Tecnologías de comunicación para aplicaciones del Internet de las Cosas.

Tesis que presenta
Alejandra Flores Buruel

Como requisito para obtener el grado de
Maestro en Ingeniería en Sistemas Computacionales

Director de tesis:
Dr. Arnoldo Díaz Ramírez.

Codirector de tesis:
Dr. Jesús Elías Miranda Vega

Mexicali, Baja California, México.
Diciembre 2021

Mexicali, Baja California, 10/Enero/2022
Oficio No.: DEPI-05/ 2022
ASUNTO: Autorización de Impresión

ALEJANDRA FLORES BURUEL
PRESENTE.

El que suscribe, Jefe de la División de Estudios de Posgrado e Investigación, comunica a usted que para la obtención del grado de **Maestría en Sistemas Computacionales**, se ha **autorizado la impresión de su trabajo de Tesis**, cuyo título es:

"Tecnologías de comunicación para aplicaciones del Internet de las Cosas"

La autorización se emite con base a la revisión y dictamen emitido favorablemente y avalado con su rúbrica por los integrantes del Comité Tutorial, integrado por:

ARNOLDO DÍAZ RAMÍREZ
Presidente

JESÚS ELÍAS MIRANDA VEGA
Secretario

VERÓNICA QUINTERO ROSAS
Vocal

A T E N T A M E N T E
Excelencia en Educación Tecnológica®
La tecnología para el bien de la humanidad®

ARNOLDO DÍAZ RAMÍREZ
JEFE DE LA DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN

S.E.P.
INSTITUTO TECNOLOGICO DE MEXICALI
DIVISION DE ESTUDIOS DE
POSGRADO E INVESTIGACION
MEXICALI, B.C.

Ccp. Archivo

Agradecimientos

Estaré eternamente agradecida con mi director de tesis Arnoldo Díaz Ramírez por su apoyo, paciencia y absoluta dedicación. Gracias a mi esposo, quién me ha acompañado desde siempre y se ha dedicado a motivarme y ayudarme incondicionalmente durante todo el proceso de mi carrera y mi crecimiento personal. También quiero agradecer a mis padres por la vida y por enseñarme a vivirla. Y por último, pero no por eso menos importante, le agradezco a mi abuelo, por siempre apoyar mis desiciones, y quién se que desde un lugar muy especial del universo se sigue alegrando por cada uno de mis logros.

Resumen

La industria 4.0 (I40) pretende, a través de tecnologías como sistemas ciberfísicos (CPS - cyber physical systems) y el Internet de las cosas (IoT - Internet of Things), crear una producción inteligente, descentralizada y autónoma con sistemas trabajando en tiempo real, con la finalidad de brindar al cliente un servicio lo mas personalizado posible. Por lo tanto, definir los estándares de comunicación más eficientes, se vuelve algo fundamental al momento de querer construir un sistema con estas características. Recientemente se han propuesto protocolos con prestaciones de tiempo-real estricto. Sin embargo, no existen muchos estudios comparativos de su desempeño ni se ha experimentado que tan superiores son con respecto a los protocolos de redes de propósito general. El desarrollo de este trabajo busca presentar el resultado de un estudio comparativo y demostrativo de los protocolos de red más importantes de la actualidad que pueden ser utilizados para el desarrollo de aplicaciones IoT en el contexto de la I40.

Índice general

Índice de figuras

Capítulo 1

Introducción

El término Industria 4.0 (I40) hace referencia a la cuarta revolución tecnológica por la que está pasando actualmente la industria. La I40 pretende, a través de tecnologías como sistemas ciberfísicos (CPS - cyber physical systems) y el Internet de las cosas (IoT - Internet of Things), crear una producción inteligente, descentralizada y autónoma con sistemas trabajando en tiempo real, con la finalidad de brindar al cliente un servicio lo mas personalizado posible [40].

Las aplicaciones IoT cumplen con la tarea de la recopilación y evaluación integral de datos de diferentes fuentes (equipos y sistemas de producción, así como sistemas de gestión), en el contexto de la I40, se convertirán en estándares para soportar la toma de decisiones en tiempo real [44]. Además, con el aumento de la conectividad y el uso de protocolos de comunicaciones estándar que vienen con ello, la necesidad de proteger los sistemas industriales críticos y las líneas de fabricación de las amenazas aumenta dramáticamente [44]. Por lo tanto, definir los estándares de comunicación más eficientes, se vuelve algo fundamental al momento de querer construir un sistema con estas características. Los CPS son una herramienta muy útil dentro de estas aplicaciones ya que su núcleo computacional es un sistema de tiempo-real estricto, esto significa que es completamente necesario que produzca respuestas correctas dentro del intervalo de tiempo definido, si el tiempo de respuesta excede el límite, se produce un funcionamiento erróneo en el sistema [4].

La red de comunicaciones del CPS también debe ofrecer prestaciones de tiempo-real estricto. Recientemente se han propuesto protocolos con prestaciones de tiempo-real estricto, tales como Real-Time Ethernet, WirelessHART o ISA 100, que en teoría son mas adecuados para ser utilizados en los CPS y en aplicaciones IoT de la I40. Sin embargo, no existen muchos estudios comparativos de su desempeño ni se ha experimentado que tan superiores son con respecto a los protocolos de redes de propósito general. Por otra parte, es importante conocer qué soporte existe para su uso en SCP (Secure Copy Protocol), su facilidad de instalación, el soporte de bibliotecas para el desarrollo de aplicaciones, entre otros parámetros [42].

El desarrollo de este trabajo busca presentar el resultado de un estudio comparativo de los protocolos de red más importantes de la actualidad que pueden ser utilizados para el desarrollo de aplicaciones IoT en el contexto de la I40.

1.1. Pregunta de investigación

¿Cuál es el desempeño y soporte de los protocolos de comunicación inalámbrica para trabajar con aplicaciones del Internet de las Cosas?

1.2. Objetivos generales

Evaluar los protocolos de redes de computadoras más importantes, para poder dar una visión más amplia sobre las características de cada uno de estos y evaluar si pueden ser utilizados en el contexto de alguna aplicación del Internet de las Cosas y cuál es el mas apto para esto.

1.3. Objetivos específicos

- Definir cuales son las características con las que debe cumplir un protocolo para que pueda ser utilizado en aplicaciones del Internet de las Cosas.
- Realizar experimentos para dar a conocer disponibilidad, soporte y configuración de los protocolos.

- Evaluar los protocolos elegidos para determinar sus características y comprobar su funcionamiento dentro de una aplicación del Internet de las Cosas.

Capítulo 2

Trabajos relacionados

Algunos de los investigadores han examinado distintos protocolos, arquitecturas y redes dentro del contexto del internet de las cosas con el objetivo de comparar sus características.

Con la finalidad de revisar diferentes protocolos de comunicación en IoT, los autores del artículo Internet of Things (IoT) Communication Protocols : Review [3] realizaron una comparacion, haciendo énfasis en las principales características y comportamientos de varias métricas como: consumo de energía, seguridad y velocidad de transmisión de datos. El estudio tiene como objetivo presentar pautas para los investigadores y así poder seleccionar el protocolo correcto para diferentes aplicaciones IoT. Los protocolos que se sometieron a estudio son: 6LoWPAN, ZigBee, Bluetooth, RFID, NFC y Z-Wave. Esta investigación arrojó un resultado interesante, se expone que 6LoWPAN será el protocolo del futuro, principalmente porque permite que una gran cantidad de dispositivos inteligentes se implementen a través de Internet fácilmente mediante el uso del espacio de direcciones de IPv6 para la recopilación de datos e información.

Los investigadores y desarrolladores del artículo Comparison of Communication Protocols for Low Cost Internet of Things Devices [26] presentaron una revisión de los protocolos de comunicación alámbricos e inalámbricos más comunes, analiza sus características, ventajas y desventajas, además realizó un estudio comparativo para elegir la mejor red de sensores bidireccionales compuesta por dispositivos de bajo consumo como Arduino, ESP-12 y Raspberry Pi. El objetivo principal era ver si los protocolos de comunicación pueden funcionar con estas plataformas, qué tan bien funcionan al aumentar la distancia entre nodos y también comparar los resultados obtenidos con los teóricos. Para esto, se realizaron pruebas diferentes con el fin de ver los siguientes valores: Rendimiento, Retraso del mensaje y Eficiencia. Una conclusión relevante del estudio reveló que LoRa es una opción muy confiable para una aplicación inalámbrica, principalmente debido a la baja complejidad y costo necesarios y al hecho de que no interfiere con WiFi. Sin embargo, estas investigaciones estan más enfocadas en la integración de dispositivos que en la evaluación de protocolos inalámbricos.

Otros investigadores se han dedicado a estudiar y comparar protocolos de comunicación sin darle necesariamente un contexto dentro del internet de las cosas y más bien para dar una vista general de ellos.

A continuación se muestra un cuadro comparativo de otros trabajos que parecieron relevantes y que en su contenido se encuentran datos de las tecnologías de comunicación principalmente utilizadas en aplicciones del Internet de las Cosas, y que ayudaron a realizar la investigación actual, se hizo énfasis en resumir objetivos, métodos y escenarios de cada trabajo, y adicionalmente se mencionaron los datos y/o conclusiones relevantes de cada investigación.

Trabajo	Protocolos estudiados	Objetivos	Métodos	Escenarios	Datos relevantes

Integración de ZigBee/6loWPAN en una red de sensores inalámbrica [47].	ZigBee y 6LoWPAN.	Empresas como SAYME ofrecen productos en el ámbito de las redes de sensores inalámbricas basados en el estándar IEEE 802.15.4. Se pretende estudiar y desarrollar un sistema compatible con los actuales que implemente otras tecnologías estandarizadas como: ZigBee y 6LoWPAN.	Se analiza el funcionamiento de los protocolos, exponiendo ventajas y desventajas que conlleva al realizar una integración con hardware que funcione con el estándar IEEE 802.15.4 en el ámbito de sensores.	Se proponen cambios al hardware existente para llevar a cabo una integración con las tecnologías, tambien se propuso el diseño de una app de prueba.	Muestran especial interés en la dificultad que tiene el estándar 6LoWPAN para implementarlo en algún producto.
A Comparison of WirelessHART and ZigBee for Industrial Applications [36].	WirelessHART y ZigBee.	Se presentan las razones del por que ZigBee no se considera adecuada para usarla en la mayoría de las aplicaciones industriales, cosa que motivó el desarrollo de un nuevo estándar de comunicación que se ajusta a las necesidades de la industria: WirelessHART.	Se hace una exposición sobre las características más distinguidas de ambos protocolos: - Descripción general. - Funcionamiento Básico. - Seguridad.	Los protocolos se comparan en áreas de interés para aplicaciones industriales.	La comparación muestra que WirelessHART aborda muchas de las debilidades por las que ha sido criticado ZigBee y por lo tanto tiene potencial de éxito en las aplicaciones industriales.

A Survey of ZigBee Wireless Sensor Network Technology: Topology, Applications and Challenges [1].	ZigBee	Dar una visión general de ZigBee como una tecnología basada en una red de sensores inalámbrica, incluyendo su topología, aplicaciones y retos.	Durante la descripción de la tecnología, características de Wifi, Bluetooth y ZigBee se resumen y comparan en una tabla, tomando en cuenta: Aplicaciones, Bandas de frecuencia, Vida de la batería, Nodos por red, ancho de banda y rango.	ZigBee tiene una amplia variedad de áreas de aplicación, tales como automatización del hogar y de comercios, infraestructura para el ahorro inteligente de energía, cuidado de la salud, monitoreo y control en procesos industriales, control remoto para dispositivos electrónicos, entre otros.	Comparado con Wifi y Bluetooth, la pila del protocolo de ZigBee tiene un peso más ligero, tiene un mayor rendimiento de transmisión, ademas, si bien se informa que los dispositivos wifi tienen entre 12 y 8 horas de duración de batería y los dispositivos bluetooth una duración de batería de unos pocos días, muchos dispositivos ZigBee pueden tener una duración de batería hasta por 5 años.
Bluetooth and Wi-Fi wireless protocols: A survey and a comparison [21].	Wi-Fi y Bluetooth	Dar una vista general de los estándares de comunicación Wi-Fi y Bluetooth.	Compara las principales características y funcionamientos de ambos protocolos en términos de varias métricas, incluyendo capacidad, topología de red, seguridad, calidad del servicio de soporte y consumo de energía.	La comparación de estos protocolos se hizo tomando en cuenta que ambas tecnologías han sido desarrolladas para aplicaciones de corto alcance, tales como: comunicación en teléfonos móviles, computadoras portátiles, vehículos con asistente de viaje, etc.	Las áreas de investigación de los protocolos incluyen una eficiente solución para el problema de la estación oculta, **soporte a las transmisiones de tiempo real,** incrementar la seguridad en la transmisión de datos, entre otras.

Capítulo 3

Protocolos de redes de computadoras para Internet de las Cosas

3.1. Wi-Fi

3.1.1. Descripción

Según el artículo "A Review of Wireless Technology Usage for Mobile Robot Controller" (publicado en ICSEM 2012), Wi-Fi o WLAN (redes de área local inalámbricas) es una red inalámbrica basada en una serie de especificaciones del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) denominada 802.11. Wi Fi utiliza radiofrecuencia sin licencia, principalmente en la banda de 2,4 GHz. Permite que una persona con una computadora o dispositivo móvil con capacidad inalámbrica se conecte a Internet a través de un punto de acceso inalámbrico. La región geográfica cubierta por uno o varios puntos de acceso se denomina zona activa. El Wi-Fi estaba destinado a ser utilizado para dispositivos móviles y redes de área local, pero ahora se utiliza a menudo para el acceso a Internet en exteriores [34].

3.1.2. Historia

En el año de 1999, varias empresas (3Com, Airones, Intersil, Lucent Technologies, Nokia y Symbol Technologies) se unieron para formar una asociación sin fines de lucro a fin de mejorar la experiencia de usuario sin importar la marca de los dispositivos, utilizando una nueva tecnología de redes inalámbricas, por el año 2000 se le definió el término “Wi-Fi” para su trabajo técnico y anunció su nombre oficial como Wireless Ethernet Compatibility Alliance o WECA, pero al día de hoy es llamada Wi-Fi Alliance [24]. Actualmente Wi-Fi Alliance está conformada por cientos de empresas en muchos países y poseen una visión común: “Conectar a todos y a todo”, por lo que se dedican a impulsar nuevas tecnologías y aplicaciones para el uso del Wi-Fi [6].

3.1.3. Organismo que lo desarrolla o regula

Wi-Fi Alliance, la cual es una organización que cumple con los estándares 802.11 establecidos por la IEEE [25].

