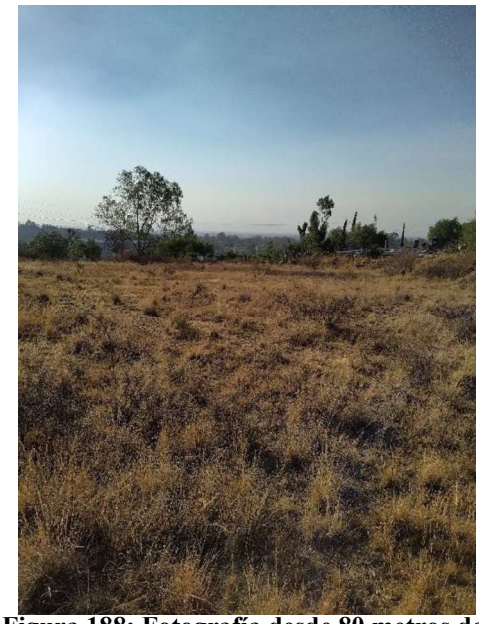

Figura 188: Fotografía desde 80 metros del transmisor (escenarios RSSI y P2P).

Figura 189: Fotografía del terreno elevado ocupado para los escenarios RP, P2M y NM.

Figura 190: Fotografía desde 90 metros del transmisor.

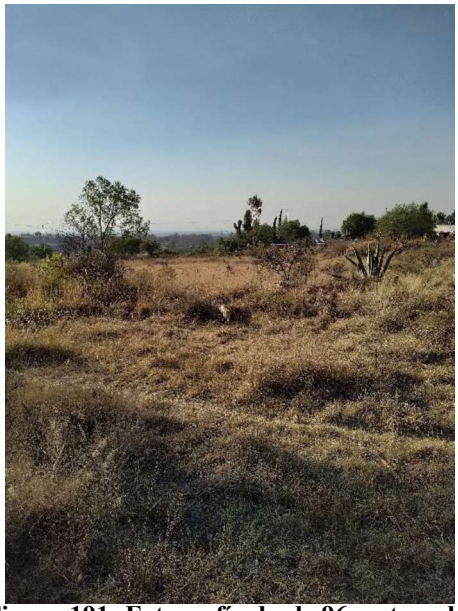

Figura 191: Fotografía desde 96 metros del transmisor.

Figura 192: Fotografía desde 100 metros del transmisor (escenario RSSI y P2P).

Figura 193: Fotografía desde 100 metros del transmisor (escenarios RP, P2M y NM).

Figura 194: Fotografía desde 120 metros del transmisor.

Figura 195: Fotografía desde 140 metros del transmisor.

Figura 196: Fotografía desde 170 metros del transmisor.

Figura 197: Fotografía desde 190 metros del transmisor.

Figura 198. Fotografía desde 220 metros del transmisor.

Figura 199: Fotografía desde 270 metros del transmisor.

Apéndice C:

Este apéndice muestra los códigos usados por Arduino para los escenarios y por la computadora para el procesamiento de la información de este trabajo.

Arduino

Escenarios P2P, RP y NM

Recepción

```
1  //Escenarios P2P, RP y NM
2  uint8_t data[2] = {0,0};
3  uint8_t pastdata = 0;
4  int cont = 0;
5  int NumPrueba = 0;
6
7  void setup() {
8    // put your setup code here, to run once:
9    Serial.begin(9600);
10   Serial1.begin(9600);
11 }
12
13 void loop() {
14   // put your main code here, to run repeatedly:
15   Read();
16 }
17
18 void Read(){
19   while(Serial1.available()>0){
20     int d = Serial1.read();
21     if(d > 200){
22       data[0] = d;
23     }
24     else{
25       data[1] = d;
26       Serial.println(data[1]);
27       if(data[0] != 0 && data[1] > pastdata/10){
28         cont = cont + 1;
29         pastdata = data[1];
```

```
30        }
31        else{
32          cont = 0;
33          data[1] = 0;
34          pastdata = 0;
35          NumPrueba = NumPrueba + 1;
36          Serial.print("NP: ");
37          Serial.println(NumPrueba);
38          break;
39        }
40      }
41    }
42 }
```

Transmisión

```
1  //Escenarios P2P, RP y NM
2  int NumPrueba = 0;
3  void setup() {
4    // put your setup code here, to run once:
5    Serial.begin(9600);
6    Serial1.begin(9600);
7  }
8
9  void loop() {
10   // put your main code here, to run repeatedly:
11   Write();
12 }
13
14 void Write(){
15   int cont = 0;
16   uint8_t H = uint8_t(0xF8);
17   while(cont < 200){
18     Serial1.println(cont);
19     Serial.println(cont);
20     cont++;
21     delay(75);
22   }
23   NumPrueba = NumPrueba + 1;
24   Serial.print("NP: ");
25   Serial.println(NumPrueba);
26   delay(2000);
27 }
```

Escenario P2M

Recepción

```
1  //Escenario P2M
2  uint8_t data[2] = {0,0};
3  uint8_t pastdata[3] = {0,0,0};
4  int NumPrueba[3] = {0,0,0};
5  int cont = 0;
```

```
6  int cont1 = 0;
7  int cont2 = 0;
8  void setup() {
9  //put yout setup code here, to run once:
10 Serial.begin(9600);
11 Serial1.begin(9600);
12 }
13
14 void loop() {
15 //put your main code here, to run repeatedly:
16 Read();
17 }
18
19 void Read(){
20 while(Serial1.available() > 0){
21 int d = Serial1.read();
22 if(d > 200){
23 data[0] = d;
24 }
25 else{
26 data[1] = d;
27 if(data[0] != 0){
28 if(data[0] == 0xF9){
29 Serial.print("ED3:");
30 Serial.println(cont2);
31 if(data[1] > pastdata[2]/10){
32 pastdata[2] = data[1];
33 cont2 = cont2 + 1;
34 }
35 else{
36 data[0] = 0;
37 data[1] = 0;
38 pastdata[2] = 0;
39 cont2 = 0;
40 NumPrueba[2] = NumPrueba[2]+1;
41 Serial.print("NP3: ");
42 Serial.println(NumPrueba[2]);
43 }
44 break;
45 }
46 else if(data[0] == 0xFA){
47 Serial.print("ED2:");
48 Serial.println(cont1);
49 if(data[1] > pastdata[1]/10){
50 pastdata[1] = data[1];
51 cont1 = cont1 + 1;
52 }
53 else{
54 data[0] = 0;
55 data[1] = 0;
56 pastdata[1] = 0;
57 cont1 = 0;
```

```
58 NumPrueba[1] = NumPrueba[1]+1;
59 Serial.print("NP2: ");
60 Serial.println(NumPrueba[1]);
61 }
62 break;
63 }
64 else if(data[0] == 0xFB){
65 Serial.print("ED1:");
66 Serial.println(cont);
67 if(data[1] > pastdata[0]/10){
68 pastdata[0] = data[1];
69 cont = cont + 1;
70 }
71 else{
72 data[0] = 0;
73 data[1] = 0;
74 pastdata[0] = 0;
75 cont = 0;
76 NumPrueba[0] = NumPrueba[0]+1;
77 Serial.print("NP1: ");
78 Serial.println(NumPrueba[0]);
79 }
80 break;
81 }
82 else{
83 Serial.print("error");
84 break;
85 }
86 }
87 }
88 }
89 }
```

Transmisor 1

```
1  //Escenario P2M ED1
2  int NumPrueba = 0;
3  void setup() {
4    // put your setup code here, to run once:
5    Serial.begin(9600);
6    Serial1.begin(9600);
7  }
8
9  void loop() {
10   // put your main code here, to run repeatedly:
11   Write();
12 }
13
14 void Write(){
15   uint8_t cont = uint8_t(0x00);
16   uint8_t H = uint8_t(0xFB);//0xFB para E12C
17   while(cont < 200){
```

```
    Serial1.write(H);
    Serial1.write(cont);
    Serial.println(cont);
    cont++;
    delay(300);
  }
  NumPrueba = NumPrueba + 1;
  Serial.print("NP: ");
  Serial.println(NumPrueba);
  delay(2000);
}
```

