UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO

DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

ANÁLISIS DEL DESEMPEÑO DE TECNOLOGÍAS DE TRANSMISIÓN DE LARGO ALCANCE EN EL DESARROLLO DE SISTEMAS IOT-INDUSTRIA 4.0.

TRABAJO DE TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN MECATRÓNICA

PRESENTA

ING. LUIS ISAAC DOMÍNGUEZ GÁMEZ

DIRECTOR

DR. JAVIER VÁZQUEZ CASTILLO

CO-DIRECTOR

DR. ROBERTO CARRASCO ÁLVAREZ

ASESORES

DR. JAIME SILVERIO ORTEGÓN AGUILAR

DRA. EDITH OSORIO DE LA ROSA

MM. JOSÉ RAÚL GARCÍA SEGURA

CHETUMAL QUINTANA ROO, MÉXICO, DICIEMBRE DEL 2022

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO

DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

TRABAJO DE TESIS BAJO LA SUPERVISIÓN DEL COMITÉ DEL PROGRAMA DE MAESTRÍA Y APROBADA COMO REQUISITO PARA OBTENER EL GRADO DE:

MAESTRO EN MECATRÓNICA

COMITÉ SUPERVISOR

DIRECTOR: _____

DR. JAVIER VAZQUEZ CASTILLO

CO-DIRECTOR: _____

DR. ROBERTO CARRASCO ÁLVAREZ

ASESOR: _____

DR. JAIME SILVERIO ORTEGÓN AGUILAR

ASESOR: _____

DRA. EDITH OSORIO DE LA ROSA

ASESOR: _____

MM. JOSÉ RAÚL GARCÍA SEGURA

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO
ÁREA DE TITULACIÓN

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE QUINTANA ROO
DIVISIÓN DE CIENCIAS, INGENIERÍA Y TECNOLOGÍA

CHETUMAL QUINTANA ROO, MÉXICO, DICIEMBRE DEL 2022

Dedicatoria

Este trabajo se lo quiero dedicar a mi familia, mis padres José Luis y Elizabeth, quienes a pesar de los malos momentos por los que pasaron siempre me enseñaron a salir adelante. Gracias a ellos, pude alcanzar este punto de mi vida, habiendo crecido también como persona.

A mi hermanos, Abril, Sofía, Andrea y Humberto, quienes son mi soporte emocional y mis risas en muchas ocasiones, a pesar de que ellos también han pasado por momentos complicados y difíciles, específicamente durante la pandemia.

A mi cuñado Eduardo quien es un gran amigo y compañero, quien siempre me hace reír con sus ocurrencias, al igual de ser una persona correcta para mi hermana y mi sobrina.

Mi sobrina Hana, quien aunque tomará muchos años para que lea esto, quiero que sepa que ella vino a cambiar nuestras vidas para siempre; lo hizo conmigo durante este periodo de maestría. Ella ha sido el motor principal de crecimiento de todos en mi familia.

A mis amigos, Rijark, Francisco, Angel, Alrik, José Mario y Rodrigo, quienes aún a la distancia me apoyaron. Unos siendo una forma de no estresarme demasiado en mi maestría y en otras para motivarme a continuar. En especial José Mario y Rodrigo que hoy los considero mis "padawans"de la vida, ya que ellos se están formando en este hermoso camino de la ingeniería y la investigación.

A la gente esporádica que estuvo en esta etapa de mi vida; de alguna u otra forma, fueron un soporte y aprendizaje para poder llegar a esta meta, la cual es la culminación del posgrado.

Agradecimientos

Quisiera agradecer al Programa Nacional de Posgrados de Calidad (PNPC) del Consejo Nacional de Ciencia y Tecnología (CONACyT) por haber brindado el apoyo económico durante los dos años que duro el posgrado en Mecatrónica de la Universidad Autónoma del Estado de Quintana Roo (UAEQROO).

Al personal docente y administrativo UAEQROO por haberme facilitado el acceso a las instalaciones necesarias para poder desarrollar las actividades del posgrado, así como también, el haberme brindado los conocimientos requeridos como maestro en mecatrónica y finalizar mi trabajo de tesis.

Resumen

Actualmente se ha incrementado de forma considerable el uso de tecnologías emergentes, las cuales permitan la automatización de procesos de forma continua, ya sea para el hogar o para la industrial. Ante este reto, o paradigma, surgen ideas novedosas para afrontar tal tecnificación como lo es el llamado Internet de las Cosas (IoT por sus siglas en ingles).