3.1.4. Principales características

- El estándar 802.11n de la IEEE es casi sinónimo de Wi-Fi , mejora respecto a sus versiones anteriores, en la actualidad, la Wi-Fi Alliance decidió hacer lo mismo que bluetooth, este estándar se llama 802.15.1 oficialmente pero es mejor conocido como bluetooth, entonces sus nuevas versiones son bluetooth 5.0, ahora la Wi-Fi Alliance utilizara Wi-Fi 6 para 802.11ax, Wi-Fi 5 para 802.11ac y Wi-Fi 4 para el 802.11n [23].
- “La norma 802.11 sigue el mismo modelo o arquitectura que toda la familia 802, es decir: capa física y capa enlace”, si se mira a mas detalle la subcapa inferior, PMD (Physical Media Dependent), que corresponde al conjunto de especificaciones de cada uno de los sistemas de transmisión a nivel físico, el estándar define cuatro: Infrarrojos, FHSS, DSSS y OFDM. La subcapa superior, PLCP (Physical Layer Convergence Procedure), se encarga de adaptar las diversas especificaciones de la subcapa PMD a la subcapa MAC, inmediatamente superior [9].
- La velocidad máxima teórica del estándar 802.11ax es de 2.4 Gbps manteniendo una frecuencia entre 2.4 y 5 GHz, alcance <100 metros. Estándar en IoT: Wi-Fi Halow es una designación para productos que incorporan

la tecnología IEEE 802.11.ah. Mejora el Wi-Fi al trabajar en un espectro por debajo de 1GHz con más alcance y conectividad de bajo consumo energético ideal para dispositivos IoT [39].

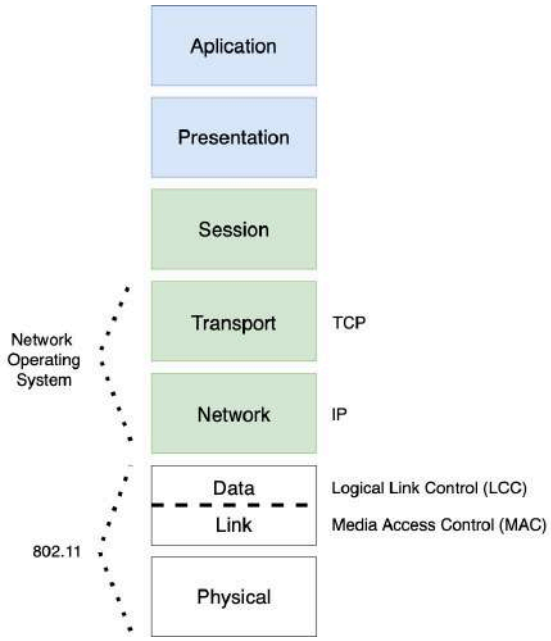

Figura 3.1.1: Modelo OSI y la familia IEEE 802.11

3.1.5. Topologías

Toda red inalámbrica compleja está compuesta por la combinación de una o más tipos de conexiones:

- Punto a punto [52].
- Punto a multipunto [52].
- Multipunto a multipunto [52].

Las redes inalámbricas WiFi contempla tres topologías o configuraciones distintas:

- Modo infraestructura o BSS: "Contrario al modo ad hoc donde no hay un elemento central, en el modo de infraestructura hay un elemento de "coordinación": un punto de acceso o estación base [52]."
- Modo ad hoc o IBSS: "También conocido como punto a punto, es un método para que los clientes inalámbricos puedan establecer una comunicación directa entre sí [52]."
- Modo ESS: "Se trata de un conjunto de BSS conectados mediante un sistema de distribución. Los puntos de acceso se comunican entre sí para permitir que las estaciones puedan pasar de un BSS a otro sin perder la comunicación, servicio denominado roaming [8]."

3.1.6. Bibliotecas para el desarrollo de aplicaciones

- Biblioteca Wifi para arduino: Para que una placa arduino se conecte a internet tiene que hacer uso de un elemento llamado "Arduino WiFi Shield" y para manipularlo se hace uso de la WiFi Library("#include <WiFi.h>") [10].
- WifiManager: es una clase que provee la principal API para manejar todos los aspectos relacionados la conectividad WiFi con java [49].
- Python-wifi: Es un módulo de python que proporciona acceso de lectura y escritura a las capacidades de una tarjeta de red inalámbrica usando las extensiones inalámbricas de linux, para implementarlo utilice "pip instalar python-wifi [14]".
- WifiUtils: es una biblioteca que proporciona un conjunto de métodos convenientes para administrar el estado de WiFi, el escaneo de WiFi y la conexión de WiFi a puntos de acceso [22].

3.2. Bluetooth

3.2.1. Descripción

Bluetooth es una norma industrial que se utiliza en redes inalámbricas de área personal y permite la transmisión de voz y otros datos entre diferentes tipos de dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. [17].

3.2.2. Historia

El artículo "Bluetooth", publicado en la enciclopedia EcuRed, cuenta que la tecnología fue nombrada así en homenaje al rey danés Harald Blatand (Harold Bluetooth en inglés) conocido por ser un gran comunicador y por haber unificado los pueblos de Dinamarca, Noruega y Suiza. Bluetooth fue creado para comunicar dos tecnologías diferentes, las computadoras y los teléfonos móviles. En el año 2000 el Bluetooth SIG publicó el logo como la fusión de dos letras del alfabeto rúnico, hagall y berkana que representan la H y la B, las iniciales del rey. En el año de 1994 Ericsson tenía la necesidad de una conexión entre dispositivos, con una interfaz vía radio y de bajos recursos, se llegó a un sistema basado en la comunicación por radio de corto alcance, llamado Mclink, en el año 1998 nació el Bluetooth Special Interest Group(SIG), promovido por las empresas Ericsson, Nokia, Toshiba, IBM e Intel, al agrupar las empresas líderes de las comunicaciones se llegó al éxito del Bluetooth [17].

3.2.3. Organismo que lo desarrolla o regula

Bluetooth Special Interest Group (SIG) es quien regula Bluetooth, en la actualidad SIG está conformado por más de 36000 empresas que buscan unificar, armonizar e impulsar la innovación en el vasto rango de dispositivos conectados que nos rodean [12].

3.2.4. Principales características

En la figura se muestra un diagrama de la pila de protocolos de Bluetooth. La pila se compone por protocolos específicos de Bluetooth, así como del protocolo de búsqueda de servicios SDP, o de otros protocolos adoptados [41].

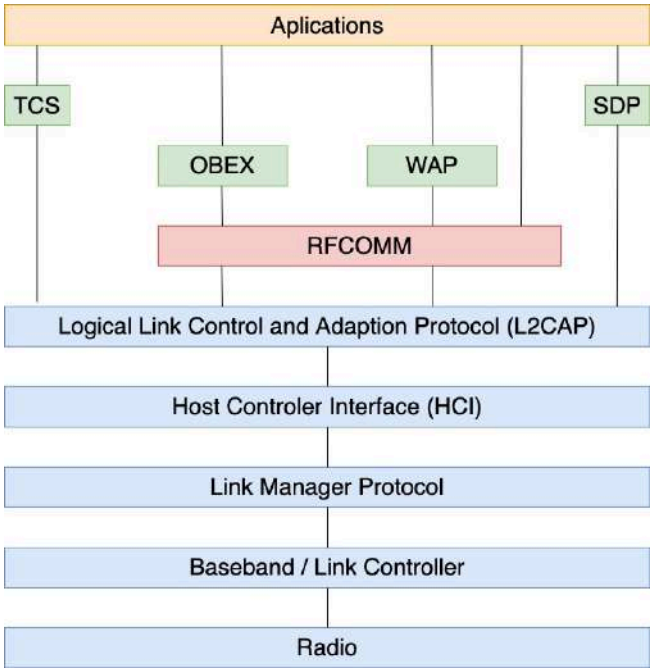

Figura 3.2.1: Pila de protocolos de Bluetooth

- El protocolo LMP (protocolo de gestión del enlace) se hace cargo de la configuración y establecimiento del enlace entre los dispositivos [41].
- La HCI aporta una interfaz con el módulo radio, con el gestor de enlace y con el controlador de la banda base [41].

- El protocolo L2CAP (protocolo de control del enlace lógico y adaptación) separa a las capas superiores de los detalles de los protocolos de capas inferiores [41].
- SDP proporciona a las aplicaciones un medio para realizar búsquedas de servicios y de sus características [41].
- RFCOMM provee una emulación de un puerto serie sobre L2CAP concediendo el mecanismo de transporte a servicios de capas más altas, permitiendo realizar múltiples conexiones con un mismo dispositivo al mismo tiempo [41].
- El TCS binary (protocolo de control telefónico), determina la señalización de control de llamadas para el establecimiento de envío de datos entre dispositivos Bluetooth [41].

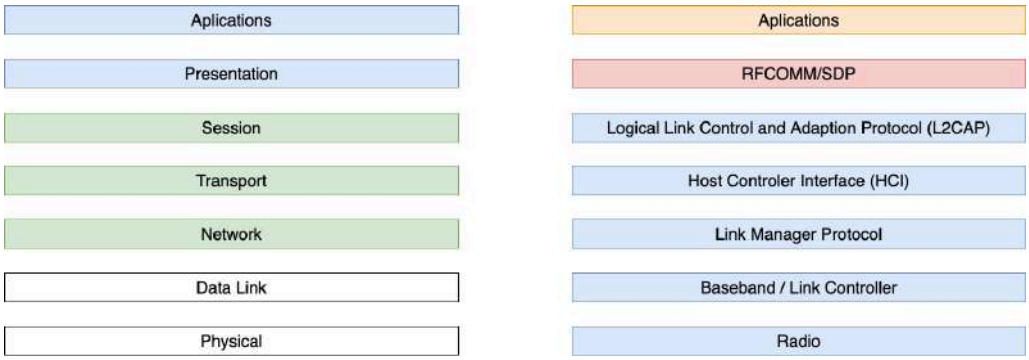

Figura 3.2.2: Comparativa del modelo OSI y los protocolos de Bluetooth

3.2.5. Topologías

Existen dos topologías de las redes Bluetooth: punto-a-punto y punto-a multipunto. Los dispositivos, se comunican en redes denominadas piconets. Estas redes tienen la posibilidad de crecer y tener hasta 8 conexiones de punto a punto. Además, se puede extender la red mediante la formación de scatternets, el cual es la red producida cuando dos dispositivos pertenecientes a dos piconets diferentes, se conectan. En una piconet, un dispositivo actua como master, este envía la información del reloj (para sincronizar procesos) y la información de los saltos de frecuencia. El resto de los dispositivos actúan como slaves (esclavos), los cuales son dispositivos que pueden participar en mas de una red, cuando un dispositivo no esta conectado a ninguna red, se le dice que su estado es Stand by, ya que se encuentra en espera de conectarse a alguna, si un dispositivo esta conectado a la red, pero no esta transmitiendo datos, se dice que su estado es hold [41].

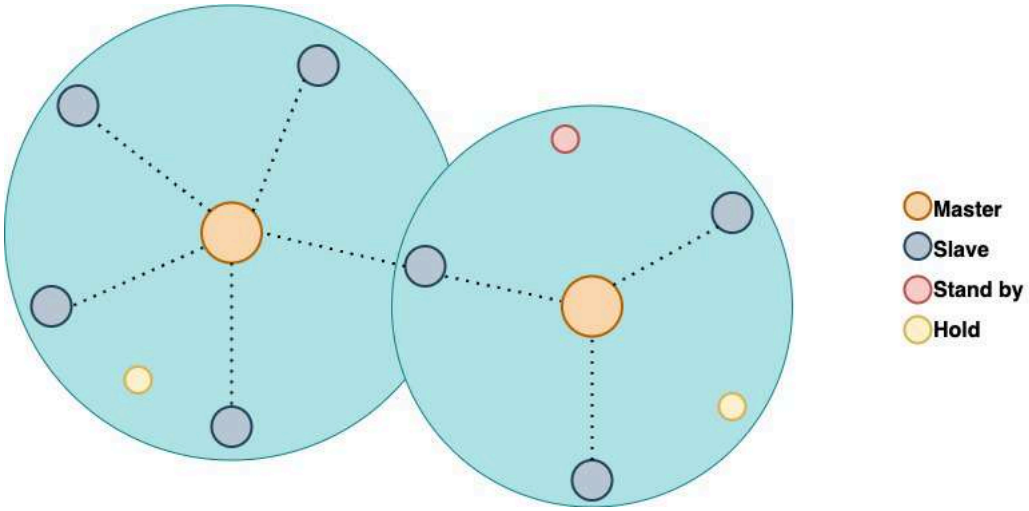

Figura 3.2.3: Topología Bluetooth

3.2.6. Bibliotecas para el desarrollo de aplicaciones

- Bluez: Se trata de una biblioteca que se utiliza para emitir comandos y escuchar eventos de los controladores bluetooth para linux [50].
- Node-bluetooth: Es una biblioteca que permite la comunicación de puerto serial bluetooth para Node.js, funciona en Linux y se necesita tener libbluetooth-dev (En Ubuntu/Debian: "$ sudo apt-get install libbluetooth-dev") para instalar node-bluetooth use la siguiente instrucción: "$ npm install node-bluetooth --save" [38].
- BluetoothManager: Es una biblioteca que facilita el uso de bluetooth en las aplicaciones de windows 10 [37].
- Android.bluetooth: La plataforma de android incluye compatibilidad con la pila de red Bluetooth, la cual permite que un dispositivo intercambie datos de manera inalámbrica con otros dispositivos Bluetooth [16].

3.3. ZigBee

3.3.1. Descripción

Zigbee es un estándar de comunicaciones inalámbricas diseñado por la Zigbee Alliance. Es un conjunto de protocolos de comunicación para el intercambio de datos inalámbricos entre dispositivos de bajo coste, bajo consumo y baja tasa de datos [47].

3.3.2. Historia

La enciclopedia EcuRed establece que la idea de las redes de familia Zigbee fue concebida en el año 1998 donde quedó claro que Wi-Fi y Bluetooth no cubrirán todos los contextos, bajo la necesidad de redes descentralizadas. En el 2003 se aprobó el estándar de la IEEE 802.15.4, por lo cual Zigbee Alliance alcanzó sus objetivos, y actualmente colabora para crear y desarrollar estándares abiertos para los dispositivos que utilizamos cada día. También aporta soluciones abiertas de internet de las cosas, principalmente en la domótica [18].

3.3.3. Organismo que lo desarrolla o regula

Zigbee fue desarrollado por Zigbee Alliance y actualmente es quien lo regula, fue establecida en el 2002 [7].

3.3.4. Principales características

Zigbee es un sistema ideal para redes domóticas, está diseñado para reemplazar la proliferación de sensores individuales, cubre las necesidades del mercado a un bajo costo, bajo consumo y de manera muy segura [46].