Transmisor 2

```
//Escenario P2M ED2
int NumPrueba = 0;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial1.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  Write();
}

void Write(){
  uint8_t cont = uint8_t(0x00);
  uint8_t H = uint8_t(0xFA);//0xFA para E22C
  while(cont < 200){
    Serial1.write(H);
    Serial1.write(cont);
    Serial.println(cont);
    cont++;
    delay(300);
  }
  NumPrueba = NumPrueba + 1;
  Serial.print("NP: ");
  Serial.println(NumPrueba);
  delay(2000);
}
```

Transmisor 3

```
//Escenario P2M ED3
int NumPrueba = 0;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial1.begin(9600);
}
```

```
 8
 9 void loop() {
10   // put your main code here, to run repeatedly:
11   Write();
12 }
13
14 void Write(){
15   uint8_t cont = uint8_t(0x00);
16   uint8_t H = uint8_t(0xF9);//0xF9 para E32C
17   while(cont < 200){
18     Serial1.write(H);
19     Serial1.write(cont);
20     Serial.println(cont);
21     cont++;
22     delay(300);
23   }
24   NumPrueba = NumPrueba + 1;
25   Serial.print("NP: ");
26   Serial.println(NumPrueba);
27   delay(2000);
28 }
```

Código en C++ para el procesamiento de informacion

Clase Segment

Header

```
 1 #ifndef THESIS_H
 2 #define THESIS_H
 3
 4 #include<iostream>
 5 #include<vector>
 6 #include<fstream>
 7 #include<string>
 8 #include<numeric>
 9 #include<random>
10 #include<cmath>
11 #include<chrono>
12
13 /**
14  * @brief La clase Segment toma las mediciones de cada elevacion en segmentos,
15  * cada segmento consta de un numero de pruebas (2 o 4), la clase toma las dichas
16  * pruebas para obtener el numero de paquetes promedio (ya normalizado a 200
17  * paquetes) y el tiempo de procesamiento del segmento.
18 */
19 class Segment
20 {
21 private:
22 std::vector<int> times;
23 std::vector<int> packs;
24 int processingtime;
```

```
25 int average;
26
27 //Setters
28 void Set_processingtime(int _processingtime);
29 void Set_average(int _average);
30
31 public:
32 //Adders
33 void Add_time(int _time);
34 void Add_pack(int _pack);
35 void Add_meassurement(std::string _data);
36
37 //Getters
38 int Get_processingtime();
39 int Get_average();
40 std::vector<int> Get_All_times();
41 std::vector<int> Get_All_packs();
42
43 //Otros
44 void Average(int num_muestras);
45 void Processing_time();
46 void ClearSegment();
47
48 //Prints
49 void Print(int _index);
50 void Print_All();
51
52 };
53
54 #endif // !THESIS_H
```

Cpp

```
1   #include "Segment.h"
2   //Setters
3
4   /**
5   * @brief Agrega el valor del tiempo en procesamiento a la clase.
6   * @param _processingtime es el tiempo de procesamiento obtenido.
7   */
8   void Segment::Set_processingtime(int _processingtime) {
9   processingtime = _processingtime;
10  }
11  /**
12  * @brief Agrega el valor de promedio de paquetes recibidos a la clase.
13  * @param _average es el valor de paquetes recibidos promedio obtenido.
14  */
15  void Segment::Set_average(int _average) {
16  average = _average;
17  }
18
19  //Adders
```

```
20
21 /**
22 * @brief Agrega el tiempo leido del archivo a un vector, mismo que despues sera
23 * usado para obtener el tiempo de procesamiento del segmento.
24 *
25 * @param _time es el tiempo leido
26 */
27 void Segment::Add_time(int _time) {
28 times.push_back(_time);
29 }
30 /**
31 * @brief Agrega un paquete del archivo a un vector, posteriormente es usado para
32 * obtener el promedio de paquetes recibidos del segmento.
33 *
34 * @param _pack es el paquete leido
35 */
36 void Segment::Add_pack(int _pack) {
37 packs.push_back(_pack);
38 }
39 /**
40 * @brief Lee una linea del archivo txt para trandformarla en informacion convirtiendo
41 * los tiempos a milisegundos y agregandolos junto con el paquete a los respectivos
42 * vectores.
43 *
44 * @param _data es la linea del archivo txt.
45 */
46 void Segment::Add_meassurement(std::string _data) {
47 int len = _data.size();
48 int hrs = std::stoi(_data.substr(0, 2));
49 int min = std::stoi(_data.substr(3, 2));
50 int sec = std::stoi(_data.substr(6, 2));
51 int msec = std::stoi(_data.substr(9, 3));
52 int pack = std::stoi(_data.substr(16, len - 16));
53 Add_time(msec + (sec * 1000) + (min * 60 * 1000) + (hrs * 60 * 60 * 1000));
54 Add_pack(pack);
55 }
56
57 //Getters
58
59 /**
60 * @brief Obtiene el tiempo de procesamiento de la clase.
61 * @return el tiempo de procesamiento.
62 */
63 int Segment::Get_processingtime() {
64 return processingtime;
65 }
66 /**
67 * @brief Obtiene el promedio de paquetes recibidos de la clase.
68 * @return el promedio de paquetes recibidos.
69 */
70 int Segment::Get_average() {
71 return average;
```

```
72 }
73 /**
74 * @brief Obtiene el vector de tiempos de la clase.
75 * @return el vector de tiempos.
76 */
77 std::vector<int> Segment::Get_All_times() {
78 return times;
79 }
80 /**
81 * @brief Obtiene el vector de paquetes de la clase
82 * @return el vector de paquetes.
83 */
84 std::vector<int> Segment::Get_All_packs() {
85 return packs;
86 }
87
88 //Otros metodos
89
90 /**
91 * @brief Obtiene el promedio de paquetes recibidos segun el numero de muestras
92 * (pruebas) que se realizo por segmento.
93 *
94 * @param num_muestras es el numero de pruebas por segmento.
95 */
96 void Segment::Average(int num_muestras) {
97 if (!packs.empty())
98 {
99 Set_average(Get_All_packs().size()/num_muestras);
100 }
101 else
102 {
103 std::cout << "No hay paquetes." << std::endl;
104 std::exit(1);
105 }
106 }
107 /**
108 * @brief Ocupa el vector de tiempo para establecer el tiempo que tardo en
109 * procesarse la informacion
110 */
111 void Segment::Processing_time() {
112 if (!times.empty())
113 {
114 Set_processingtime(times[times.size() - 1] - times[0]);
115 }
116 else
117 {
118 std::cout << "No hay tiempos." << std::endl;
119 std::exit(1);
120 }
121 }
122 /**
123 * @brief Elimina la informacion del segmento (limpieza)
```

```
124 */
125 void Segment::ClearSegment() {
126 times.clear();
127 packs.clear();
128 processingtime = 0;
129 average = 0;
130 }
131
132 //Impresiones
133
134 /**
135 * @brief Imprime un determinado valor del segmento en el formato HH:MM:SS.MS - paquete
136 *
137 * @param _index es el indice para saber que paquete mostrar del vector.
138 */
139 void Segment::Print(int _index) {
140 if (_index < times.size() && _index < packs.size())
141 {
142 int ms = times[_index];
143 int hrs = ms / (60 * 60 * 1000);
144 ms = ms - (60 * 60 * 1000 * hrs);
145 int min = ms / (60 * 1000);
146 ms = ms - (60 * 1000 * min);
147 int sec = ms / (1000);
148 ms = ms - (1000 * sec);
149 int pack = packs[_index];
150 std::cout << "\t\t\t" << hrs << ":" << min << ":" << sec << "." << ms << " - " << pack
151 << std::endl;
152 }
153 else {
154 std::cout << "Paquete no encontrado" << std::endl;
155 }
156 }
157 /**
158 * @brief Imprime todos los valoresdel vector bajo el mismo formato del método anterior.
159 */
160 void Segment::Print_All() {
161 if (!times.empty() || !packs.empty())
162 {
163 for (size_t i = 0; i < times.size(); i++)
164 {
165 Print(i);
166 }
167 }
168 else
169 {
170 std::cout << "Vector vacio." << std::endl;
171 std::exit(1);
172 }
173 }
```