IoT se puede considerar como una tecnología emergente la cual permite brindar soluciones para transformar diferentes operaciones y roles en varios sistemas industriales como sistemas de transporte y de manufactura.

Sin embargo, los alcances de IoT no solo recaen en el uso doméstico sino también en la industria; de hecho, Industria 4.0 es la implementación de sistemas industriales que cuentan con IoT. En este sentido; hoy en día es necesario implementar estudios específicos para sus escenario o ambientes de operación, infraestructura de red, entre otros. Ante esta situación, es importante realizar un análisis de los diferentes estándares de comunicación tales como BLE, LoRa, NB-IoT, Sigfox, Wifi, 5G ZigBee, entre otros.

Realizar un estudio de los esquemas de transmisión de datos disponibles para sistemas IoT, específicamente, orientados a Industria 4.0, es de suma importancia debido a que el escenario de propagación es afectado por las máquinas y dispositivos que lo conforman. Es decir, el desempeño de los sistemas orientados a Industria 4.0 debe ser considerado con la finalidad de definir las estrategias de mejora para el sistema. Por ejemplo, establecer algoritmos de ruteo que busquen la mejora del sistema con base al consumo energético, nivel de fuentes de almacenamiento energético, capacidad del canal, latencia, entre otros.

Este trabajo reporta el análisis del desempeño de un sistema de transmisión a largo alcance con el objetivo de mejorar el desempeño de esquemas IoT-Industria 4.0. Para cumplir el objetivo de este trabajo, fue necesario realizar un estudio de las métricas de los estándares de comunicaciones ampliamente utilizados en la IoT-Industria 4.0,

así como también, realizar un estudio de las métricas de desempeño de sistemas de comunicaciones tales como: SNR, RSSI y consumo energético. Así mismo, una vez analizado el desempeño de un sistema de comunicaciones, dos algoritmos de ruteo fueron implementados con la finalidad de observar su desempeño ante distorsiones generadas por el ruido industrial: el algoritmo Dijkstra y el algoritmo AODV (Ad hoc On-demand Distance Vector). Lo anterior, como propuesta de alternativa de uso en sistemas de comunicaciones IoT-Industria 4.0.

Índice general

Índice de figuras

Índice de tablas

Tabla de Acrónimos

Acrónimo	Significado
IoT	Internet of Things
LoRa	Long Range
LPWAN	Low-Power Wide Area NetWork
NB-IoT	Narrow Band IoT
LoRaWAN	Long Range Wide Area Network
SNR	Signal-to-Noise Ratio
RSSI	Received Signal Strength Indicator
IP	Internet Protocol
LTE	Long-Term Evolution
QPSK	Quadrature Phase-Shift Keying
CSS	Chirp Spread Spectrum
CDMA	Code Division Multiple Access
FSK	Frequency Shift Keying
3GPP	Third Generation Partnership Project
BPSK	Binary Phase-Shit Keying
ISM	Industria Scientific and Medical
BR/EDR	Bluetooth basic Rate/Enhanced Data Rate
BLE	Bluetooth Low Energy
IDE	Integrated Development Enviroment
WSN	Wireless Sensor Network
AODV	Ad hoc On-demand Distance Vector

Capítulo 1

Introducción

Actualmente se ha incrementado de forma considerable el uso de tecnologías, las cuales permitan la automatización de procesos de forma continua, ya sea para el hogar o para la industrial. Ante este reto, o paradigma, surgen ideas novedosas para afrontar tal tecnificación y que es a través del llamado Internet de las Cosas (IoT por sus siglas en ingles) [1].

IoT se puede considerar como una tecnología emergente la cual permite brindar soluciones para transformar diferentes operaciones y roles en varios sistemas industriales como sistemas de transporte y de manufactura. De igual forma, se puede definir como una red de infraestructura dinámica con capacidades autoconfigurables basados en protocolos de comunicación donde «cosas» físicas tienen identidades que poseen atributos físicos y las personalidades virtuales comunicadas por interfaces inteligentes [2].