Está basado en el estándar IEEE 802.15.4 de redes inalámbricas, define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos. Opera en la banda libre de ISM 2.4 GHz para conexiones inalámbricas con un alojamiento de 16 bits a 64 bits de dirección extendida y 128-bit AES de cifrado por lo cual provee conexiones seguras entre dispositivos, además, su alcance es de 10 m a 75m [46].

3.3.5. Topologías

Zigbee permite tres topologías de red:

- Topología de estrella: En el centro se localiza el coordinador.
- Topología en árbol: La raíz del árbol es el coordinador .
- Topología de malla: Por lo menos uno de los nodos tiene más de dos conexiones. Hasta 65.000 nodos en una red (Las limitaciones físicas no permiten tantos nodos, por ej. ancho de banda o memoria) [2].

La topología de malla es la más interesante, ya que permite que si un nodo del camino falla y se cae, la comunicación pueda seguir entre todos los demás nodos gracias a que se vuelven a hacer los caminos [2].

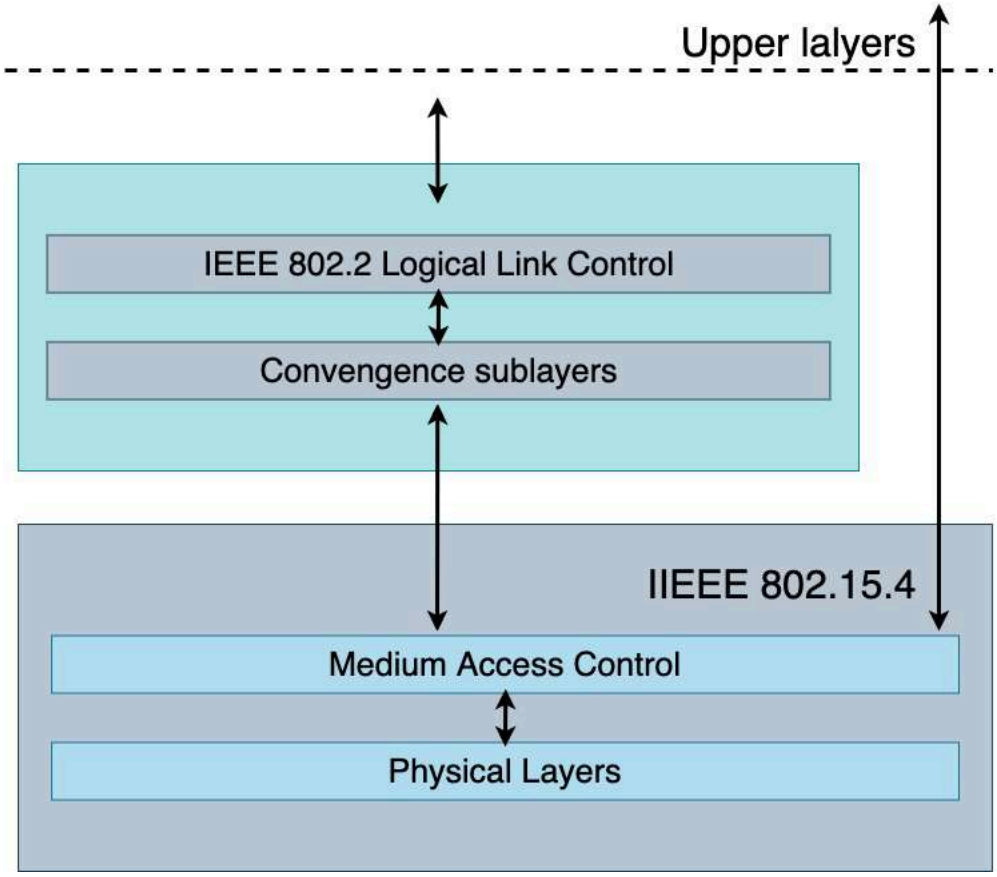

Figura 3.3.1: Pila de protocolos IEEE 802.5.4

3.3.6. Bibliotecas para el desarrollo de aplicaciones

- ZigBeeNet: biblioteca .Net multiplataforma para la comunicación entre dispositivos Zigbee, debido a que Zigbee es solo una especificación, necesita una pila de un fabricante que la implemente [30].
- XBee C# Library: API desarrollada en C# que le permite interactuar con los módulos de radiofrecuencia (RF), ofrecida por Digi International, empresa que ofrece soluciones para IoT. Posee dos módulos, XBeeLibrary.Core, contiene el código común para cualquier plataforma y XBeeLibrary.Xamarin, la cual contiene las API´s necesarias para desarrollar aplicaciones móviles multiplataforma [31].
- XBee Java Library: APi desarrollada en java que permite interactuar con los módulos de radiofrecuencia XBee de Digi International, también existe una biblioteca llamada XBee Android ofrecida igualmente por Digi International basada en XBee Java library, la cual hace énfasis en el desarrollo de aplicaciones móviles para android y ofrece soporte para diferentes interfaces de comunicación con dispositivos XBee por Bluetooth, usb o por el puerto serial [29].
- XBee Python: Esta es una biblioteca para python, la cual utiliza el módulo PySerial para comunicarse con los módulos de radio, también utiliza el módulo SRP para una autenticación entre dispositivos a través de Bluetooth de baja energía, ambos módulos se descargan de manera autónoma al instalar la biblioteca mediante: “pip install digi-xbee” [28].

3.4. LoRa

3.4.1. Descripción

LoRa (significa Long Range) es una tecnología inalámbrica que utiliza modulación en radiofrecuencia patentado por Semtech, usa una tecnología que se denomina Chirp Spread Spectrum (o CSS) [33].

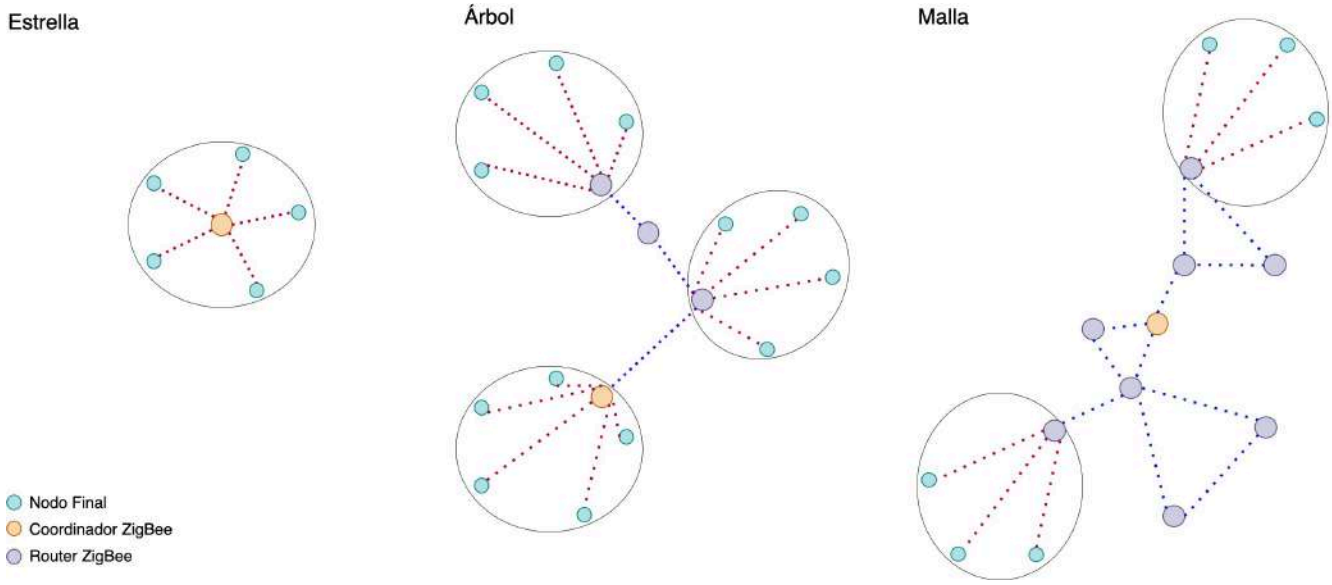

Figura 3.3.2: Topologías ZigBee

3.4.2. Historia

LoRa fue desarrollada por Semtech, proveedor de semiconductores analógicos. Hoy en día, LoRa es administrado por LoRa Alliance quien es una organización creada en el 2015 para admitir el protocolo LoRaWAN y garantizar la interoperabilidad de los productos que utilizan esta tecnología, actualmente posee más de 500 miembros, entre ellos IBM, Cisco y Semtech [33].

3.4.3. Organismo que lo desarrolla o regula

En la actualidad LoRa está administrada por la LoRa Alliance, quien certifica a todo fabricante de hardware que desee trabajar con esta tecnología. LoRa Alliance es una organización sin fines de lucro que se a convertido en una de las más grandes alianzas del sector tecnológico, comprometida a permitir el despliegue a gran escala con un bajo nivel energético en redes de área amplia (LPWAN) IO a través del desarrollo y la promoción de LoRaWAN [5].

3.4.4. Principales características

La tecnología de LoRa utiliza LoRaWAN, el cuál es un protocolo de red que se usa para redes de baja potencia y de áreas muy extensas, LPWAN (Low Power Wide Area Network) se usa para administrar y comunicar a los dispositivos LoRa. El protocolo LoRaWAN se compone de gateways y nodos [53]:

- Gateways (antenas): estos se encargan de recibir y enviar información a los nodos. Nodos (dispositivos): son dispositivos finales que envían y reciben información hacia el gateway [53].

LoRa posee una alta tolerancia a las interferencias, alta sensibilidad para recibir datos (-168dB), es basada modulación “chirp” y su bajo consumo energético rinde hasta 10 años con una sola batería [53].

- Alcance de 10 a 20 km [53].
- Baja transferencia de datos (hasta 255 bytes) [53].
- Conexión punto a punto [53].
- Frecuencias de trabajo: 868 Mhz en Europa, 915 Mhz en América, y 433 Mhz en Asia [53].

LoRa es una tecnología modelo para conexiones de extensas distancias y para redes de IoT en las que son necesarios sensores que no dispongan de corriente eléctrica cableada, algunas de las aplicaciones más comunes:

- Ciudades inteligentes [53].
- Lugares con poca cobertura (procesos agrícolas o ganaderos en el campo por ejemplo) [53].
- Redes privadas de sensores [53].

3.4.5. Topologías

Permite la interconexión entre objetos inteligentes sin la necesidad de instalaciones locales complejas, y además otorga amplia libertad de uso al usuario final, al desarrollador y a las empresas que quieran instalar su propia red para Internet de las Cosas (IoT) [53].

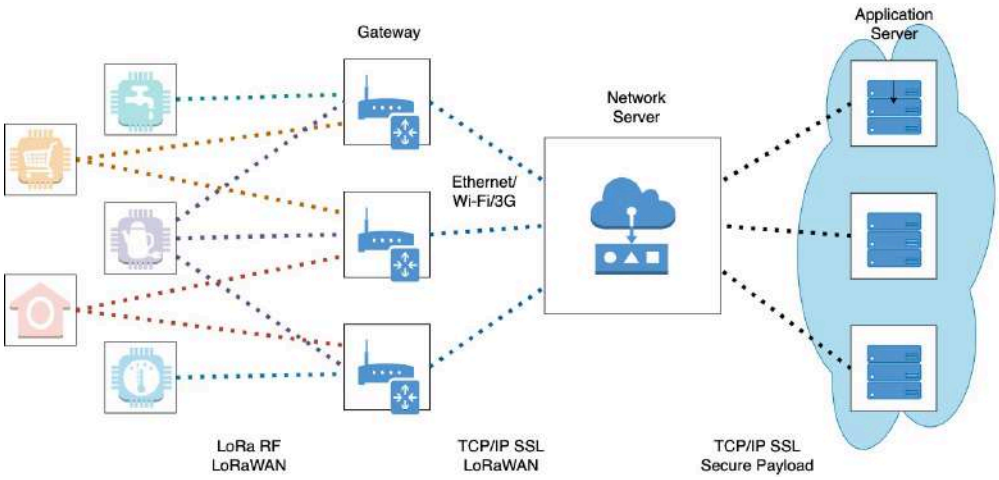

Figura 3.4.1: Topología LoRa

3.4.6. Bibliotecas para el desarrollo de aplicaciones

- Arduino-LoRa: Una biblioteca Arduino para enviar y recibir datos usando radios LoRa. Admite placas / shields basados en Semtech SX1276 /77/78/79 [45].
- pyLoRa: es una adaptación y una versión mejorada del trabajo mayeranalytic original (mayeranalytics / pySX127x) pyLoRa se puede usar para comunicarse con Arduino a través de la biblioteca RADIOHEAD, la cual es una biblioteca orientada a objetos para enviar y recibir mensajes empaquetados a través de una variedad de radios de datos comunes y otros transportes en una variedad de microprocesadores integrados [51].
- ESP32_LoRaWAN: Proporciona una implementación de LoRaWAN Clase A y Clase C bastante completa, este es un puerto de la biblioteca LoRaWAN del nodo LoRaMac de Semtech para el ESP32 y solo se puede utilizar para productos Heltec ESP32 LoRa (Se requiere licencia para su uso) [11].

3.5. 6LoWPAN

3.5.1. Descripción

6LoWPAN es un conjunto de estándares definidos por Internet Engineering Task Force (IETF), que construye y sostiene todo el trabajo y los estándares básicos de la arquitectura de Internet. Los estándares de 6LoWPAN hacen posible la utilización eficiente e ingeniosa de IPv6 sobre baja potencia, la baja velocidad de redes inalámbricas en dispositivos integrados simples a través de una capa de adaptación y la optimización de protocolos relacionados [19].

En la siguiente imagen se puede ver que 6loWPAN se encuentra entre las capas de enlace y red, pudiendo llegar hasta a la capa de transporte.

3.5.2. Historia

El grupo de trabajo IETF 6LoWPAN fue fundada en 2005. Durante toda la década de 1990 se presuponía que la ley de Moore progresaría en la computación y competencia de comunicación tan rápidamente que en muy poco tiempo cualquier dispositivo integrado podría posiblemente poner en práctica los protocolos IP. Aunque hasta cierto punto es cierto, y el IoT ha madurado rápidamente, no insistieron en las tecnologías de radio inalámbricas baratas y de bajo consumo o microcontroladores de baja potencia [20].