Clase Processor

Header

```
#ifndef PROCESSOR_H
#define PROCESSOR_H

#include"Segment.h"

/**
 * @brief La clase Processor se auxilia con la clase Segment, donde ocupa vectores
 * de clases Segment para tener las elevaciones de cada distancia, todo el escenario
 * termina reducido a vectores de vectores de clases Segment, Processor tambien
 * considera un error inicial para la desincronizacion entre los relojes de las
 * computadoras. Processor compara la informacion del transmisor y del receptor,
 * obtiene los tiempos de Delay, el PER, almacena los tiempo de procesamiento y el
 * promedio de paquetes de las clases Segment, ademas de la informacion para la
 * grafica de barras del escenario NM, imprime los resultados y crea un documento
 * CSV cuya informacion es pasada a un documento excel para las graficas.
*/
class Processor
{
private:
        std::vector<std::vector<Segment>> Rx;
        std::vector<std::vector<Segment>> Tx;
        std::vector<std::vector<Segment>> Result;
        std::vector<int> package_averages;
        std::vector<float> PERs;
        std::vector<int> processing_time_rx;
        std::vector<int> processing_time_tx;
        std::vector<float> delay_averages;
        int error;
        int num_muestras;

        //Setters privados
        void Set_error(int _error);
        void Set_num_muestras(int num);
        void Add_package_average(int _package_average);
        void Add_PER(float _PER);
        void Add_processing_time_rx(int _processing_time_rx);
        void Add_processing_time_tx(int _processing_time_tx);
        void Add_delay_average(float delay_average);

public:
        //Constructor
        Processor();

        //Destructor
        ~Processor();

        //Setters publicos
        void Set_Rx(std::string _Rxfilename, int num);
        void Set_Tx(std::string _Txfilename, int num);
```

```
50          void Set_Result();
51
52          //Getters
53          int Get_error();
54          int Get_num_muestras();
55          std::vector<int> Get_package_averages();
56          std::vector<float> Get_PERs();
57          std::vector<int> Get_processing_times_rx();
58          std::vector<int> Get_processing_times_tx();
59          std::vector<float> Get_delay_averages();
60          std::vector<std::vector<Segment>> Get_Rx();
61          std::vector<std::vector<Segment>> Get_Tx();
62          std::vector<std::vector<Segment>> Get_Result();
63
64          //Otros
65          void Delay_averages();
66          void Processing_times();
67          void Package_averages();
68          void PERS();
69          void NM(int Deltat);
70
71          //Print
72          void Check(std::vector<std::vector<Segment>> _Vector);
73          void Print_results();
74          void Make_csv();
75  };
76
77  #endif // !PROCESSOR_H
```