Sin embargo, los alcances de IoT no solo recaen en el uso doméstico sino también en la industria; de hecho, Industria 4.0 es la implementación de sistemas industriales que cuentan con IoT. En este sentido, hoy en día es necesario implementar estudios específicos para sus escenario o ambientes de operación, infraestructura de red, entre otros. Lo anterior debido a que los ambientes de propagación de la señal cambiarán acorde al escenario de propagación, y los sensores de red/equipos industriales estarán operando bajo condiciones de ruido distintas en el ambiente industrial (ruido eléctrico generado por motores, equipo y maquinaria pesada).

Las principales características que un sistema IoT requiere son: (i) reducido uso del ancho de banda; (ii) limitado número de paquetes mandados por nodo al día; (iii) un amplio número de dispositivos finales conectados simultáneamente; (iv) conexiones de largo alcance y (v) dispositivos finales de bajo costo [3]. Ante esta situación es importante realizar un análisis de los diferentes estándares de comunicación tales como BLE, LoRa, NB-IoT, Sigfox, Wifi, 5G ZigBee, entre otros.

El conocimiento de un estándar de comunicación, y específicamente del ambiente de propagación (o canal de comunicaciones), permite prever las pérdidas en la información enviada entre dispositivos inalámbricos [4].

Siguiendo esta idea, los canales de transmisión analizados en este trabajo se les considera como estándares LPWAN (Low-Power Wide-Area Network). Es decir, son estándares que se enfocan en la eficiencia energética y en una alta eficiencia en la transmisión de la información; sin embargo, sacrifican el alcance máximo (o propagación de la señal) que los datos puedan alcanzar. Dentro de estas tecnologías podemos tener a LoRa, NB-IoT, SigFox, ZigBee, entre otros [5].

El objetivo de este proyecto es llevar acabo un análisis de tecnologías de transmisión de datos de largo alcance para la mejora de esquemas IoT para Industria 4.0.

1.1. Planteamiento del problema

Realizar un estudio de los esquemas de transmisión de datos disponibles para sistemas IoT, específicamente, orientados a Industria 4.0, es de suma importancia debido a que el escenario de propagación es afectado por las máquinas y dispositivos que lo conforman. Es decir, el escenario puede verse afectado por las interferencias generadas los distintos equipos de las fábricas y su configuración dentro del área de trabajo.

Cada tipo de industria maneja diferentes escenarios, los cuales buscan un desempeño diferente de los sistemas con IoT, elegir un sistema de forma incorrecta puede provocar una perdida en la capacidad del canal, así como una baja transmisión de

la información. Para corregir estos errores, el sistema pudiera: a) buscar reforzar las transmisiones incrementando la potencia de la señal, o b) hacer reenvíos de la información deseada. Para casos de dispositivos con requerimientos de uso limitado de energía, es indispensable realizar el análisis energético y el impacto que cada equipo podría tener en su implementación.

Otro aspecto relevante a destacar, es la disponibilidad de los nodos sensores de red al Gateway, ya que, en entornos industriales reales, los nodos no se encuentran en lugares en que puedan comunicarse fácilmente con el Gateway, por lo que se tiene que considerar que para mejorar el desempeño de la red, se deben revisar los diferentes protocolos de enrutamiento que permitan mantener una determinada eficiencia en la operación del sistema.

1.2. Objetivos

1.2.1. Objetivo general

Realizar un análisis del desempeño de tecnologías de transmisión de largo alcance para la mejora de esquemas IoT-Industria 4.0.

1.2.2. Objetivos específicos

- Estudio de estándares LPWAN ampliamente utilizados en la Industria IoT.
- Estudio de métricas de desempeño de sistemas IoT (capacidad del canal, SNR, RSSI, consumo energético).
- Evaluación de métricas de desempeño de sistema IoT bajo un estándard LPWAN.
- Implementar esquemas de ruteo en sistemas LoRAWAN para mejora del desempeño en entornos industriales.
- Redacción del trabajo de tesis.

1.3. Justificación

Hoy en día, las tecnologías de transmisión a largo alcance para sistemas IoT para Industria 4.0 no han sido completamente estudiadas. Sin embargo, el desempeño de este tipo de sistemas debe tomarse en cuenta con la finalidad de definir estrategias de ruteo que busquen su mejora con base al consumo energético, capacidad del canal, entre otros. En este sentido, este trabajo sentará las bases en cuanto al estudio del desempeño de sistemas de comunicaciones a largo alcance. Como inicio, se utilizará el esquema de comunicación IoT Industria 4.0 LoRa para realizar las primeras mediciones de desempeño, en conjunto con la implementación de dos algoritmos de enrutamiento. Es decir, se realizará un estudio de los algoritmos de ruteo Dijkstra y AODV, así como también, se evaluará su desempeño como propuestas alternativas para satisfacer las necesidades tecnológicas de la Industria 4.0.