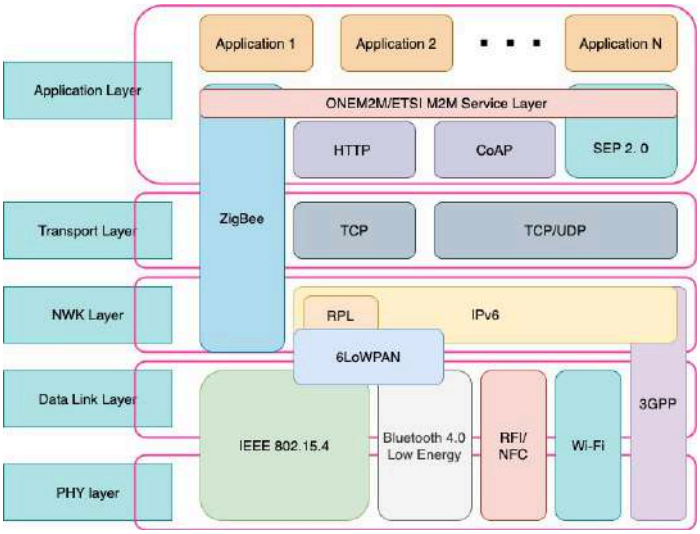

Figura 3.5.1: 6LoWPAN

3.5.3. Organismo que lo desarrolla o regula

6LoWPAN es regulado por el grupo de trabajo IETF(Internet Engineering Task Force), es una comunidad internacional abierta de investigadores, diseñadores, comercializadores y operadores de redes que se ocupan de la evolución del Internet y su buen funcionamiento [54].

3.5.4. Principales características

- IPV6 no fue desarrollado para operar en redes IEEE 802.15.4. La cantidad de información que se podría transportar sería muy pequeña y resultaría ineficiente por la sobrecarga de datos gracias a la transmisión de cabeceras. Por tal motivo, 6LoWPAN proporciona mecanismos de comprensión que permiten reducir su tamaño [47].
- 6LoWPAN contiene una función de fragmentación y reensamblaje que permite descomponer los paquetes IPv6 en fragmentos mas pequeños, y así puedan ser transportados de manera transparente para IPv6 [47].
- En las redes LoWPAN, el edge router es el encargado de encaminar el trafico dentro y fuera de la red. Integra dos interfaces distintas (IP y 6LoWPAN) para poder interactuar con dispositivos pertenecientes a distintos dominios y es, por tanto, el responsable del proceso de compresión de las cabeceras de red y transporte. Ademas, emplea una versión modificada del protocolo de descubrimiento de vecinos (ND: Neighbor Discovery) de IPv6 para la gestión y mantenimiento de la red [47].
- Durante el proceso de inicialización, el edge router es el encargado de distribuir el prefijo que los nodos emplean para generar sus direcciones de red. Para ello, responde a los RS (Router Solicitation) enviados por los nodos con RA (Router Advertisement). La respuesta del edge router, ademas de contener el prefijo, proporciona información adicional sobre la red. Una vez completado el proceso de autoconfiguración, todos los nodos de la red deberán registrarse en el edge router mediante el uso de los mensajes Neighbor Solicitation/Neighbor Advertisement (NS/NA). Con este proceso, el edge router lleva a cabo la detección de posibles direcciones duplicadas y almacena información sobre los nodos [47].
- No es necesario que los hosts estén siempre activos como los routers, por lo que es posible implementar en ellos estrategias de bajo consumo para el ahorro energético. No existe un único método para su gestión, cada uno de los fabricantes que implementa la tecnología 6LoWPAN propone sus propios mecanismos [47].

3.5.5. Topologías

Una red LowPAN es una colección de nodos que comparten el mismo prefijo de red. Existen tres tipos de arquitecturas de red:

1. LoWPAN-Simple: Red que dispone de un solo edge router [13].

2. LoWPAN-Extendida: Red con varios edge routers, habitual en aplicaciones de movilidad o con un gran número de nodos. Pueden adicionalmente formar una subred con otros dispositivos IP [13].
3. LoWPAN-ad-hoc: Red que no está conectada a internet. Uno de los routers se configura para que actúe como un edge router [13].

Una red 6LoWPAN cuenta principalmente con tres tipos de dispositivos:

- Edge-router: tambien conocido cómo dispositivo frontera, este se encuentra conectado a una red IPv6 y a una red LowPAN. Se encarga de intercambio de datos entre:
  - Dispositivos 6LoWPAN e Internet [13].
  - Dispositivos dentro de la red 6LoWPAN [13].
  - Generación y mantenimiento de la red [13].
- Router: Se encargan de direccionar los datos desde su destino hacia otro nodo dentro de la red [13].
- Host: Son dispositivos finales y no son capaces de efectuar tareas de enrutamiento de datos hacia otros dispositivos en red [13].

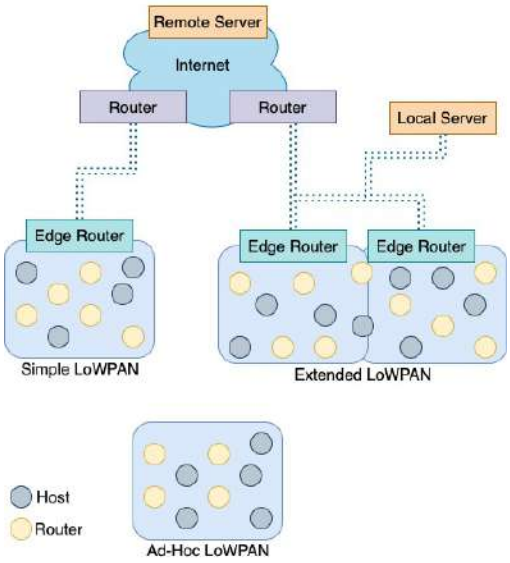

Figura 3.5.2: Topología 6LoWPAN

3.6. Wireless HART

3.6.1. Descripción

WirelessHART es un estándar (62591) global aprobado por la IEC que especifica una tecnología de malla interoperable auto-organizada en el cual los dispositivos de la red inalámbrica mitigan obstáculos en el ambiente del proceso, posee excelentes habilidades para el control y monitoreo, con una gran confiabilidad [43].

3.6.2. Historia

El estándar comenzó en el 2004 y fue desarrollado por 37 empresas de la fundación de comunicaciones HART (HCF), entre las empresas pertenecientes se encuentran: Emerson, Pepperl y Siemens. WirelessHART fue aprobada por la junta directiva de la fundación en el año 2007, posteriormente en el 2009 WirelessHART fue aprobado por la Comisión Electrotécnica Internacional (IEC) [15].

3.6.3. Organismo que lo desarrolla o regula

FieldComm Group es propietario de las especificaciones HART y proporciona servicios de registro de productos, capacitación y desarrollo de especificaciones asociados con la tecnología [27].

3.6.4. Principales características

El protocolo WirelessHART fue desarrollado para satisfacer las demandas de las aplicaciones de automatización industrial. El protocolo es totalmente compatible con dispositivos certificados HART anteriores y se puede utilizar para actualizar dispositivos cableados más antiguos a través de adaptadores. Utiliza la capa física IEEE 802.15.4 con el nivel global banda ISM de 2,4 GHz con licencia gratuita. La capa de red se basa en un mecanismo TDMA (acceso múltiple por división de tiempo) para arbitrar y coordinar las transacciones entre los nodos de la red [32].

3.6.5. Topologías

WirelessHART forma una red mallada plana en la que todos los dispositivos de campo forman una red. Todas las estaciones actúan simultáneamente como fuente de señal y como repetidor. El transmisor original envía un mensaje a su vecino más cercano, este a su vez transmite el mensaje hasta que llega a la estación base y al receptor. También se configuran rutas alternas en la fase de inicialización. Si el mensaje no se puede transmitir en una ruta concreta por causa de algún obstáculo o algún receptor defectuoso, el mensaje se transmite automáticamente por otra ruta. Además de ampliar el rango, este tipo de red proporciona rutas de comunicación redundantes que aumentan la fiabilidad de la transmisión [43].

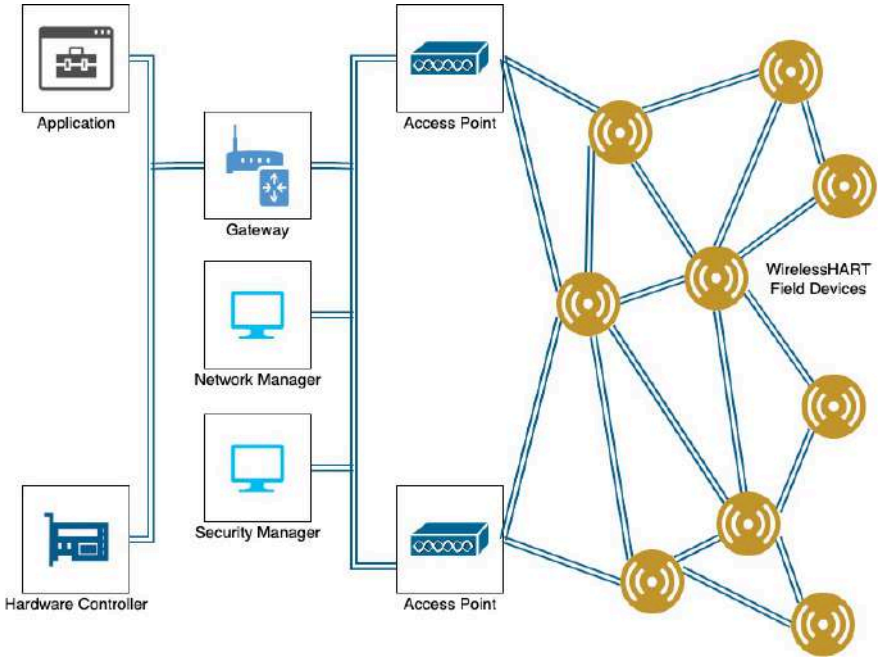

Figura 3.6.1: Topología de red mallada WirelessHART

3.6.6. Bibliotecas para el desarrollo de aplicaciones

- SmartMesh SDK: Smart Mesh SDK es un paquete de Python que simplifica la integración de una red SmartMesh IP o SmartMesh WirelessHART en su aplicación. Implementa la interfaz de programación de aplicaciones (API) del dispositivo al que está conectado. Se incluye un conjunto de aplicaciones de muestra en SmartMesh SDK, lo que permite al programador comprender rápidamente la API y usarla como parte de un sistema más grande [55].

Capítulo 4

Evaluación del desempeño de tecnologías de comunicación para IoT

Evaluar el jitter de una conexión es importante, ya que es una variación en la demora entre los paquetes enviados a través de esta. En pocas palabras, un jitter alto, introduce inconsistencias, lo cual interfiere con la calidad de la comunicación y la velocidad de la transferencia de datos. La tasa de paquetes perdidos es otro punto al que se le debe dar importancia, ya que, como su nombre lo dice, es el porcentaje de paquetes perdidos que hubo durante una conexión entre dos nodos.

En este proyecto, un conjunto de herramientas fue desarrollado para evaluar jitter y tasa de paquetes perdidos dentro de una conexión entre dos dispositivos. Se creó una aplicación cliente-servidor para cada uno de los protocolos escogidos a evaluar. Todas la aplicaciones fueron creadas con el lenguaje C, bajo el ambiente de Linux. Para algunos protocolos solo se necesitaron algunas librerías especializadas, mientras que en otros, como en LoRa, fue necesaria una integración.

A continuación se muestra una descripción detallada de la aplicación que se creo para cada protocolo.

4.1. Descripción, instalación y configuración

- El código fuente de las herramientas desarrolladas se encuentra publicado en Github:
  - https://github.com/floresBe/network_performance.
  - En el apéndice se puede encontrar una explicación básica cada una de las funciones principales de los programas Cliente y Servidor que se utilizaron.
- En todos los experimentos se utilizó por lo menos una computadora de escritorio con las siguientes características:
  - Sistema operativo Linux, Ubuntu 20.04.2 LTS.
    - Libreria GNU C instalada.
      - Instalación de GCC en Ubuntu:
        1. Actualizar lista de paquetes del sistema: sudo apt update.
        2. Instalar el paquete build-essential con el comando: sudo apt install build-essential.
        3. Para validar que el compilador de GCC se instaló correctamente, usar el comando gcc --version, el cual deberá de imprimir la versión de GCC.
    - Los datos obtenidos por las aplicaciones serán procesados para facilitar su uso utilizando la biblioteca json-c basada en JSON. Para la instalación de dicha biblioteca utilizamos el siguiente comando:
      - sudo apt-get install libjson-c-dev
  - Procesador Intel Core i3-2100 CPU @ 3.10GHz x4.
  - 6GB RAM.

4.1.1. Wi-fi (Paquetes UDP)

Descripción En la programación en C, un socket es un "enchufe" con el cual se crea una conexión con otro ordenador en la que se permite intercambiar datos dentro de una red. Para la evaluación de este protocolo, se creó una aplicación cliente-servidor, la cual, con la ayuda de la librería <sys/socket.h> (parte de la biblioteca GNU de C) , se utilizaron sockets de tipo UDP para crear conexiones entre dos nodos.

Instalación y configuración

1. Para realizar el experimento se utilizó el siguiente equipo:

   a) Una red Wi-Fi utilizando un modem/enrutador AC1750 de doble banda, inalámbrico, modelo Archer A7.

   b) Dos computadoras de escritorio con las características anteriormente mencionadas.

   - Tarjeta de red inalambrica Ubit PCI-E Wireless Wi-FI.
     - Condiciones de instalación: La tarjeta madre del CPU debe tener al menos un puerto PCI-E.
     - Funciona de inmediato en Linux sin necesidad de instalar controladores adicionales a las actualizaciones automáticas que el sistema sugiere.
     - La tarjeta tambien soporta conexiones de Bluetooth.

2. Para ejecutar los programas, primero es necesario compilar los archivos ClienteUDP.c y ServidorUDP.c, utilizando el comando gcc en una terminal:

   a) gcc ClienteUDP.c -o Cliente_UDP -ljson-c

   b) gcc ServidorUDP.c -o Servidor_UDP -ljson-c

3. Una vez compilados los programas, para ejecutarlos se utiliza el comando "./", no se deben de olvidar los parámetros

   a) ./Cliente_UDP server_name size_message number_packages

   b) ./Servidor_UDP

Figura 4.1.1: Tarjeta Wi-Fi/Bluetooth instalada en CPU.