Cpp

```
1   #include "Processor.h"
2   //Constructor
3
4   /**
5   * @brief Construlle la clase con parametros 0.
6   */
7   Processor::Processor(){
8   error = 0;
9   num_muestras = 0;
10  package_averages.clear();
11  processing_time_rx.clear();
12  processing_time_tx.clear();
13  delay_averages.clear();
14  Rx.clear();
15  Tx.clear();
16  Result.clear();
17  }
18  //Destructor
19
20  /**
21  * @brief Limpia la memoria.
```

```
22 */
23 Processor::~Processor() {
24 error = 0;
25 num_muestras = 0;
26 package_averages.clear();
27 processing_time_rx.clear();
28 processing_time_tx.clear();
29 delay_averages.clear();
30 Rx.clear();
31 Tx.clear();
32 Result.clear();
33 }
34 //Setters
35
36 /**
37 * @brief Configura el error con la diferencia del tiempo del primer paquete a distancia
38 0.
39 * @param _error es la diferencia de tiempo.
40 */
41 void Processor::Set_error(int _error) {
42 error = _error;
43 }
44 /**
45 * @brief Configura el numero de muestras (pruebas) por segmento.
46 * @param num es el numero de muestras.
47 */
48 void Processor::Set_num_muestras(int num) {
49 num_muestras = num;
50 }
51 /**
52 * @brief Agrega el promedio de paquetes recibidos de cada segmento a un vector de la
53 clase.
54 * @param _package_average es el promedio de paquetes recibidos.
55 */
56 void Processor::Add_package_average(int _package_average) {
57 package_averages.push_back(_package_average);
58 }
59 /**
60 * @brief Agrega el PER obtenido a un vector de la clase.
61 * @param _PER es esl PER obtenido.
62 */
63 void Processor::Add_PER(float _PER) {
64 PERs.push_back(_PER);
65 }
66 /**
67 * @brief Agrega el valor de tiempo de procesamiento del segmento al vector de recepcion
68 * de la clase.
69 *
70 * @param _processing_time_rx es el tiempo de procesamiento de un segmento en la
71 recepcion
72 */
73 void Processor::Add_processing_time_rx(int _processing_time_rx) {
```

```
74 processing_time_rx.push_back(_processing_time_rx);
75 }
76 /**
77 * @brief Agrega el valor de tiempo de procesamiento del segmento al vector de
78 transmision
79 * de la clase.
80 *
81 * @param _processing_time_tx es le tiempo de procesamiento de un segmento en la
82 transmision
83 */
84 void Processor::Add_processing_time_tx(int _processing_time_tx) {
85 processing_time_tx.push_back(_processing_time_tx);
86 }
87 /**
88 * @brief Agrega el valor de tiempo de retraso promedio obtenido de la diferencia de
89 tiempo
90 * entre la transmision y la recpecion.
91 *
92 * @param _delay_average es el tiempo de retraso promedio obtenido.
93 */
94 void Processor::Add_delay_average(float _delay_average) {
95 delay_averages.push_back(_delay_average);
96 }
97 /**
98 * @brief Recibe el documento, lo lee y transforma los tiempos y paquetes en infomacion
99 * almacenandolos en un vector de recepcion conformado por vectores de distancia
100 * conformados por vectores de elevaciones de clases Segment.
101 *
102 * @param _Rxfilename es el nombre del archivo en txt para la recepcion.
103 * @param num es el numero de muestras (pruebas) para cada segmento.
104 */
105 void Processor::Set_Rx(std::string _Rxfilename, int num) {
106 Set_num_muestras(num);
107 std::string _tmps;
108 std::vector<Segment> _tmpv;
109 Segment _tmpsegment;
110 std::ifstream _rx(_Rxfilename);
111 bool meassuretrigg = false;
112 int distancecount = 0;
113 int meassurecount = 0;
114 int packcount = 0;
115 int linecounter = 0;
116 if (_rx.is_open())
117 {
118 while (std::getline(_rx,_tmps))
119 {
120 if (_tmps.size() < 10 && _tmps.size() > 1) {
121 if (meassuretrigg == false)
122 {
123 meassuretrigg = true;
124 distancecount++;
125 }
```

```
126 else
127 {
128 if (!_tmpsegment.Get_All_packs().empty())
129 {
130 _tmpv.push_back(_tmpsegment);
131 meassurecount++;
132 _tmpsegment.ClearSegment();
133 packcount = 0;
134 }
135 else
136 {
137 std::cout << "Ha ocurrido un error: Objeto no obtenido" << std::endl;
138 std::exit(1);
139 }
140 }
141 }
142 else if (_tmps.size() == 0)
143 {
144 if (meassuretrigg == true) {
145 if (!_tmpv.empty())
146 {
147 _tmpv.push_back(_tmpsegment);
148 _tmpsegment.ClearSegment();
149 Rx.push_back(_tmpv);
150 _tmpv.clear();
151 meassurecount = 0;
152 packcount = 0;
153 meassuretrigg = false;
154 }
155 else
156 {
157 std::cout << "Ha ocurrido un error: Vector no capturado" << std::endl;
158 std::exit(1);
159 }
160 }
161 }
162 else if (_tmps.size()>20)
163 {
164 if (meassuretrigg == true)
165 {
166 packcount++;
167 if (packcount >= Get_num_muestras() + 1)
168 {
169 std::cout << "Ha ocurrido un error: numero de pruebas >" << Get_num_muestras() << " : "
170 << linecounter << std::endl;
171 std::exit(1);
172 }
173 }
174 }
175 else
176 {
177 if (meassuretrigg == true)
```

```
178 {
179 _tmpsegment.Add_meassurement(_tmps);
180 }
181 }
182 linecounter++;
183 
184 }
185 
186 _rx.close();
187 }
188 else
189 {
190 std::cout << "Ha ocurrido un error: No se ha abierto el archivo" << std::endl;
191 }
192 }
193 /**
194 * @brief Recibe el documento, lo lee y transforma los tiempos y paquetes en infomacion
195 * almacenandolos en un vector de transmision conformado por vectores de distancia
196 * conformados por vectores de elevaciones de clases Segment.
197 *
198 * @param _Txfilename es el nombre del archivo txt para la transmision.
199 * @param num es el numero de muestras (pruebas) para cada segmento.
200 */
201 void Processor::Set_Tx(std::string _Txfilename, int num) {
202 Set_num_muestras(num);
203 std::string _tmps;
204 std::vector<Segment> _tmpv;
205 Segment _tmpsegment;
206 std::ifstream _tx(_Txfilename);
207 bool meassuretrigg = false;
208 int distancecount = 0;
209 int meassurecount = 0;
210 int packcount = 0;
211 int linecounter = 0;
212 if (_tx.is_open())
213 {
214 while (std::getline(_tx, _tmps))
215 {
216 if (_tmps.size() < 10 && _tmps.size() > 1) {
217 if (meassuretrigg == false)
218 {
219 meassuretrigg = true;
220 distancecount++;
221 }
222 else
223 {
224 if (!_tmpsegment.Get_All_packs().empty())
225 {
226 _tmpv.push_back(_tmpsegment);
227 meassurecount++;
228 _tmpsegment.ClearSegment();
229 packcount = 0;
```

```
230 }
231 else
232 {
233 std::cout << "Ha ocurrido un error: Objeto no obtenido" << std::endl;
234 std::exit(1);
235 }
236 }
237 }
238 else if (_tmps.size() == 0)
239 {
240 if (meassuretrigg == true) {
241 if (!_tmpv.empty())
242 {
243 _tmpv.push_back(_tmpsegment);
244 _tmpsegment.ClearSegment();
245 Tx.push_back(_tmpv);
246 _tmpv.clear();
247 meassurecount = 0;
248 packcount = 0;
249 meassuretrigg = false;
250 }
251 else
252 {
253 std::cout << "Ha ocurrido un error: Vector no capturado:" << linecounter << std::endl;
254 std::exit(1);
255 }
256 }
257 }
258 else if (_tmps.size() > 20)
259 {
260 if (meassuretrigg == true)
261 {
262 packcount++;
263 if (packcount >= Get_num_muestras() + 1)
264 {
265 std::cout << "Ha ocurrido un error: numero de pruebas >" << Get_num_muestras() << " : "
266 << linecounter << std::endl;
267 std::exit(1);
268 }
269 }
270 }
271 else
272 {
273 if (meassuretrigg == true)
274 {
275 _tmpsegment.Add_meassurement(_tmps);
276 }
277 }
278 linecounter++;
279 }
280 _tx.close();
281 }
```

```
282 else
283 {
284 std::cout << "Ha ocurrido un error: No se ha abierto el archivo" << std::endl;