Capítulo 2

Marco teórico

Para poder realizar el análisis de los diferentes estándares de comunicación, es necesario realizar una búsqueda conceptual de cada uno de ellos, es por ello que a continuación se detallarán algunos de los sistemas LPWAN que son posible de utilizar para desarrollar Industria 4.0.

2.1. LoRa

LoRa es una tecnología de protocolo abierto que se basa en los chips de Semtech. LoRa emplea una modulación de espectro disperso por chirp (CSS por sus siglas en inglés) cuando se realiza una modulación por pulsos de chirp para aumentar la resiliencia frente a las interferencias tales como el efecto Doppler y los trayectos múltiples. LoRa ha adquirido mucha popularidad debido a que se ha convertido en uno de los esquemas de comunicación más eficientes para sistemas IoT [5, 6].

LoRa emplea el protocolo de comunicación basado en el estándar LoRaWAN. Realizando una analogía del estándar WiFi con LoRa y LoRaWAN, LoRa es la forma de conexión y LoRaWAN se puede interpretar como el protocolo IP. LoRaWAN (Long-Range Wide Area Network) emplea 3 tipos direccionamiento para diferentes dispositivos finales. La clase A es bidireccional, la clase B es bidireccional con ranuras de recepción programadas y la clase C es bidireccional con ranuras de recepción máximas. Cabe destacar que mientras mas bajamos en las clases la eficiencia energética será menor,

sin embargo, la latencia ira disminuyendo, es decir, que en la clase A existe una mayor eficiencia energética, pero la latencia es mucho mayor, mientras que en la clase C se tendrá un mayor consumo de energía pero la latencia será reducida considerablemente [5].

LoRaWAN no soporta comunicación de dispositivo a dispositivo, solo soporta comunicaciones entre dispositivo y puerta de enlace. Para lograr una comunicación de dispositivo a dispositivo con LoRaWAN es necesario pasar la información a través de la puerta de enlace. Sin embargo, hoy en día existen referencias donde se reportan ejercicios de transferencia de información de dispositivo a dispositivo a través de LoRa [7, 8, 9, 10, 11].

Entre las aplicaciones de LoRa tenemos los siguientes ejemplos: agricultura inteligente, fábrica e industria, seguimiento logístico, cuidado de la salud, administración de aeropuertos, entre otros. Por ejemplo, en [12] se explica la aplicación de LoRa para medir el flujo de aire en almacenes congelados para la preservación de manzanas. La razón principal del uso de LoRa para dicho trabajo, es debido a que el uso del protocolo LoRaWAN, al no tener la propiedad de soportar multi saltos, podría representar un menor consumo de energía que otros protocolos de red que si emplean esta propiedad.

Como se mencionó anteriormente, LoRaWAN emplea 3 diferentes clases para direccionar la información entre los dispositivos finales. A continuación se detallan las características primordiales de cada una de ellas.

- **Clase A:** La clase A es igualmente conocida como la clase de los dispositivos finales con comunicación bidireccional. En esta comunicación el dispositivo final realiza un envío en la transmisión de subida y recibe dos de transmisión de bajada. La forma en como se realizará la comunicación dependerá de las necesidades de cada dispositivo final con una pequeña variación de tiempo aleatorio. En este tipo de clase se dice que existe una menor potencia en las aplicaciones IoT ya que se requiere una comunicación de bajada breve seguida de un mensaje de comunicación de subida [13].

- **Clase B:** En esta clase, los dispositivos finales tienen comunicación bidireccional por lotes de recepción programados. En este caso los dispositivos mantienen la misma recepción de información que en la clase A pero en este caso se añaden ventanas de recepción en un tiempo programado. Esta calendarización de tiempo es enviada desde la estación base para poder abrir las ventanas de recepción de información en los dispositivos finales. Esto permite que el servidor de la red sepa cuando el dispositivo final esta escuchando [13].
- **Clase C:** En esta clase la comunicación es bidireccional con ranuras de recepción programadas. En este caso las ventanas de comunicación están en su mayoría abiertas a la comunicación con el servidor base, esto puede producir un mayor consumo energético de los aparatos que se están comunicando, pero reduce considerablemente la latencia de la información [13].