Diseño de experimento

El programa cliente se define con la siguiente estructura:

1. Verifica argumentos.

   Debería de ejecutarse así: ./Client_UDP server_name size_message number_packages

2. Crea un socket con el protocolo para establecer la comunicación con el servidor.

   (sockfd = socket(AF_INET, SOCK_DGRAM, 0))

3. Crea un hilo para escuchar respuestas del servidor.

   pthread_create(&thread_listen_server, NULL, listen_server, NULL);

4. Envía al servidor el número de mensajes especificado en los argumentos.
   send_data(s);
5. Espera por un tiempo determinado
   while (seconds_elapsed < timeout * 1000)
6. Calcula Jitter y Tasa de paquetes perdidos con los datos de los paquetes recibidos.
   jitter = get_average();
   rate = abs((((float) count_packages_received/(float)number_packages)* 100) - 100);
7. Crea archivos de texto para graficar con gnu.
   create_gnu_files(rate,jitter);

El programa servidor se define con la siguiente estructura:

1. Crea un socket con el protocolo para poder establecer comunicación con algún cliente.
   (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
2. Escucha hasta que un cliente le envía información a través de la conexión.
   recvfrom(sockfd, (char *)message_json, MAXLINE, MSG_WAITALL, (struct sockaddr *) &cliaddr, &len);
3. Al recibir datos, agrega la hora actual en el mensaje y lo envía de vuelta al cliente.
   sendto(sockfd, (char *)message_json, MAXLINE, 0, (const struct sockaddr *) &cliaddr, len);

Ejecutar el programa servidor en una de las computadoras inicia el experimento, después de ejecutarse se queda a la espera de algún cliente que quiera enviarle información; a continuación se ejecuta el programa cliente en la otra computadora utilizando los parámetros: *server_ name size_ message number_ packages*, se crea la conexión y comienza el envío de mensajes. Al finalizar, el cliente creara reportes con la información obtenida del jitter y tasa de paquetes perdidos durante la conexión.

Figura 4.1.2: Descripción gráfica de las conexiones en el experimento Wi-Fi.

4.1.2. Bluetooth

Descripción Para evaluar el protocolo de comunicación Bluetooth, se realizó una aplicación que se conforma por dos archivos .c, Server_Bluetooth.c (Servidor) y Client_Bluetooth.c (Cliente). Ambos utilizan la librería BlueZ, una potente pila de comunicaciones Bluetooth con amplias APIs que permite al usuario aprovechar al máximo todos los recursos Bluetooth locales.

Instalación y configuración

1. Para realizar el experimento se utilizo el siguiente equipo:
   a) Dos computadoras de escritorio con las características anteriormente mencionadas.
   - Tarjeta de red inalambrica Ubit PCI-E Wireless Wi-FI/Bluetooth.
     - Condiciones de instalación: La tarjeta madre del CPU debe tener al menos un puerto PCI-E.
     - Funciona de inmediato en Linux sin necesidad de instalar controladores adicionales a las actualizaciones automáticas que el sistema sugiere.
     - La tarjeta soporta Wi-Fi y Bluetooth.
   - Librería BlueZ instalada.

- Descargar la librería de codigo abierto en el siguiente link, donde actualmente se describen las instrucciones de instalación:
  - https://github.com/bluez/bluez
- Para compilar las utilidades de Bluetooth, Linux necesita los siguientes paquetes:
  - Biblioteca GLib, para instalar correr el comando sudo apt install libglib2.0-dev en una terminal.
  - Biblioteca D-Bus, para instalar correr el comando sudo apt install dbus en una terminal.
- Para configurar, ejecutar en una terminal el comando: ./configure --prefix=/usr --mandir=/usr/share/man \ --sysconfdir=/etc --localstatedir=/var
  - ./configure busca automáticamente todos los componentes y paquetes necesarios.
- Para compilar e instalar BlueZ, posicionarse en la carpeta principal en la que se descargó BlueZ desde la terminal y correr el comando: make && make install.

2. Para ejecutar los programas, primero es necesario compilar los archivos Cliente_Bluetooth.c y Servidor_Bluetooth.c, utilizando el comando gcc en una terminal:

   a) gcc Cliente_Bluetooth.c -o Client_Bluetooth -ljson-c -lbluetooth

   b) gcc Servidor_Bluetooth.c -o Servidor_Bluetooth -ljson-c -lbluetooth

3. Una vez compilados los programas, para ejecutarlos se utiliza el comando "./", no se deben de olvidar los parámetros:

   a) ./Cliente_Bluetooth device_name size_message number_packages

   b) ./Servidor_Bluetooth

Figura 4.1.3: Manual de instalación de la tarjeta Ubit.

Diseño de experimento

El programa cliente se define con la siguiente estructura:

1. Verifica argumentos.

   Debería de ejecutarse así: ./Client_Bluetooth device_name size_message number_packages

2. Crea un socket con el protocolo de Bluetooth para establecer la comunicación con el servidor.

   socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);

3. Inicializa una búsqueda para localizar al dispositivo bluetooth servidor.

   Si lo encuentra, se conecta con el. connect(s, (struct sockaddr *)&address, sizeof(address));

4. Crea un hilo para escuchar respuestas del servidor.

   pthread_create(&thread_listen_server, NULL, listen_server, (void *)&listen_server);

5. Envía al servidor el número de mensajes especificado en los argumentos.
   send_data(s);
6. Espera por un tiempo determinado.
   while (seconds_elapsed < timeout * 1000)
7. Calcular Jitter y Tasa de paquetes perdidos con los datos de los paquetes recibidos.
   jitter = fabs(get_average());
   rate = abs((((float) count_packages_received/(float)number_packages)* 100) - 100);
8. Crea archivos de texto para graficar con gnu.
   create_gnu_files(rate,jitter);

El programa servidor se define con la siguiente estructura:

1. Crea un socket con el protocolo de Bluetooth para establecer la comunicación con el cliente.
   socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
2. Escucha hasta que un cliente pide establecer conexión.
   listen(s, 1);
3. Espera por datos del cliente.
   ba2str(&rem_addr.rc_bdaddr, buf);
4. Al recibir datos, agrega la hora actual en el mensaje y lo envía de vuelta al cliente.
   void callback_function(char buf[MAXLINE], int client);

Ejecutar el programa servidor en una de las computadoras inicia el experimento, después de ejecutarse se queda a la espera de algún cliente que quiera enviarle información; a continuación se ejecuta el programa cliente en la otra computadora incluyendo los parámetros: *device_ name size_ message number_ packages*, se crea la conexión y comienza el envío de mensajes. Al finalizar, el cliente creara reportes con la información obtenida del jitter y tasa de paquetes perdidos durante la conexión.

Figura 4.1.4: Descripción gráfica de las conexiones en el experimento Bluetooth.

4.1.3. ZigBee

Descripción. Para medir la calidad de un enlace de red de dispositivos ZigBee (Digi XBee®) , se realizó una aplicación en lenguaje C que se conforma por dos archivos: Server_ZigBee.c (Servidor) y Client_ZigBee.c (Cliente). Cada programa se comunica con un dispositivo XBee® a través de un puerto serial/USB con la ayuda de la librería de código abierto <xbee.h>.

Los dispositivos se deben configurar según la función a ejecutar y para esto se utiliza XCTU, una aplicación diseñada para interactuar con los módulos RF de Digi.

Instalación y configuración

1. Para realizar el experimento, se utilizó el siguiente equipo:

   a) Dos computadoras de escritorio con las características anteriormente descritas.

   - Aplicación XCTU instalada:
     - El acceso a los puertos serie y USB en Linux está restringido a los usuarios del grupo root y dialout de forma predeterminada. Para acceder a sus dispositivos XBee y usar XCTU para comunicarse con ellos, se requiere que el usuario de Linux pertenezca a este grupo. Para agregar el usuario de Linux al grupo se debe de ejecutar el siguiente comando en una terminal:
       - sudo usermod -a -G dialout <user> , donde <user> es el usuario al que se quiere agregar al grupo.
     - Reiniciar el sistema es necesario para que los cambios se apliquen.
     - Ir a la pagina www.digi.com/xctu y dar click en Descargar.
     - Click en el link de Linux.
     - Una vez completada la descarga, ejecutar el archivo ejecutable y seguir los pasos del asistente de XCTU.
     - Cuando se completa la instalación, aparece un cuadro de diálogo "Novedades" donde se pueden revisar las nuevas características de XCTU.
   - Biblioteca libxbee instalada.
     - Clonar el proyecto: git clone https://github.com/attie/libxbee3.git
     - Moverse dentro de la carpeta del proyecto acabado de descargar y se crea el archivo de configuración con: make configure.
     - Compilar el proyecto: make all.
     - Finalmente realizar la instalación de la biblioteca: sudo make install.

2. Para configurar los dispositivos ZigBee, se necesitan conectar por puerto USB y abrir la aplicación XCTU.

3. El siguiente paso es agregar el dispositivo a la lista de XCTU:

   a) Clic en el botón Descubrir módulos de radio.

   - En el cuadro de diálogo, seleccionar los puertos serie en los que desea buscar dispositivos y hacer click en Siguiente.
   - En la ventana Establecer parámetros de puerto, mantener los valores predeterminados y hacer click en Finalizar.
   - A medida que XCTU localiza dispositivos, aparecen en el cuadro de diálogo.
   - Hacer clic en Agregar dispositivos seleccionados una vez que el proceso de descubrimiento haya finalizado.
   - Se debe ver algo como esto en la sección Módulos de radio:

Figura 4.1.5: Dispositivos añadidos.

4. Una vez agregado el dispositivo, se debe configurar segun su función dentro de la aplicación:

   a) Seleccionar de la lista el dispositivo para ver los datos de su configuración, asegurarse de que cumpla con los siguientes parámetros:

- Configuración del dispositivo cliente:
  - ID, puede ser un número aleatorio pero deberá ser el mismo para ambos dispositivos, tanto «cliente» como «servidor».
  - JV, activar la función de Verificación de canal, esto sirve para que el dispositivo verifique si existe un dispositivo «servidor» válido en ese canal.
  - AP, asegurarnos de que el dispositivo está en modo API.
  - P4, deberá de estar en modo de entrada para datos para que el dispositivo reciba las lecturas tomadas del sensor.
- Configuración del dispositivo servidor:
  - ID, como ya se mencionó anteriormente, éste parámetro podrá ser un número aleatorio pero deberá ser el mismo para ambos dispositivos.
  - JV, la función de Verificación de canal no es necesario que este activada en este dispositivo.
  - CE, deberá estar activada la función de Coordinador.
  - AP, el modo API deberá estar activado.
  - P4, tampoco es necesario que esté activada la entrada de datos.
  - SC, si el CH (operating channel) no es el mismo, SC(Scan Channels) deberá permitir todos los canales con el valor FFFF.

5. Para ejecutar los programas, primero es necesario compilar los archivos Cliente_ZigBee.c y Servidor_ZigBee.c, utilizando el comando gcc en una terminal:

   a) gcc Cliente_ZigBee.c -o Client_Bluetooth -ljson-c

   b) gcc Servidor_ZigBee.c -o Servidor_Bluetooth -ljson-c

6. Una vez compilados los programas, para ejecutarlos se utiliza el comando "./", no se deben de olvidar los parámetros

   a) ./Cliente_ZigBee USB_port_number size_message number_packages

   b) ./Servidor_ZigBee

Figura 4.1.6: Dispositivo Digi XBee®.

Diseño de experimento

El programa cliente se define con la siguiente estructura:

1. Verifica argumentos.

   Debería de ejecutarse así: ./Cliente xbee_address, USB_port_number, size_message, number_packages;

2. Configura la conexión con el dispositivo físico.
   struct xbee * configure_xbee(struct xbee *xbee, xbee_err ret);
3. Configura la conexión con el dispositivo remoto (servidor).
   struct xbee_con * connection_xbee(struct xbee *xbee, struct xbee_con *con, xbee_err ret);
4. Crea un hilo para escuchar respuestas del servidor.
   pthread_create(&thread_listen_server, NULL, listen_server, (void *)&server_zigBee);
5. Envía al servidor el número de mensajes especificado en los argumentos.
   void callback_function(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt **pkt, void **data);
6. Espera por un tiempo determinado.
   while (seconds_elapsed < timeout * 1000)
7. Calcula Jitter y Tasa de paquetes perdidos con los datos de los paquetes recibidos.
   jitter = fabs(get_average());
   rate = abs((((float) count_packages_received/(float)number_packages)* 100) - 100);
8. Crea archivos de texto para graficar con gnu.
   create_gnu_files(rate,jitter);

El programa servidor se define con la siguiente estructura:

1. Configura la conexión con el dispositivo físico.
   struct xbee * configure_xbee(struct xbee *xbee, xbee_err ret);
2. Configura la conexión con el dispositivo remoto (cliente).
   struct xbee_con * connection_xbee(struct xbee *xbee, struct xbee_con *con, xbee_err ret);
3. Espera por datos del cliente indeterminadamente.
   while (1) { receive_data(struct xbee xbee, struct xbee_con con, xbee_err ret); }
4. Al recibir datos, agrega la hora actual en el mensaje y lo envía de vuelta al cliente.
   void callback_function(struct xbee *xbee, struct xbee_con *con, struct xbee_pkt **pkt, void **data);

Ejecutar el programa servidor en una de las computadoras inicia el experimento, después de ejecutarse este se conecta con el dispositivo (previamente conectado y configurado), se crea la conexión con el cliente y se queda en espera de algún mensaje de este. Al ejecutar el cliente, de igual manera se conecta con el dispositivo, después configura la conexión con el servidor, una vez establecida la conexión comienza el envío de mensajes. Al finalizar, el cliente creara reportes con la información obtenida del jitter y tasa de paquetes perdidos durante la conexión.

Figura 4.1.7: Descripción gráfica de las conexiones en el experimento ZigBee.

4.1.4. Lora

Descripción La aplicación para evaluar al protocolo LoRa está compuesta por tres programas, dos de ellos: Sender y Receiver, están desarrollados para trabajar con placas Arduino UNO en conjunto con un módulo Ra-01 LoRa desarrollado por la compañía Ai-Thinker y con la ayuda de la librería <LoRa.h>, el tercer programa se desarrolló en el lenguaje C, el cual, a través de un puerto serial, se comunica con la placa arduino para recibir los datos calculados por los programas que corren en las placas.

Instalación y configuración

1. Para realizar el experimento se utilizó el siguiente equipo:

   a) Una computadora con las caracteristicas anteriormente descritas.
   - Arduino IDE instalado.
     - Descargar y extraer el archivo de la pagina de arduino: https://www.arduino.cc/en/software
       - Crear una carpeta temporal para guardar ahi el archivo descargado y posicionarse en ella desde una terminal.
       - Ejecutar el siguiente comando para descomprimir el archivo descargado: tar -xvf ./arduino-1.8.15-linux64.tar.xz
     - Cambiar a la carpeta descomprimida y ejecutar el instalador con los siguientes comandos:
       - cd arduino-1.8.15/
       - sudo ./install.sh
     - Buscar el icono de Arduino en el escritorio para abrir la aplicación.
     - Instalar librería LoRa.h
       - Seleccionar Sketch -> Incluir biblioteca -> Administrar bibliotecas...
       - Escribir LoRa en el cuadro de búsqueda.
       - Hacer clic en la fila para seleccionar la biblioteca.
       - Hacer clic en el botón Instalar para instalar la biblioteca.
   - Agregar el usuario del sistema al grupo dialout para evitar posibles problemas al utilizar el IDE de Arduino.
     - Puede ocurrir que al subir un sketch aparezca el siguiente error: "Error al abrir el puerto serie..."
     - De ser así, se deben configurar los permisos del puerto serie con el siguiente comando:
       - sudo usermod -a -G dialout <user> , donde <user> es el usuario al que se quiere agregar al grupo.

   b) Dos placas Arduino UNO, *sender* y *receiver*.
   - Cada placa se conecta a un módulo Ra-01 LoRa, siguiendo la siguiente configuración

LoRa SX1278 Module	Arduino UNO Board
3.3V	3.3V
Gnd	Gnd
En/Nss	D10
G0/DIO0	D2
SCK	D13
MISO	D12
MOSI	D11
RST	D9

Figura 4.1.8: Configuración de conexiones.