285 }
286 }
287 /**
288 * @brief Obtiene el tiempo de retraso de todos los paquetes con la diferencia de tiempo
289 * entre la transmision y la recepcion.
290 */
291 void Processor::Set_Result() {
292 Segment _result;
293 std::vector<Segment> _tmpresultv;
294 Set_error(Rx[0][0].Get_All_times()[0] - Tx[0][0].Get_All_times()[0]);
295 for (size_t i = 0; i < Get_Rx().size(); i++)
296 {
297 for (size_t j = 0; j < Get_Rx()[i].size(); j++)
298 {
299 for (size_t k = 0; k < Get_Rx()[i][j].Get_All_packs().size(); k++)
300 {
301 _result.Add_time(std::abs(Rx[i][j].Get_All_times()[k] - Tx[i][j].Get_All_times()[k] -
302 Get_error()));
303 _result.Add_pack(Rx[i][j].Get_All_packs()[k]);
304 }
305 _tmpresultv.push_back(_result);
306 _result.ClearSegment();
307 }
308 Result.push_back(_tmpresultv);
309 _tmpresultv.clear();
310 }
311 }
312 //Getters
313
314 /**
315 * @brief Obtiene el error de sincronizacion de tiempo en la computadoras.
316 * @return el error.
317 */
318 int Processor::Get_error() {
319 return error;
320 }
321 /**
322 * @brief Obtiene el numero de muestras (pruebas) por segmento.
323 * @return el numero de muestras por segmento.
324 */
325 int Processor::Get_num_muestras() {
326 return num_muestras;
327 }
328 /**
329 * @brief Obtiene el vector de promedios de paquetes recibidos.
330 * @return el vector de promedios de paquetes recibidos.
331 */
332 std::vector<int> Processor::Get_package_averages() {
333 return package_averages;
```

```
334 }
335 /**
336 * @brief Obtiene el vector de PERs.
337 * @return el vector de PERs.
338 */
339 std::vector<float> Processor::Get_PERs() {
340 return PERs;
341 }
342 /**
343 * @brief Obtiene el vector de tiempos de procesamiento en la transmision.
344 * @return el vector de tiempos de procesamiento en la transmision.
345 */
346 std::vector<int> Processor::Get_processing_times_rx() {
347 return processing_time_rx;
348 }
349 /**
350 * @brief Obtiene el vector de tiempos de procesamiento en la transmision.
351 * @return el vector de tiempos de procesamiento en la transmision.
352 */
353 std::vector<int> Processor::Get_processing_times_tx() {
354 return processing_time_tx;
355 }
356 /**
357 * @brief Obtiene el vector de retrasos promedio de la clase.
358 * @return el vector de retrasos promedio.
359 */
360 std::vector<float> Processor::Get_delay_averages() {
361 return delay_averages;
362 }
363 /**
364 * @brief Obtiene el vector de recepcion de la clase.
365 * @return el vector de recepcion.
366 */
367 std::vector<std::vector<Segment>> Processor::Get_Rx() {
368 return Rx;
369 }
370 /**
371 * @brief Obtiene el vector de transmision de la clase.
372 * @return el vector de transmision.
373 */
374 std::vector<std::vector<Segment>> Processor::Get_Tx() {
375 return Tx;
376 }
377 /**
378 * @brief Obtiene el vector de resultados de la clase.
379 * @return el vector de resultados.
380 */
381 std::vector<std::vector<Segment>> Processor::Get_Result() {
382 return Result;
383 }
384 //Otros
385
```

```
386 /**
387 * @brief Obtiene el promedio de cada uno de los retasos de los paquetes por segmento
388 */
389 void Processor::Delay_averages() {
390 int _tmp = 0;
391 if (!Result.empty())
392 {
393 for (size_t i = 0; i < Result.size(); i++)
394 {
395 for (size_t j = 0; j < Result[i].size(); j++)
396 {
397 for (size_t k = 0; k < Result[i][j].Get_All_times().size(); k++)
398 {
399 _tmp += Result[i][j].Get_All_times()[k];
400 }
401 delay_averages.push_back(float(_tmp) / float(Result[i][j].Get_All_times().size()));
402 _tmp = 0;
403 }
404 }
405 }
406 else
407 {
408 std::cout << "Ha ocurrido un problema: Sin resultados." << std::endl;
409 std::exit(1);
410 }
411 }
412 /**
413 * @brief Agrega los tiempos de procesamiento a los vectores de la clase Processor
414 */
415 void Processor::Processing_times() {
416 if (!Rx.empty() && !Tx.empty())
417 {
418 for (size_t i = 0; i < Rx.size(); i++)
419 {
420 for (size_t j = 0; j < Rx[i].size(); j++)
421 {
422 Rx[i][j].Processing_time();
423 Add_processing_time_rx(Rx[i][j].Get_processingtime());
424 }
425 }
426
427 for (size_t i = 0; i < Tx.size(); i++)
428 {
429 for (size_t j = 0; j < Tx[i].size(); j++)
430 {
431 Tx[i][j].Processing_time();
432 Add_processing_time_tx(Tx[i][j].Get_processingtime());
433 }
434 }
435 }
436 else
437 {
```

```
438 std::cout << "Ha ocurrido un error: Rx o Tx estan vacios." << std::endl;
439 std::exit(1);
440 }
441 }
442 /**
443 * @brief Agrega el promedio de los paquetes recibidos de los segmentos a un vector de
444 la clase.
445 */
446 void Processor::Package_averages() {
447 if (!Rx.empty())
448 {
449 for (size_t i = 0; i < Rx.size(); i++)
450 {
451 for (size_t j = 0; j < Rx[i].size(); j++)
452 {
453 Rx[i][j].Average(Get_num_muestras());
454 Add_package_average(Rx[i][j].Get_average());
455 }
456 }
457 }
458 else
459 {
460 std::cout << "Ha ocurrido un error: Rx esta vacio." << std::endl;
461 std::exit(1);
462 }
463 }
464 /**
465 * @brief Calcula el PER de cada promedio de paquetes recibidos y los agrega al vector
466 * de la clase.
467 */
468 void Processor::PERS() {
469 if (!Get_package_averages().empty())
470 {
471 for (size_t i = 0; i < Get_package_averages().size(); i++)
472 {
473 Add_PER(float(1.0 - Get_package_averages()[i] / 200.0));
474 }
475 }
476 else
477 {
478 std::cout << "Se recomienda, primero, llenar los promedios de paquetes." << std::endl;
479 }
480 }
481 /**
482 * @brief Separa la informacion, segun el tiempo, en 8 segmentos en los que cada
483 * paquete fue recibido. estos segmentos aparecen en la grafica de barras para el
484 * escenario NM, el metodo tambien crea un archivo CSV, pero es exclusivo para los
485 * resultados de este escenario.
486 *
487 * @param Deltat es el tiempo que se considera para separar cada segemento de una
488 * supuesta distancia.
489 */
```

```
490 void Processor::NM(int Deltat)
491 {
492 if (!Get_Rx().empty())
493 {
494 std::vector<std::vector<std::string>> Distance;
495 std::vector<std::string> _tmps1;
496 std::vector<std::string> _tmps2;
497 std::vector<std::string> _tmps3;
498 std::vector<std::string> _tmps4;
499 std::vector<std::string> _tmps5;
500 std::vector<std::string> _tmps6;
501 std::vector<std::string> _tmps7;
502 std::vector<std::string> _tmps8;
503
504 std::vector<int> NM_averages;
505 std::vector<int> NM_processing_times;
506
507 int datacounter = 0;
508
509 for (size_t i = 0; i < Get_Rx().size(); i++)
510 {
511 for (size_t j = 0; j < Get_Rx()[i].size(); j++)
512 {
513 int t0 = Get_Rx()[i][j].Get_All_times()[0];
514 datacounter = datacounter + Get_Rx()[i][j].Get_All_times().size();
515 for (size_t k = 0; k < Get_Rx()[i][j].Get_All_times().size(); k++)
516 {
517 if (Get_Rx()[i][j].Get_All_times()[k] >= t0 && Get_Rx()[i][j].Get_All_times()[k] < t0 +
518 Deltat || Get_Rx()[i][j].Get_All_times()[k] >= t0 + (7 * Deltat) &&
519 Get_Rx()[i][j].Get_All_times()[k] < t0 + (8 * Deltat))
520 {
521 _tmps5.push_back(std::to_string(i) + '|' + std::to_string(j) + '|' +
522 std::to_string(k));
523 }
524 if (Get_Rx()[i][j].Get_All_times()[k] >= t0 + Deltat &&
525 Get_Rx()[i][j].Get_All_times()[k] < t0 + (2 * Deltat) ||
526 Get_Rx()[i][j].Get_All_times()[k] >= t0 + (6 * Deltat) &&
527 Get_Rx()[i][j].Get_All_times()[k] < t0 + (7 * Deltat))