   - Instalar los programas sender.ino y receiver.ino, deben de instalarse uno en cada una de las placas utilizando el IDE de Arduino.
     - Conectar por usb la placa a la computadora con el IDE instalado.
     - Abrir el programa sender.ino con el IDE.

- Seleccionar el puerto serial por el cual esta conectada la placa, seleccionar: Herramientas/Puerto.
- Click en el boton Subir para cargar el programa en la tarjeta.

2. Para ejecutar los programas, primero es necesario compilar el archivo serial_port.c , utilizando el comando gcc en una terminal:

   a) gcc serial_port.c -o serial_port.c -lpthread

3. Una vez compilado el programa, para ejecutarlo se utiliza el comando "./", no se deben de olvidar los parámetros

   a) ./serial_port USB_port size_message number_packages
      - Donde USB_port es el nombre del puerto serial por el cual el Arduino se conecta, ejemplo:
        - ./serial_port /dev/ttyACM0 8 12

4. En la computadora con el programa serial_port.c instalado se debe conectar la placa arduino con el programa sender.ino.

5. El segundo arduino, con el programa receiver.ino, se conecta a una fuente de energía para que se pueda ejecutar el programa.

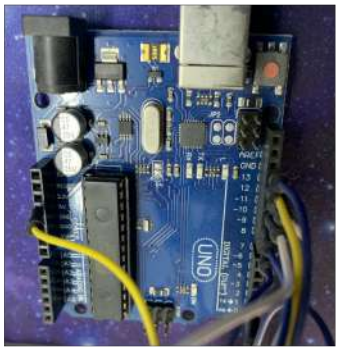

Figura 4.1.9: Arduino UNO.

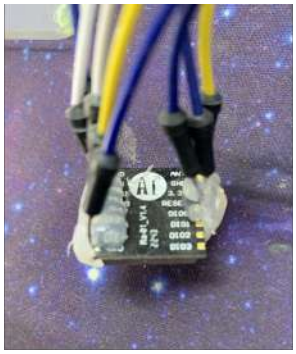

Figura 4.1.10: Dispositivo LoRa Ra-01.

Figura 4.1.11: Dispositivo LoRa Ra-01.

Diseño de experimento

El programa receiver (.ino) se define con la siguiente estructura:

1. Inicializa LoRaReceiverCallback
   LoRa.begin(433E6);
2. Registra el nombre de la función que se realizara al recibir datos
   LoRa.onReceive(onReceive);
3. Configura la RF en modo de recibir.
   LoRa.receive();
4. Si llegan datos, se activa onReceive, quien lee mensaje y lo envía de vuelta al sender.
   sendMessage(message);

El programa sender (.ino) se define con la siguiente estructura:

1. Inicializa Lora.
   LoRa.begin(433E6)
2. Registra el nombre de la función que se realizara al recibir datos
   LoRa.onReceive(onReceive);
3. Recibe por puerto serial los parámetros para ejecutar el experimento.
   number_messages = data[0];
   message_size = data[1];
   message = "";
4. Se crea y envía el mensaje al receiver.
   LoRa.beginPacket();
   LoRa.print(message);
5. Se guarda la hora de salida de cada paquete.
6. Se calcula el tiempo de viaje del paquete cuando el mensaje llega de regreso.
7. Los datos de tiempo de viaje y paquetes recibidos son enviados por puerto serial.

El programa serial_port (.c) se define con la siguiente estructura:

1. Verifica argumentos.
   Debe de ejecutarse así: ./serial_port USB_port_number size_message number_packages
2. Los datos size_message y number_packages se envían por puerto serial para iniciar el experimento.
3. Se queda en espera para recibir los resultados del experimento.
4. Calcula Jitter y Tasa de paquetes perdidos con los datos de los paquetes recibidos.
   jitter = fabs(get_average());
   rate = abs((((float) count_packages_received/(float)number_packages)* 100) - 100);
5. Crea archivos de texto para graficar con gnu.
   create_gnu_files(rate,jitter);

Una vez que ambos arduinos son conectados a la fuente de poder o a la computadora, comienza la ejecución de los programas, para inicializar el experimento se ejecuta el programa serial_port incluyendo los parámetros de tamaño y cantidad de mensajes. Al terminar, el programa mostrara en consola los datos obtenidos del experimento y habrá realizado los archivos de texto para graficar con gnu.

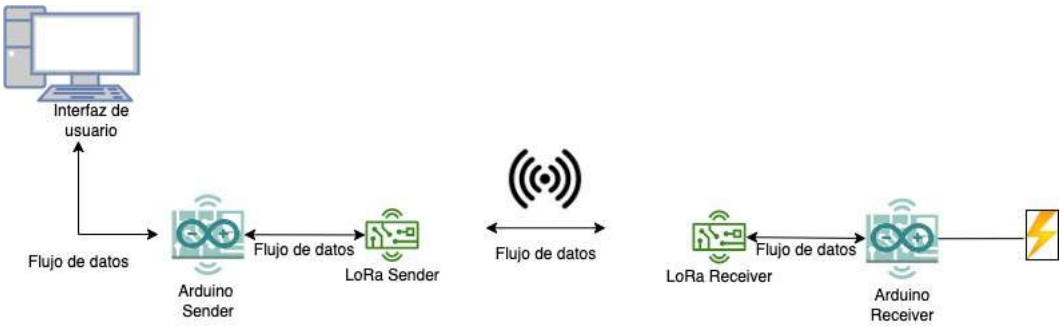

Figura 4.1.12: Descripción gráfica de las conexiones en el experimento LoRa.

4.2. Resultados.

Se realizaron pruebas con las herramientas desarrolladas para medir la calidad de los distintos protocolos de comunicación de redes inalámbricas, con base en el Jitter (Segundos) y la Tasa de paquetes perdidos (Porcentaje) calculados en función del tamaño del buffer (2, 4, 6, 8, 10, 12, 14 y 16 bytes). Las pruebas se realizaron con y sin interferencia, a distintas distancias entre los dispositivos: 2.5m, 5m, 10m, 15m, 20m, 25m y 30m.

4.2.1. Wi-Fi

4.2.1.1. Resultados cualitativos

Para el desarrollo de esta aplicación se utilizaron sockets, en la programación en C, un socket es un "enchufe" con el cual se crea una conexión con otro ordenador en la que se permite intercambiar datos dentro de una red. Se creó una aplicación cliente-servidor, la cual, con la ayuda de la librería <sys/socket.h>, que es parte de la biblioteca GNU de C, se utilizaron sockets de tipo UDP (Datagramas) para crear conexiones entre dos nodos.

- Servidor: es el programa que permanece pasivo en la espera de que algún cliente solicite conexión. Puede o no devolver datos.
- Cliente: es el programa que solicita la conexión para enviar o solicitar datos al servidor.

Existen dos tipos de sockets, los orientados a conexión y los no orientados a conexión.

- Orientados a conexión (TCP): Dos programas deben conectarse entre ellos a través de un socket y hasta que no esté establecida correctamente la conexión, ninguno puede transmitir datos.

- Sin conexión (UDP): No es necesario que los programas tengan una conexión establecida. Cualquiera puede transmitir datos en cualquier momento, independientemente de que el otro programa esté "escuchando" o no.

En este caso se utilizaron sockets de tipo UDP para analizar la cantidad de paquetes perdidos dentro de la red durante el envío de datos.

Para poder utilizar sockets se incluyeron a los programas las librerias:

- #include <sys/types.h>
- #include <sys/socket.h>
- #include <unistd.h>
- #include <arpa/inet.h>
- #include <netinet/in.h>

Un socket se crea con la siguiente función:

- int socket(int domain, int type, int protocol);

Esta función devuelve un fichero descriptor del socket, la cual se utilzara más adelante para llamadas al sistema. Si nos devuelve -1, se ha producido un error, lo que resulta útil para la de verificación de errores.

Analizando los argumentos:

- domain. Se establece como AF_INET para usar los protocolos ARPA de Internet.
- type. Aquí se especifica la clase de socket que se va a usar, en este caso de Datagramas, se utiliza el valor SOCK_DGRAM.
- protocol. Aquí se establece el protocolo a 0.

Para enviar datos a traves de la red, se utilizó la función:

- int sendto(int fd, const void *msg, int len, int flags);

El propósito de esta función es enviar datos usando sockets no conectados de datagramas, devuelve -1 en caso de error, o el número de los bytes enviados en caso de éxito.

Analizando los argumentos de esta llamada:

- fd. Es el fichero descriptor del socket, con el cual se desea enviar datos.
- msg. Es un puntero al dato que se va a enviar.
- len. Longitud del dato que se va a enviar, en bytes.
- flags. Se estableció a 0.

Para que el programa servidor pudiera recibir los datos del cliente, se utilizó la funcion:

- int bind(int fd, struct sockaddr *my_addr,int addrlen);

Su función esencial es asociar un socket con un puerto de la máquina.

Un punto importante sobre los puertos y esta llamada es que todos los puertos menores que 1024 se encuentran reservados. Por lo tanto, solo se podrá establecer un puerto, siempre y cuando esté entre 1024 y 65535 y que no estén siendo usados por otros programas de la máquina. Además, ambos programas (cliente y servidor) deben compartir el número de puerto.

En esta aplicación tambien fue necesario recibir los datos enviados, para lo que se utilizó la siguiente función:

- int recvfrom(int fd,void *buf, int len, unsigned int flags struct sockaddr *from, int *fromlen);

El objetivo de esta función es recibir datos usando sockets de Datagramas y devuelve el número de bytes recibidos, o -1 en caso de error.

Al finalizar el proceso se debe cerrar la conexión que se creo, para eso se utilizó la función:

- Close(fd); donce fd. Es el fichero descriptor del socket que se va a cerrar.

La función es usada para cerrar la conexión del descriptor de socket. Si se llama a close() no se podrá escribir o leer usando ese socket, y si alguien trata de hacerlo recibirá un mensaje de error.

Es una tecnología flexible y muy fácil de utilizar debido a la gran cantidad de documentación que existe al respecto, no fue necesario hacer instalaciones especiales, debido a que la librería ya se encuentra entre las esenciales del lenguaje C, fue diseñada en un inicio para UNIX, pero en la actualidad existen versiones para una gran variedad de plataformas y sistemas operativos.

4.2.1.2. Resultados cuantitativos

Los resultados arrojados por las pruebas, se vaciaron en gráficas para poder observar el comportamiento de los datos Jitter y Tasa de paquetes perdidos.

Graficas Jitter En general, el Jitter se mantuvo en 0 (segundos) durante todo el proceso de pruebas, a excepción de un par de ocasiones en las que subió a 0.5 y 1, se puede deducir que a pesar de la distancia, Wi-Fi mantiene su velocidad estable y no se ve afectada por las interferencias.

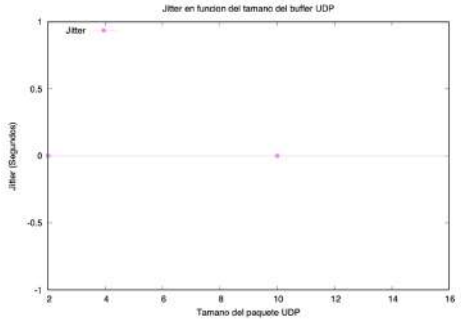

Figura 4.2.1: Pruebas de 2.5 metros

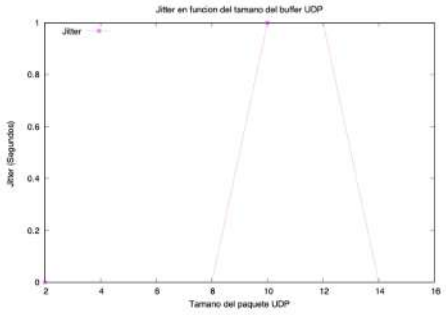

Figura 4.2.2: Pruebas de 2.5 metros

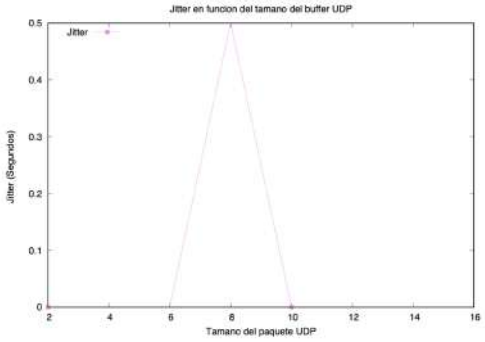

Figura 4.2.3: Pruebas de 10 metros

Graficas Tasa de paquetes perdidos La tasa de paquetes perdidos varió entre el 20 % y 30 % durante las pruebas a pesar de la distancia y la interferencia. A la hora de recibir datos, Wi-Fi mostró que puede no siempre ser el mas confiable.