528 {
529 _tmps6.push_back(std::to_string(i) + '|' + std::to_string(j) + '|' +
530 std::to_string(k));
531 }
532 if (Get_Rx()[i][j].Get_All_times()[k] >= t0 + (2 * Deltat) &&
533 Get_Rx()[i][j].Get_All_times()[k] < t0 + (3 * Deltat) ||
534 Get_Rx()[i][j].Get_All_times()[k] >= t0 + (5 * Deltat) &&
535 Get_Rx()[i][j].Get_All_times()[k] <= t0 + (6 * Deltat))
536 {
537 _tmps7.push_back(std::to_string(i) + '|' + std::to_string(j) + '|' +
538 std::to_string(k));
539 }
540 if (Get_Rx()[i][j].Get_All_times()[k] >= t0 + (3 * Deltat) &&
541 Get_Rx()[i][j].Get_All_times()[k] < t0 + (4 * Deltat) ||
```

```
542 Get_Rx()[i][j].Get_All_times()[k] >= t0 + (4 * Deltat) &&
543 Get_Rx()[i][j].Get_All_times()[k] < t0 + (5 * Deltat))
544 {
545 _tmps8.push_back(std::to_string(i) + '|' + std::to_string(j) + '|' +
546 std::to_string(k));
547 }
548 if (Get_Rx()[i][j].Get_All_times()[k] >= t0 + (8 * Deltat) &&
549 Get_Rx()[i][j].Get_All_times()[k] < t0 + (9 * Deltat) ||
550 Get_Rx()[i][j].Get_All_times()[k] >= t0 + (15 * Deltat) &&
551 Get_Rx()[i][j].Get_All_times()[k] < t0 + (16 * Deltat))
552 {
553 _tmps4.push_back(std::to_string(i) + '|' + std::to_string(j) + '|' +
554 std::to_string(k));
555 }
556 if (Get_Rx()[i][j].Get_All_times()[k] >= t0 + (9 * Deltat) &&
557 Get_Rx()[i][j].Get_All_times()[k] < t0 + (10 * Deltat) ||
558 Get_Rx()[i][j].Get_All_times()[k] >= t0 + (14 * Deltat) &&
559 Get_Rx()[i][j].Get_All_times()[k] < t0 + (15 * Deltat))
560 {
561 _tmps3.push_back(std::to_string(i) + '|' + std::to_string(j) + '|' +
562 std::to_string(k));
563 }
564 if (Get_Rx()[i][j].Get_All_times()[k] >= t0 + (10 * Deltat) &&
565 Get_Rx()[i][j].Get_All_times()[k] < t0 + (11 * Deltat) ||
566 Get_Rx()[i][j].Get_All_times()[k] >= t0 + (13 * Deltat) &&
567 Get_Rx()[i][j].Get_All_times()[k] < t0 + (14 * Deltat))
568 {
569 _tmps2.push_back(std::to_string(i) + '|' + std::to_string(j) + '|' +
570 std::to_string(k));
571 }
572 if (Get_Rx()[i][j].Get_All_times()[k] >= t0 + (11 * Deltat) &&
573 Get_Rx()[i][j].Get_All_times()[k] < t0 + (12 * Deltat) ||
574 Get_Rx()[i][j].Get_All_times()[k] >= t0 + (12 * Deltat) &&
575 Get_Rx()[i][j].Get_All_times()[k] < t0 + (13 * Deltat))
576 {
577 _tmps1.push_back(std::to_string(i) + '|' + std::to_string(j) + '|' +
578 std::to_string(k));
579 }
580 if (Get_Rx()[i][j].Get_All_times()[k] >= t0 + (16 * Deltat))
581 {
582 _tmps5.push_back(std::to_string(i) + '|' + std::to_string(j) + '|' +
583 std::to_string(k));
584 t0 = t0 + (16 * Deltat);
585 }
586
587 }
588 Rx[i][j].Average(Get_num_muestras());
589 Rx[i][j].Processing_time();
590 NM_averages.push_back(Get_Rx()[i][j].Get_average());
591 NM_processing_times.push_back(Get_Rx()[i][j].Get_processingtime());
592 Distance.push_back(_tmps1);
593 Distance.push_back(_tmps2);
```

```
Distance.push_back(_tmps3);
Distance.push_back(_tmps4);
Distance.push_back(_tmps5);
Distance.push_back(_tmps6);
Distance.push_back(_tmps7);
Distance.push_back(_tmps8);
_tmps1.clear();
_tmps2.clear();
_tmps3.clear();
_tmps4.clear();
_tmps5.clear();
_tmps6.clear();
_tmps7.clear();
_tmps8.clear();
}
std::cout << datacounter << std::endl;

}
for (size_t i = 0; i < Distance.size(); i++)
{
std::cout << Distance[i][0] << std::endl;
}
std::ofstream results;
std::string _tmps;
std::vector<float> NM_PER;
for (size_t i = 0; i < NM_averages.size(); i++)
{
NM_PER.push_back(float(1 - (NM_averages[i] / 200.0)));
}

int rebote = 0;
int reboted = 0;
for (size_t i = 0; i < Get_Rx().size(); i++)
{
for (size_t j = 0; j < Get_Rx()[i].size(); j++)
{
_tmps.append(std::to_string(NM_averages[j + rebote]) + ",");
}
for (size_t j = 0; j < Get_Rx()[i].size(); j++)
{
_tmps.append(std::to_string(NM_PER[j + rebote]) + ",");
}
for (size_t j = 0; j < 24; j++)
{
_tmps.append(std::to_string(Distance[j + reboted].size()) + ",");
}
for (size_t j = 0; j < Get_Rx()[i].size(); j++)
{
_tmps.append(std::to_string(NM_processing_times[j + rebote]/(60.0 * 1000.0)) + ",");
}
_tmps.append("\n");
rebote = rebote + Get_Rx()[i].size();
```

```
reboted = reboted + 24;
}
results.open("NMresults.csv");
results << _tmps;
results.close();
}
else
{
std::cout << "Sin datos para el NM" << std::endl;
}
}
//Print

/**
* @brief Imprime un vector, ya sea de transmision o de recepcion, con el fin de que
* en la pantalla pueda verse que los segmentos, elevaciones y distancias se han
* procesado correctamente.
*
* @param _Vector es el vector a Imprimir.
*/
void Processor::Check(std::vector<std::vector<Segment>> _Vector) {
for (size_t i = 0; i < _Vector.size(); i++)
{
std::cout << "Distance: " << float(i) << std::endl;
for (size_t j = 0; j < _Vector[i].size(); j++)
{
std::cout << "\tMeassurement: " << j << std::endl;
std::cout << "\t\t\t\t" << _Vector[i][j].Get_All_times().size() << std::endl;
}
}
}
/**
* @brief Imprime todos los vectores y resultados procesados en la pantalla.
*/
void Processor::Print_results() {
if (!Get_delay_averages().empty() && !Get_package_averages().empty() &&
!Get_processing_times_rx().empty())
{
std::cout << "Resultados" << std::endl;
std::cout << "Error obtenido: " << Get_error() << "ms" << std::endl;
std::cout << "Promedio de Delays[s]" << "\tPromedio de paquetes" << "\tPER" <<
"\tTiempo de procesamiento en Rx[min]" << "\tTiempo de procesamiento en Tx[min]" <<
std::endl;
for (size_t i = 0; i < Get_delay_averages().size(); i++)
{
std::cout << "\t" << float(Get_delay_averages()[i] / 1000.0) << "\t\t" <<
Get_package_averages()[i] << "\t\t\t" << Get_PERs()[i] << "\t\t\t" <<
float(Get_processing_times_rx()[i]) / (1000.0 * 60.0) << "\t\t\t" <<
float(Get_processing_times_tx()[i])/(1000.0 * 60.0) << std::endl;
}
}
else
```

```
698 {
699 std::cout << "Elementos incompletos." << std::endl;
700 std::exit(1);
701 }
702 }
703 /**
704 * @brief Crea el archivo CSV con todos los resultados procesados para que sea posible
705 * pasarlos
706 * a excel y obtener las graficas.
707 */
708 void Processor::Make_csv() {
709 if (!Get_delay_averages().empty() && !Get_package_averages().empty() &&
710 !Get_PERs().empty() && !Get_processing_times_rx().empty())
711 {
712 std::ofstream results;
713 std::string _tmps;
714 size_t rebote = 0;
715 results.open("Daresults.csv");
716 for (size_t i = 0; i < Get_Rx().size(); i++)
717 {
718 for (size_t j = 0; j < Get_Rx()[i].size(); j++)
719 {
720 _tmps.append(std::to_string(Get_package_averages()[j+rebote]));
721 _tmps.append(",");
722
723 }
724 for (size_t k = 0; k < Get_Rx()[i].size(); k++)
725 {
726 _tmps.append(std::to_string(Get_PERs()[k + rebote]));
    _tmps.append(",");
    }
    for (size_t l = 0; l < Get_Rx()[i].size(); l++)
    {

    _tmps.append(std::to_string(Get_delay_averages()[l + rebote] / 1000.0));
    _tmps.append(",");

    }
    for (size_t m = 0; m < Get_Rx()[i].size(); m++)
    {
    _tmps.append(std::to_string(Get_processing_times_rx()[m + rebote] / (60.0 * 1000.0)));
    _tmps.append(",");
    _tmps.append(std::to_string(Get_processing_times_tx()[m + rebote] / (60.0 * 1000.0)));
    _tmps.append(",");
    }
    rebote = rebote + Get_Rx()[i].size();
    _tmps.append("\n");
    }
    results << _tmps;
    results.close();
    }
    else
```

```
{
std::cout << "Datos incompletos." << std::endl;
}
}
```