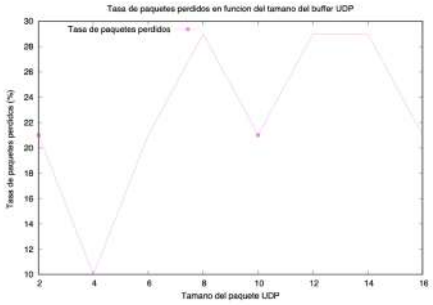

Figura 4.2.4: Pruebas de 2.5 metros

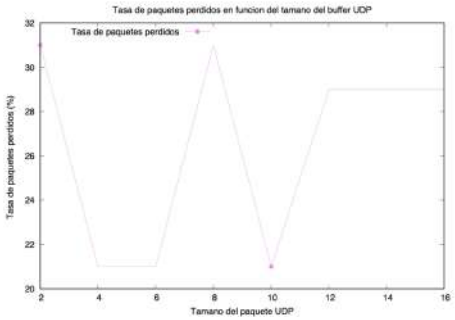

Figura 4.2.5: Pruebas de 5 metros

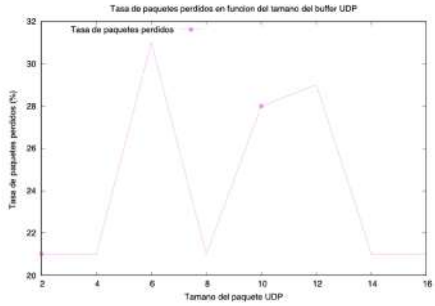

Figura 4.2.6: Pruebas de 15 metros

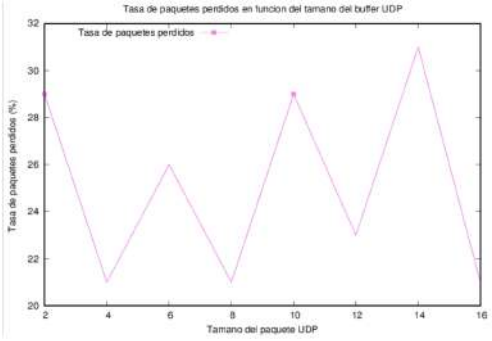

Figura 4.2.7: Pruebas de 30 metros

4.2.2. Bluetooth

4.2.2.1. Resultados cualitativos

Para evaluar el protocolo de comunicación Bluetooth, se realizó una aplicación cliente-servidor que utiliza la librería BlueZ , una potente pila de comunicaciones Bluetooth con amplias APIs para el lenguaje C, que permite al usuario aprovechar los recursos Bluetooth locales. BlueZ proporciona soporte para las capas y protocolos centrales de Bluetooth.

Para establecer y usar conexiones, Blutooth utiliza RFCOMM sockets, tecnología que se reduce a las mismas técnicas de programación de sockets mencionada anteriormente para la programación de TCP/IP. La única diferencia es que las estructuras de direccionamiento de los sockets son diferentes, y se utilizan diferentes funciones para el ordenamiento de bytes.

Para utilizar las funciones de BlueZ, se agregaron las siguientes librerías al programa:

- #include <bluetooth/bluetooth.h>
- #include <bluetooth/rfcomm.h>
- #include <bluetooth/hci.h>
- #include <bluetooth/hci_lib.h>
- #include <bluetooth/sdp.h>
- #include <bluetooth/sdp_lib.h>
- #include <bluetooth/sco.h>
- #include <json-c/json.h>
- #include <sys/socket.h>

Al igual que con la programación de Internet, primero se asigna un socket con la llamada al sistema. En lugar de AF_INET, se usa AF_BLUETOOTH y en lugar de IPPROTO_TCP, se usa BTPROTO_RFCOMM. Debido a que RFCOMM proporciona la misma semántica de entrega que TCP, se puede usar SOCK_STREAM para el tipo de socket.

- ssocket = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);

Para establecer una conexión con otro dispositivo Bluetooth, se crea una estructura de direccionamiento struct sockaddr_rc. Al igual que la estructura sockaddr_in que se usa en TCP/IP, la estructura de direccionamiento especifica los detalles de una conexión saliente o un conector de escucha.

- struct sockaddr_rc { sa_family_t rc_family; bdaddr_t rc_bdaddr; uint8_t rc_channel; };
- struct sockaddr_rc address = { 0 };

Analizando los argumentos:

- El campo rc_family especifica la familia de direccionamiento del socket y siempre será AF_BLUETOOTH.
  - address.rc_family = AF_BLUETOOTH;
- Para una conexión saliente, rc_bdaddr y rc_channel especifican la dirección de Bluetooth y el número de puerto para conectarse, respectivamente.
  - address.rc_channel = (uint8_t) 1;
- Para un conector de escucha, rc_bdaddr especifica el adaptador Bluetooth local que se va a utilizar y normalmente se establece en BDADDR_ANY para indicar que cualquier adaptador Bluetooth local es aceptable.
- Para los sockets de escucha, rc_channel especifica el número de puerto para escuchar.

A los adaptadores Bluetooth se les asignan números de identificación, y un programa debe especificar qué adaptador usar al asignar los recursos del sistema. Generalmente, solo hay un adaptador o no importa mucho cuál se use, por lo que pasar NULL a hci_get_route regresará el número de recurso del primer adaptador Bluetooth disponible.

- int hci_get_route(bdaddr_t *bdaddr);

- int hci_open_dev(int dev_id);

Después de elegir el adaptador Bluetooth local para usar y asignar los recursos del sistema, el programa está listo para buscar dispositivos Bluetooth cercanos. En esta aplicación, el programa cliente realiza un descubrimiento utilizando la función hci_inquiry y devuelve una lista de dispositivos detectados más alguna información básica sobre ellos en la variable ii. En caso de error, devuelve -1.

- num_rsp = hci_inquiry(dev_id, len, max_rsp, NULL, &ii, flags);

Una vez que se genera la lista de dispositivos cercanos y sus direcciones, el programa determinara los nombres fáciles de usar asociados con esas direcciones y en este caso, buscará que el servidor se encuentre en la lista.

- Para esto se utilizó la función hci_read_remote_name.
- Para establecer la conexión con el servidor, se utiliza la función connect(ssocket, (struct sockaddr *)&address, sizeof(address));

Después de que el servidor elige el adaptador Bluetooth local, el programa se queda en espera y "escuchando" si llega alguna conexión de un cliente, para esto se utiliza la función listen();, cuando un cliente hace la petición de conexión, el servidor utiliza la función accept();. Una vez que se ha establecido una conexión, las llamadas para leer, enviar y recibir se pueden usar para la transferencia de datos.

- bytes_read = read(client, buf, sizeof(buf));
- write(client, buf, 72);, se puede enviar un máximo de 72 caracteres.
- ba2str(&rem_addr.rc_bdaddr, buf);

BlueZ es flexible y eficiente. Las bibliotecas y utilidades del kernel de BlueZ funcionan perfectamente en muchas arquitecturas compatibles con Linux. El soporte para BlueZ se puede encontrar en muchas distribuciones de Linux y, en general, es compatible con cualquier sistema Linux del mercado.

4.2.2.2. Resultados cuantitativos

Los resultados arrojados por las pruebas, se vaciaron en gráficas para poder observar el comportamiento de los datos Jitter y Tasa de paquetes perdidos.

Graficas Jitter Las pruebas dieron el mismo resultado desde los 2.5m hasta los 10m en todas las medidas de buffer, con el Jitter en 0 segundos. El resultado arrojó un pequeño cambio a los 15m sin interferencia, en el Jitter en el tamaño de buffer 2, pero continuó en 0 el resto de las pruebas. En las pruebas de 20m, el jitter mostró un resultado más variado, pero aun así, sigue siendo menos de 1 segundo.

Decir que tarda menos de 1 segundo puede hacer pensar que el sistema de envío de datos es muy rápido, pero de cierto modo también puede ser poco confiable para la programación de aplicaciones de tiempo real estricto, debido a las variaciones de microsegundos que suceden de manera aleatoria.

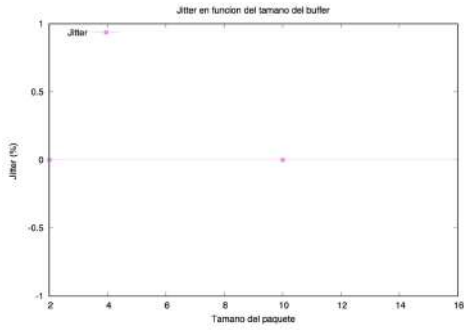

Figura 4.2.8: Pruebas de jitter 2.5 a 10 metros

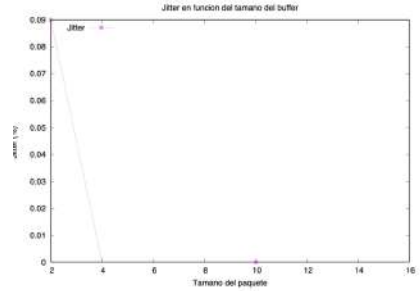

Figura 4.2.9: Pruebas de jitter a 15 metros

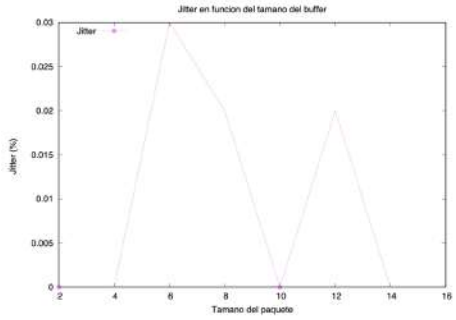

Figura 4.2.10: Pruebas de jitter a 20 metros

Graficas Tasa de paquetes perdidos La tasa de paquetes perdidos se mantuvo en 0 % durante todas las pruebas, a excepción de un par de ocasiones que subió al 90 % y al 100 % en las pruebas de 30 metros con interferencia. Por lo que se puede decir que Bluetooth es bastante confiable respecto al envío de información completa, siempre y cuando haya poca interferencia y las aplicaciones sean de corta distancia.

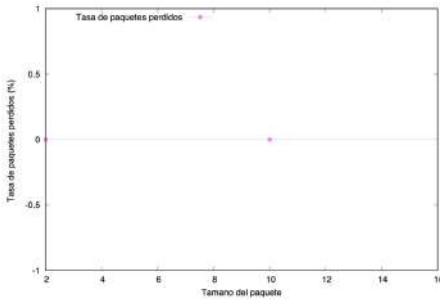

Figura 4.2.11: Pruebas de 2.5 a 25 metros con y sin interferencia

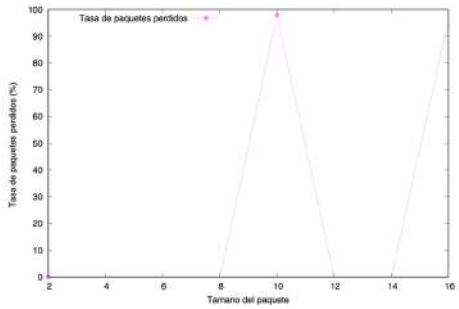

Figura 4.2.12: Pruebas de 30 metros con interferencia

4.2.3. ZigBee

4.2.3.1. Resultados cualitativos

Para desarrollar aplicación en lenguaje C que mide la calidad de un enlace de red ZigBee, se utilizaron dispositivos ZigBee (Digi XBee®), los cuales, a través de un puerto serial/USB y con la ayuda de la librería de código abierto <xbee.h>, especialmente creada para la intercomunicación con dispositivos xbee, se comunican con los programas cliente y servidor de la aplicación. La librería <xbee.h>, resulta muy fácil de utilizar debido a la gran cantidad de ejemplos con los que cuenta en el repositorio de GitHub en el que se encuentra publicada.

La tecnología sigue la misma lógica de programación que hermos venido viendo hasta el momento en las demas aplicaciones, de igual manera, lo que cambia son las funciones de direccionamiento y configuración de los dispositivos.

- struct xbee *xbee = configure_xbee(xbee, ret, USB_port_number);
  - donde USB_port_number es el puerto por el cual el dispositivo está conectado al CPU.
- struct xbee_con *con = connection_xbee(xbee, con, ret, xbee_address);
  - donde xbee_address es la dirección MAC del dispositivo, se puede consultar con la aplicación XCTU o en la placa.

Una vez establecida la conexión, se puede hacer uso de las funciones para enviar y recibir datos:

- receive_data(server_zigBee->xbee, server_zigBee->con, server_zigBee->ret);
- ret = xbee_conTx(con, &retVal, message_json));

Además, para que funcionen adecuadamente, los dispositivos se deben configurar según la función a ejecutar, para esto se utiliza XCTU, una aplicación multiplataforma gratuita diseñada para que los desarrolladores puedan interactuar con los módulos RF de Digi a través de una interfaz gráfica. Incluye herramientas que facilitan la instalación, configuración y prueba de los módulos XBee® RF. Utilizar la aplicación es bastante sencillo gracias a que la pagina oficial ofrece soporte y documentación para windows y Linux en general.

Un dato muy importante que se debe tomar en cuenta al querer hacer una aplicación donde la información fluya de cliente a servidor y de servidor a cliente indistintamente, es que un dispositivo con el modo API desactivado no será capaz de recibir los datos correctamente, es por eso que en esta aplicación, el dispositivo cliente tambien se configura como API.

4.2.3.2. Resultados cuantitativos

Los resultados arrojados por las pruebas, se vaciaron en gráficas para poder observar el comportamiento de los datos Jitter y Tasa de paquetes perdidos.

Graficas Jitter Los resultados del Jitter fueron bastante estables durante todas las pruebas, desde los 2.5 hasta los 30 metros con y sin interferencia, con un promedio entre 0 y 0.12 segundos de velocidad. Se esperaría que ZigBee funcione en una aplicación del internet de las cosas, más por su estabilidad que por su velocidad.

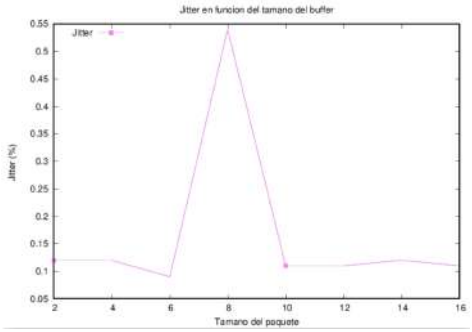

Figura 4.2.13: Pruebas de jitter 2.5 con y sin interferencia

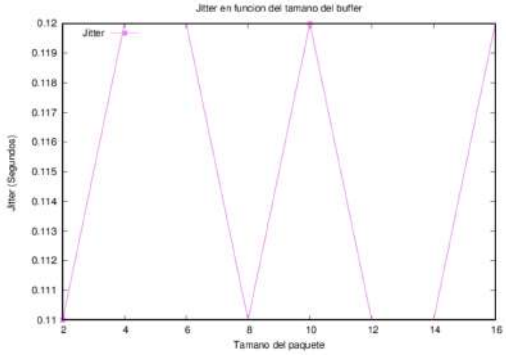

Figura 4.2.14: Pruebas de jitter a 15 metros con interferencia

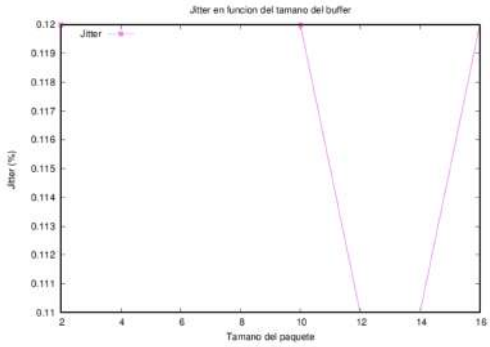

Figura 4.2.15: Pruebas de jitter a 30 metros sin interferencia

Graficas Tasa de paquetes perdidos La tasa de paquetes perdidos se mantuvo en 0 % durante todas las pruebas, con y sin interferencia. Por lo que se puede decir que ZigBee es bastante confiable respecto al envío de información completa, independientemente de distancias e interferencias.

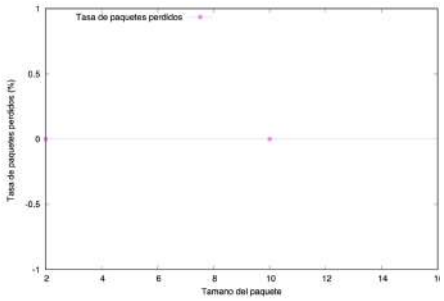

Figura 4.2.16: Pruebas de 2.5 a 30 metros con y sin interferencia

4.2.4. LoRa

4.2.4.1. Resultados cualitativos

La aplicación para evaluar al protocolo LoRa se compone por tres programas, dos de ellos: Sender y Receiver, fueron desarrollados para trabajar en placas Arduino UNO en conjunto con un módulo Ra-01 LoRa desarrollado por la compañía Ai-Thinker.

Para poder programar el modulo en arduino se utilizó la librería:

- #include <LoRa.h>

Gracias a esta librería y sus funciones, configurar los dispositivos es muy sencillo.

- LoRa.receive(); para que el dispositivo se encuentre abierto a recibir datos.

- message += (char)LoRa.read(); para acumular los datos que se vayan recibiendo.
- LoRa.write(outgoing.length()); para escribir en el puerto los datos que se desean enviar.
- LoRa.endPacket(); para enviar el paquete en cola.

El tercer programa se desarrolló en el lenguaje C, el cual se comunica con la placa arduino para recibir los datos calculados por los programas que corren en las placas, todo esto a través de un puerto serial con la ayuda de la librería <termios.h>, el programa se construyó utilizando las siguientes funciones:

- serial_port = open(USB_port_number, O_RDWR);
- write(serial_port, msg, sizeof(msg));
- int num_bytes = read(serial_port, &read_buf, 4);
- close(serial_port);
- tcgetattr(serial_port, &tty); lee la configuración existente.

4.2.4.2. Resultados cuantitativos

Durante las pruebas de desarrollo, los dispositivos se comportaron como se esperaba, recibiendo y enviando datos de uno a otro, Jitter y Tasa de paquetes perdidos resultaron en 0 en las pruebas de 1/2 a 1 metro. Los resultados comenzaron a cambiar cuando se decidió a aumentar la distancia entre los dispositivos; la comunicación se perdió a unos escasos centímetros más de distancia, se tendrán que repetir las pruebas con nuevos dispositivos para asegurar o descartar que los utilizados en el proyecto no funciaban correctamente.

Capítulo 5

Conclusiones y trabajo futuro

Queda claro que, para realizar aplicaciones del Internet de las Cosas, algo muy importante por tener en consideración es la comunicación, así como la potencia requerida para transmitir y recibir datos, por lo tanto, la elección de la tecnología de red correcta es fundamental. Elementos importantes por considerar son la distancia entre el emisor y el receptor, la naturaleza de los obstáculos, el ruido ambiental y las regulaciones gubernamentales definidas para cada tecnología. Se debe elegir un protocolo de red inalámbrica determinado que se ajuste a las necesidades del proyecto. Este estudio demuestra que existen muchas tecnologías inalámbricas para aplicaciones IoT, y que cada una tiene sus propias especificaciones, perjuicios y beneficios. Sin embargo, es importante recalcar que las tecnologías más modernas aún se encuentran en etapa de mejora y que no siempre será lo más sencillo trabajar con ellas; hacen falta más estudios dedicados a la configuración y su aplicación en ambientes de experimentación.

En un trabajo futuro, esta investigación se extenderá para revisar las tecnologías de comunicación que por falta de tiempo y poca disponibilidad de dispositivos no se revisaron más a fondo. Sería un gran aporte a la comunidad un documento que especifique cómo configurar y usar las tecnologías existentes para que cualquier investigador o desarrollador lo tome como base y no tenga que empezar su proyecto desde cero.

Capítulo 6

Apéndice

6.1. Estructura base Cliente.c

```
int main(int argc, char *argv[]) {
    // Verificar argumentos recibidos
    if (argc != 4) {

        fprintf(stderr,"Debería de usarse así: host_name size_message number_packages\n");
        exit(1);
        }
    // Obtener información del servidor - cambia con cada protocolo
        ...
    // Obtenener información de los argumentos recibidos
    size_message = atoi(argv[2]);
    number_packages = atoi(argv[3]);
    // Crear socket - cambia con cada protocolo
        ...


    // Preparar datos para la conexion con el servidor - cambia con cada protocolo
        ...
    // Inicializar el hilo que se encarga de escuchar al servidor
        pthread_create(&thread_listen_server, NULL, listen_server, NULL);
    // Crear mensaje que se enviará en cada paquete
    memset(&message_, 0, sizeof(message_));
    for (int i = 0; i < size_message - 8; i++) {
        strcat(message_, "x");
    }


    // Enviar la cantidad de paquetes especificados en argumentos
    send_data(int number_packages, message_)
    // Tiempo de espera "Timeout"
    while ( seconds_elapsed < timeout * 1000) {
        end_t = clock();
        seconds_elapsed = (double)(end_t - start_time) / CLOCKS_PER_SEC * 1000;
        printf( "[WAITING FOR%.0f Of%.0f MILISECONDS ]",
            seconds_elapsed, timeout * 1000);
        printf( "\r");
    }
    // Detener escuchar servidor
    pthread_mutex_lock(&mutex_packages_received);
```

```
    pthread_cancel(thread_listen_server);
    //Cerrar socket
    close( socket );
    //Calcular jitter  y rate
    jitter = get_average();
    rate = abs((((float) count_packages_received
        /(float)number_packages)* 100) - 100);
}
void send_data(int s){
    //Se obtiene la hora en la que sale el paquete
    time (&curtime);
    //La forma en que se envía el paquete al servidor cambia segun el protocolo
        ...
}
void * listen_server(){
    //Esta función es un proceso que se utiliza con hilos para mantener
    //abierta la comunicación con un servidor
    //Cambia segun el protocolo
    ...
    //Aquí se reciben los datos que el servidor envía de regreso
    //Cada paquete se marca con la hora en que llego y se calcula su tiempo de viaje
    time_travel = (server_in_int - client_out_int) + (client_in_int - server_out_int);
}
float get_average(){
    float avrg = 0;
    for (int i = 0; i < i_package - 1; i++) {
        //times es un arreglo donde se guardan los tiempos de viaje
        int difference = (times)[i + 1] - (times)[i];
        avrg += (float) difference;
    }
    avrg = avrg/(float)count_packages_received;
    return avrg;
}
int create_array(int **array, int c){
    //Se utilizan arreglos para guardar la información de los tiempos
    (*array) = (int *) malloc(c * sizeof(int));
    if ( !(*array) ) {
        return 0;
    }
    return 1;
}
int print_to_file(char * filename, int data1, float data2){
    //Se utilizan archivos para almacenar los resultados de las pruebas
    FILE * ou_file;
    ou_file = fopen(filename, "a+");
    if ( !ou_file ) {
        return 0;
    }
    fprintf(ou_file,"%d\t%.2f\n", data1, data2);
    return 1;
}
int create_gnu_file(char * filename, char * file_data_name){
```

```
    //Se crean archivos gnu para graficar los resultados de las pruebas
    FILE * ou_file;
    ou_file = fopen(filename, "w");
    if ( !ou_file ) {
        return 0;
    }
        ...
    fprintf(ou_file,"set xlabel \"Tamaño del paquete UDP\" \n");
    fprintf(ou_file,"set ylabel \"Tasa de paquetes perdidos (%c) \" \n", '%');
        ...
}
```

6.2. Estructura base Servidor.c

```
int main() {

    //Crea socket segun el protocolo
    ...
    printf("Socket created.\n");

    //Prepara al servidor para recibir conexiones
    printf("Binded\n");
      // Queda en espera de que algún cliente pida conectarse
    printf("Listening...\n");
      //Si llegan datos de algun cliente se llama
    callback_function(buf, client);

}
void callback_function(char buf[MAXLINE], int client){

    //Se obtiene la información del mensaje recibido según el protocolo
        ...
    //Se obtiene el tiempo en el que llego el mensaje
    time_t server_in = time(NULL);
    char current_time[MAXLINE];
    struct tm *lt = localtime(&server_in);
    //Se obtiene el tiempo en el que el mensaje sale de vuelta
    time_t server_out = time(NULL);
    // Enviar mensaje de vuelta según las reglas del protocolo

}
```

Bibliografía

[1] Omojokun G Aju. A survey of zigbee wireless sensor network technology: Topology, applications and challenges. *International Journal of Computer Applications*, 2015.

[2] Omojokun G. Aju. A survey of zigbee wireless sensor network technology: Topology, applications and challenges. *International Journal of Computer Applications*, 11 2015.

[3] Shadi Al-Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi. Internet of things (iot) communication protocols : Review. *8th International conference on information technology (ICIT)*, 5 2017.

[4] M. Aldea Rivas. Planificación de tareas en sistemas operativos de tiempo real estricto para aplicaciones empotradas. 11 2002.

[5] LoRa Alliance. About lora alliance.

[6] Wi-Fi Alliance. Who we are. *https://www.wi-fi.org/*, 04 2021.

[7] ZigBee Alliance. About us. *Zigbee Alliance.*, 04 2021.

[8] Manuel Rodríguez Alija Andrea Recalde Baraibar. Redes inalámbricas. *Universidad Pública de Navarra*, 04 2021.

[9] J. J. Anguís Horno. Redes de área local inalámbricas: Diseño de la wlan de wheelers lane technology college. *UNIVERSIDAD DE SEVILLA*, 03 2008.

[10] arduino.cc. Wifi library. 04 2021.

[11] Hel Tec Automation. Esp32$_lorawan.github$, 122020.

[12] Bluetooth. About bluetooth.

[13] Andrea Murillo Carrillo. Evaluación del desempeño en la transmisión de señales biomédicas en un ambiente inalámbrico en redes 6lowpan. *Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California*, 2017.

[14] Python Contributors. python-wifi 0.6.1. 11 2015.

[15] Wikipedia contributors. Wirelesshart. *Wikipedia*.

[16] Android Developers. Bluetooth overview. 04 2021.

[17] Ecured. Bluetooth. 04 2021.

[18] EcuRed. Zigbee. 04 2021.

[19] Khalid Fakieh. A survey on 6lowpan its future research challenges. 01 2014.

[20] Khalid Fakieh. A survey on 6lowpan its future research challenges. 01 2014.

[21] Erina Ferro and Francesco Potorti. Buetooth and wi-fi wireless protocols: A survey and a comparison. *IEEE Wireless Communications*, 2005.

[22] Thanos Fisherman. Wifiutils. 02 2021.

[23] A. García. El wifi cambia de nombre: 802.11ac ahora es wi-fi 5 y 802.11ax es wi-fi 6. *ADSLZone*, 05 2020.

[24] Cristina Cueto García. 20 años de conexiones inalámbricas. *Computer World*, 06 2019.

[25] R. García. Qué es el wifi y cómo funciona para conectar todo a internet. *ADSLZone*, 03 2021.

[26] Andre Gloria, Francisco Cercas, and Nuno Souto. Comparison of communication protocols for low cost internet of things devices. *South Eastern European Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*, 2017.

[27] FieldComm Group. Hart. *FieldComm*.

[28] DiGi International. Digi xbee python library 1.4.0. 2017.

[29] DiGi International. xbee-csharp. *github*, 07 2019.

[30] DiGi International. Zigbeenet. *github*, 2019.

[31] DiGi International. xbee-csharp. *github*, 08 2020.

[32] Joao C. Netto Ivan Muller, Carlos E. Pereira. Development of a wirelesshart compatible field device. *Academia*, 2010.

[33] Juha, Konstantin Mikhaylov, Marko Pettissalo, Janne Janhunen, and Jari Iinatti. Performance of a low-power wide-area network based on lora technology: Doppler robustness, scalability, and coverage. 2017.

[34] Saliyah Kahar, Riza Sulaiman, Anton Satria Prabuwono, Nahdatul Akma Ahmad, Mohammad Ashri, and Abu Hassan. A review of wireless technology usage for mobile robot controller. *IPCSIT vol. 34 (2012)*, 04 2012.

[35] Anna N Kim, Fredrik Hekland, Stig Petersen, and Paula Doyle. When hart goes wireless: Understanding and implementing the wirelesshart standard. 2008.

[36] Tomas Lennvall, Stefan Svensson, and Fredrik Hekland. A comparison of wirelesshart and zigbee for industrial applications. 2008.

[37] libraries.io. Bluetooth manager for windows 10. *Robmiles.com*, 04 2021.

[38] libraries.io. node-bluetooth release 1.2.6. 04 2021.

[39] J. A. Lorenzo. Conoce todo sobre wi-fi halow, el estándar wi-fi para iot. *Redes Zone*, 2020.

[40] William MacDougall. Industrie 4.0: Smart manufacturing for the future. *macdougall2014industrie*, 2014.

[41] Jorge Marín Guerrero. Sistema de guiado para invidentes (s.g.i) basado en bluetooth, j2me y dispositivos telefónicos móviles. *e-REding*, 04 2021.

[42] Claudia Martínez. Evaluación de protocolos de redes de computadoras para aplicaciones de la industria 4.0. *Instituto Tecnológico Nacional de México*, 2021.

[43] Leonardo Pérez Martïnez. Consideraciones de diseño e implementación de un sistema de monitoreo inalámbrico en campo con tecnología wireleshart. *Instituto Politécnico Nacional*, 09 2016.

[44] Michael, Markus Lorenz, Philipp Gerbert, Manuela Waldner, Jan Justus, Pascal Engel, and Michael Harnisch. Industry 4.0: The future of productivity and growth in manufacturing industries. *Boston Consulting Group*, 2015.

[45] Sandeep Mistry. Lora. *Arduino.cc*.

[46] José Luis Camargo Olivares. Modelo de cobertura para redes inalámbricas de interiores. *UNIVERSIDAD DE SEVILLA*, 05 2019.

[47] Manuel Menchaca Paz. Integración de zigbee/6lowpan en una red de sensores inalabrica. *Universidad de Cantabria*, 07 2012.

[48] pcwdld.com. Network jitter – what is it and how to monitor it with software/tools.

[49] The Android Open Source Project. Wifimanager.java. 2008.

[50] Linux Bluetooth protocol stack contributors. Bluez. 10 2021.

[51] pypi.org. This fork - pylora.

[52] Alberto Escudero Sebastian Buettrich. Topología e infraestructur estructura básica de redes inalámbricas. *Universidad Autónoma de Chiapas*, 10 2017.

[53] Cat Sensors. Tecnología lora y lorawan.

[54] Internet Society. Sobre el ietf. *Internet Society*.

[55] T. Watteyne. Smartmesh sdk - confluence.