Main

```
#include"Processor.h"
/**
* @brief Durante el escenario P2M, el receptor obtiene infomacion de 3 fuentes
diferentes, esta
* funcion separa la informacion de cada fuente y crea archivos txt para cada fuente,
cada documento
* de txt se usa para el procesamiento individual con su respectivo transmisor.
*
* @param _multidata es el archivo en bruto del receptor con todas las fuentes.
* @param num es el numero de fuentes en el archivo _multidata.
*/
void Separator(std::string _multidata, int num) {
std::ifstream data(_multidata);
std::ofstream Writter;
std::string _tmps;
int cont = 0;
if (data.is_open())
{
while (std::getline(data, _tmps))
{
std::cout << "Linea de datos:" + std::to_string(cont) << std::endl;
if (_tmps.size() > 19 && _tmps.size() < 25)
{
std::string _timewa = _tmps.substr(0, 16);
std::string datavalidation = _tmps.substr(16, 2);
std::string info = _tmps.substr(20, _tmps.size() - 20);
std::string file = _tmps.substr(18, 1);
if (datavalidation == "ED")
{
Writter.open(file + ".txt", std::fstream::app);
Writter << _timewa + info << std::endl;
Writter.close();
}
else
{
Writter.open(file + ".txt", std::fstream::app);
Writter << _timewa + "NP:" + info << std::endl;
Writter.close();
}
}
else
{
std::cout << "Cadena no valida para analisis:" + _tmps << std::endl;
std::exit(1);
}
```

```
cont++;
}
}
}
/**
* @brief Tener dos sistemas operativos en una computadora puede generar un error en el
reloj,
* generalmente no cuesta mucho actualizar porque lo hace con Internet, sin embargo,
durante una
* de las mediciones de RP (sin Internet), este problema ocurrio. La solucion fue
desarrollar esta
* funcion que permite modificar el tiempo de todo un archivo, compensando el tiempo con
cierto error
* en las Horas,minutos y segundos, la funcion crea un archivo txt con el tiempo
corregido.
* El error de sincronizacion de tiempo en las computadoras, durante esta unica prueba,
es de 0.
*
* @param _Wrongfile es el archivo con errores.
* @param errorh es la diferencia de tiempo en horas.
* @param errorm es la diferencia de tiempo en minutos.
* @param errors es la diferencia de tiempo en segundos.
*/
void ClockErrorFixer(std::string _Wrongfile, int errorh, int errorm, int errors) {
std::ifstream wdata(_Wrongfile);
std::ofstream fdata;
std::string _tmps;
fdata.open("fixed-" + _Wrongfile + ".txt");
if (wdata.is_open())
{
while (std::getline(wdata, _tmps))
{
if (_tmps.size() > 5)
{
std::string h = _tmps.substr(0, 2);
std::string m = _tmps.substr(3, 2);
std::string s = _tmps.substr(6, 2);
std::string _Everythingelse = _tmps.substr(8, _tmps.size() - 8);
int _h = stoi(h) + errorh;
int _m = stoi(m) + errorm;
int _s = stoi(s) + errors;
if (_s >= 60)
{
_m++;
_s = _s - 60;
}
if (_m >= 60)
{
_h++;
_m = _m - 60;
}
if (_s <= -1)
```

```
98  {
99  _m--;
100 _s = _s + 60;
101 }
102 if (_m <= -1)
103 {
104 _h--;
105 _m = _m + 60;
106 }
107 std::string fh = std::to_string(_h);
108 std::string fm = std::to_string(_m);
109 std::string fs = std::to_string(_s);
110 if (fh.size() < 2)
111 {
112 fh = "0" + fh;
113 }
114 if (fm.size() < 2)
115 {
116 fm = "0" + fm;
117 }
118 if (fs.size() < 2)
119 {
120 fs = "0" + fs;
121 }
122 fdata <<  fh + ":" + fm + ":" + fs + _Everythingelse << std::endl;
123 }
124 else
125 {
126 fdata << _tmps << std::endl;
127 }
128
129 }
130 fdata.close();
131 }
132 }
133 /**
134 * @brief A la mitad de una sesion de P2M, una computadora perdio conexion con los datos
135 * del dispositivo 2, perdiendo informacion de parte del transmisor. La solucion fue
136 * estudiar las variaciones que Arduino metia a la informacion transmitida, lo mas
137 cercano
138 * al comportamiento fue insertar variables aleatorias condistribucion normal
139 encontrando el
140 * promedio y la desviacion estandar de la informacion existente del transmisor, el
141 resultado
142 * fueron tiempos de transmision con variaciones maximas de decimas de milisegundo de la
143 * informacion faltante. Por lo que esta funcion fue creada para finalizar la
144 informacion de
145 * este archivo txt ocupando esta generacion aleatoria. La funcion requiere del archivo
146 incompelto
147 * la direccion del llenado y el limite del mismo en tiempo. Crea el archivo con un
148 aproximado
149 * de los tiempos de la informacion faltante.
```

```
150 *
151 * @param _incompletefile es el archivo incompleto.
152 * @param direction es si el archivo a a rellenar hacia arriba o hacia abajo.
153 * @param limit es el limite de tiempo en el que la funcion debe parar.
154 */
155 void Completer(std::string _incompletefile, std::string direction, int limit) {
156
157 std::ifstream idata(_incompletefile);
158 std::ofstream cdata;
159 std::string _tmps;
160 std::vector<int> times;
161 std::vector<int> diferencesp;
162 std::vector<int> diferencesm;
163 std::vector<int> desvm;
164 std::vector<int> desvp;
165 std::vector<int> Pruebas_rescatadas;
166 unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
167 std::default_random_engine generator(seed);
168 size_t index = 0, inext = 0;
169 int p2c = 0;
170
171 while (std::getline(idata, _tmps))
172 {
173 std::string h = _tmps.substr(0, 2);
174 std::string m = _tmps.substr(3, 2);
175 std::string s = _tmps.substr(6, 2);
176 std::string ms = _tmps.substr(9, 3);
177 std::string N = _tmps.substr(16, 2);
178 std::string _Everythingelse = _tmps.substr(12, _tmps.size() - 12);
179 if (N == "NP")
180 {
181 std::string _stmps = _tmps.substr(20,_tmps.size() - 20);
182 int ttmps = stoi(_stmps);
183 Pruebas_rescatadas.push_back(ttmps);
184 }
185
186 int _h = stoi(h);
187 int _m = stoi(m);
188 int _s = stoi(s);
189 int _ms = stoi(ms);
190 times.push_back(_ms + (_s * 1000) + (_m * 1000 * 60) + (_h * 1000 * 60 * 60));
191 }
192 for (size_t i = 0; i < Pruebas_rescatadas.size(); i++)
193 {
194 std::cout << Pruebas_rescatadas[i] << std::endl;
195 }
196 while (inext < times.size() - 1)
197 {
198 inext = index + 1;
199 if ((times[inext] - times[index]) < 1000)
200 {
201 diferencesm.push_back(times[inext] - times[index]);
```

```
202 }
203 else
204 {
205 diferencesp.push_back(times[inext] - times[index]);
206 }
207
208 index++;
209 }
210 double averagem = std::accumulate(diferencesm.begin(), diferencesm.end(), 0.0) /
211 diferencesm.size();
212 double averagep = std::accumulate(diferencesp.begin(), diferencesp.end(), 0.0) /
213 diferencesp.size();
214 std::cout << averagem << std::endl;
215 std::cout << averagep << std::endl;
216
217 for (size_t i = 0; i < diferencesm.size(); i++)
218 {
219 desvm.push_back(std::pow((diferencesm[i] - averagem), 2));
220 }
221 for (size_t i = 0; i < diferencesp.size(); i++)
222 {
223 desvp.push_back(std::pow((diferencesp[i] - averagep), 2));
224 }
225
226 float sigmam = std::sqrt(std::accumulate(desvm.begin(), desvm.end(), 0.0) /
227 desvm.size());
228 float sigmap = std::sqrt(std::accumulate(desvp.begin(), desvp.end(), 0.0) /
229 desvp.size());
230 std::cout << sigmam << std::endl;
231 std::cout << sigmap << std::endl;
232
233 std::normal_distribution<double> distributionm(averagem * 1.0, std::sqrt(sigmam *
234 1.0));
235 std::normal_distribution<double> distributionp(averagep * 1.0, std::sqrt(sigmap *
236 1.0));
237
238 if (direction == "up" && limit < Pruebas_rescatadas[0])
239 {
240
241 p2c = limit;
242 int IoC = times[0] - (((averagem * 200) + averagep)*(Pruebas_rescatadas[0] - limit));
243 int CoM = 0;
244 cdata.open("completed" + _incompletefile);
245 while (p2c <= Pruebas_rescatadas[0])
246 {
247 int NMs = IoC;
248 int NH = std::round(NMs / (60 * 60 * 1000));
249 NMs = NMs - (60 * 60 * 1000 * NH);
250 int NMin = std::round(NMs / (60 * 1000));
251 NMs = NMs - (60 * 1000 * NMin);
252 int NS = std::round(NMs / 1000);
253 NMs = NMs - (1000 * NS);
```

```
254 if (NMin >= 60)
255 {
256 NH++;
257 NMin = 0;
258 }
259 if (NS >= 60)
260 {
261 NMin++;
262 NS = 0;
263 }
264 if (NMs >= 1000)
265 {
266 NS++;
267 NMs = 0;
268 }
269 std::string NHS = std::to_string(NH);
270 std::string NMinS = std::to_string(NMin);
271 std::string NSS = std::to_string(NS);
272 std::string NMsS = std::to_string(NMs);
273 if (NHS.size() < 2)
274 {
275 NHS = "0" + NHS;
276 }
277 if (NMinS.size() < 2)
278 {
279 NMinS = "0" + NMinS;
280 }
281 if (NSS.size() < 2)
282 {
283 NSS = "0" + NSS;
284 }
285 while (NMsS.size() < 3)
286 {
287 NMsS = "0" + NMsS;
288 }
289 std::string NInput = NHS + ":" + NMinS + ":" + NSS + "." + NMsS + " -> NP: " +
290 std::to_string(p2c);
291 cdata << NInput << std::endl;
292 while (CoM < 200)
293 {
294 int NMs = IoC;
295 int NH = std::round(NMs / (60 * 60 * 1000));
296 NMs = NMs - (60 * 60 * 1000 * NH);
297 int NMin = std::round(NMs / (60 * 1000));
298 NMs = NMs - (60 * 1000 * NMin);
299 int NS = std::round(NMs / 1000);
300 NMs = NMs - (1000 * NS);
301 if (NMin >= 60)
302 {
303 NH++;
304 NMin = 0;
305 }
```

```
306 if (NS >= 60)
307 {
308 NMin++;
309 NS = 0;
310 }
311 if (NMs >= 1000)
312 {
313 NS++;
314 NMs = 0;
315 }
316 std::string NHS = std::to_string(NH);
317 std::string NMinS = std::to_string(NMin);
318 std::string NSS = std::to_string(NS);
319 std::string NMsS = std::to_string(NMs);
320 if (NHS.size() < 2)
321 {
322 NHS = "0" + NHS;
323 }
324 if (NMinS.size() < 2)
325 {
326 NMinS = "0" + NMinS;
327 }
328 if (NSS.size() < 2)
329 {
330 NSS = "0" + NSS;
331 }
332 while (NMsS.size() < 3)
333 {
334 NMsS = "0" + NMsS;
335 }
336 std::string NInput = NHS + ":" + NMinS + ":" + NSS + "." + NMsS + " -> " +
337 std::to_string(CoM);
338 cdata << NInput << std::endl;
339 IoC = IoC + std::round(distributionm(generator));
340 CoM = CoM + 1;
341 }
342 IoC = IoC + std::round(distributionp(generator));
343 p2c = p2c + 1;
344 CoM = 0;
345 }
346 cdata.close();
347
348 }
349 else if (direction == "down" && limit > Pruebas_rescatadas[Pruebas_rescatadas.size()])
350 {
351 p2c = limit - Pruebas_rescatadas[Pruebas_rescatadas.size() - 1];
352 int IoC = times[times.size() - 1] + averagem + averagep;
353 int CoM = 1;
354 cdata.open("completed" + _incompletefile);
355 while (p2c <= limit)
356 {
357 int NMs = IoC;
```

```
358 int NH = std::round(NMs / (60 * 60 * 1000));
359 NMs = NMs - (60 * 60 * 1000 * NH);
360 int NMin = std::round(NMs / (60 * 1000));
361 NMs = NMs - (60 * 1000 * NMin);
362 int NS = std::round(NMs / 1000);
363 NMs = NMs - (1000 * NS);
364 if (NMin >= 60)
365 {
366 NH++;
367 NMin = 0;
368 }
369 if (NS >= 60)
370 {
371 NMin++;
372 NS = 0;
373 }
374 if (NMs >= 1000)
375 {
376 NS++;
377 NMs = 0;
378 }
379 std::string NHS = std::to_string(NH);
380 std::string NMinS = std::to_string(NMin);
381 std::string NSS = std::to_string(NS);
382 std::string NMsS = std::to_string(NMs);
383 if (NHS.size() < 2)
384 {
385 NHS = "0" + NHS;
386 }
387 if (NMinS.size() < 2)
388 {
389 NMinS = "0" + NMinS;
390 }
391 if (NSS.size() < 2)
392 {
393 NSS = "0" + NSS;
394 }
395 while (NMsS.size() < 3)
396 {
397 NMsS = "0" + NMsS;
398 }
399 std::string NInput = NHS + ":" + NMinS + ":" + NSS + "." + NMsS + " -> NP: " +
400 std::to_string(p2c);
401 cdata << NInput << std::endl;
402 while (CoM < 200)
403 {
404 int NMs = IoC;
405 int NH = std::round(NMs / (60 * 60 * 1000));
406 NMs = NMs - (60 * 60 * 1000 * NH);
407 int NMin = std::round(NMs / (60 * 1000));
408 NMs = NMs - (60 * 1000 * NMin);
409 int NS = std::round(NMs / 1000);
```

```
410 NMs = NMs - (1000 * NS);
411 if (NMin >= 60)
412 {
413 NH++;
414 NMin = 0;
415 }
416 if (NS >= 60)
417 {
418 NMin++;
419 NS = 0;
420 }
421 if (NMs >= 1000)
422 {
423 NS++;
424 NMs = 0;
425 }
426 std::string NHS = std::to_string(NH);
427 std::string NMinS = std::to_string(NMin);
428 std::string NSS = std::to_string(NS);
429 std::string NMsS = std::to_string(NMs);
430 if (NHS.size() < 2)
431 {
432 NHS = "0" + NHS;
433 }
434 if (NMinS.size() < 2)
435 {
436 NMinS = "0" + NMinS;
437 }
438 if (NSS.size() < 2)
439 {
440 NSS = "0" + NSS;
441 }
442 while (NMsS.size() < 3)
443 {
444 NMsS = "0" + NMsS;
445 }
446 std::string NInput = NHS + ":" + NMinS + ":" + NSS + "." + NMsS + " -> " +
447 std::to_string(CoM);
448 cdata << NInput << std::endl;
449 IoC = IoC + std::round(distributionm(generator));
450 CoM = CoM + 1;
451 }
452 IoC = IoC + std::round(distributionp(generator));
453 p2c = p2c + 1;
454 CoM = 0;
455 }
456 cdata.close();
457 }
458 else
459 {
460 std::cout << "Direccion no definida" << std::endl;
461 }
```

```
462 }
463
464 int main() {
465
466 //Completer("P2MTX2TX0123.txt", "up", 0);
467 //ClockErrorFixer("completedP2MTX2TX0123.txt", 0, 0, -8);
468 //Separator("P2MDM0123TXRX.txt", 3);
469 //int Deltat = (4*60*60*1000)/(5.85*1000);
470 /*Processor processor;
471 processor.Set_Rx("2DM.txt", 4);
472 processor.Check(processor.Get_Rx());
473 //std::cout << Deltat << std::endl;
    processor.Set_Tx("P2MDM2.txt", 4);
    processor.Check(processor.Get_Tx());
    processor.Set_Result();

    processor.Delay_averages();
    processor.Processing_times();
    processor.Package_averages();
    processor.PERS();

    //processor.NM(Deltat);

    processor.Print_results();

    processor.Make_csv();
    */
    system("pause");
    return 0;
    }
```