Las principales características del estándar LoRa a nivel capa física son [14]:

- Ancho de banda y frecuencia escalables; es decir, LoRa puede modificar su ancho de banda para ser de banda estrecha o banda ancha cuando requiera la red.
- Resistencia al desplazamiento Doppler .
- Inmunidad al desvanecimiento y al multirruta, especialmente en zonas urbanas.
- Robustez a la interferencia.

2.2. NB-IoT

NB-IoT o mejor conocido como Internet de las cosas de banda estrecha (por sus síglas en inglés), forma parte del estándar de comunicaciones LTE (Long Term Evolution), el cual es utilizado para conectarse a Internet.

A diferencia de las otras tecnologías del estándar LTE, NB-IoT busca la reducción en los costos de los dispositivos finales que empleen este estándar de comunicaciones, además de, priorizar en el consumo de energía en los dispositivos. Así mismo, emplea las mismas bandas de frecuencias licenciadas por LTE y utiliza una modulación QPSK

(Quadrature Phase-Shift Keying).

Las mayores aplicaciones de NB-IoT recaen en el desarrollo de aplicaciones con conectividad a la red de telefonía celular. Por ejemplo, se tienen aplicaciones relacionadas a bandas inteligentes, bicicletas inteligentes, monitoreo de niños, localizadores de mascotas entre otras. La mayoría de los dispositivos que usan NB-IoT son dispositivos pequeños que no dependen de un punto de conexión especifico (como lo es WiFi) sino mas bien pueden emplearse con tecnología celular para poder brindar comunicación entre los equipos, así mismo, los equipos son equipos de baja potencia, por lo que no requieren de un consumo energético alto.

En la Tabla 2.1, obtenida de [15], se detallan algunas de las características físicas principales entre LoRA y NB-IoT.

Parámetros	LoRa	NB-IoT
Espectro	Sin licencia	Ancho de banda con licencia de LTE
Modulación	CSS	QPSK
Ancho de banda	500-125 kHz	180 kHz
Banda de operación	902-928 MHz	700, 800, 1800, 1900 MHz
Tasa de datos pico	290 bps-50 kbps (bajada/subida)	Bajada: 234.7 kbps subida: 204.8 kbps
Balance de enlace	154 dB	150 dB
Número máximo de mensajes por día	Ilimitado	Ilimitado
Operación del dúplex	–	Semi dúplex
Eficiencia energética	Muy alta	Medianamente alta
Movilidad	Mejor que NB-IoT	Sin movilidad conectada
Densidad de la conexión	La utilizada en NB-IoT	1500 km^2
Vida útil de la batería	Mayor a 10 años	Mayor a 10 años
Eficiencia del espectro	CSS con CDMA es mejor que modulación por desplazamiento de frecuencia (FSK)	Mejorado por operación independiente.
Capacidad de tráfico de área	Depende del tipo de gateway	De 40 a 55k dispositivos por hogar.
Inmunidad a interferencias	Muy alta	Baja
Corriente pico	32 mA	120-300 mA
Corriente en suspensión	1 μ A	5 μ A
Estandarización	Estándar por defecto	3GPP

Tabla 2.1: Características físicas de LoRa y NB-IoT.

2.3. Bluetooth

A diferencia de LoRa y NB-IoT, Bluetooth es un estándar de comunicación de corto alcance que opera en la banda de los 2.4 GHz de acuerdo al estándar de las frecuencias industriales, científicas y médicas (ISM por sus siglas en ingles).

La especificación Bluetooth está dividida principalmente en dos partes, la primera llamada BR/EDR la cual se conoce como Bluetooth clásico, la otra es conocida como Bluetooth de bajo consumo energético (BLE por sus siglas en ingles) el cual fue implementado en la versión 4.0 de Bluetooth. Ambos son dos protocolos completamente diferentes; sin embargo, BLE ha adquirido cierto interés en la industria 4.0 debido a que consume mucho menos energía que BR/EDR y por su soporte en redes de mallas [16].

Como se mencionó, BLE fue implementado en la versión 4.0 de Bluetooth en el año 2009; sin embargo, por diversos motivos, como es el de la seguridad del protocolo, se prevé que la versión 4.0 será descontinuada durante el 2022. Hasta ahora, han existido diversas revisiones del protocolo y actualmente se esta trabajando en estandarizar la versión 5.0 de BLE, la cual contará con mejoras a la problemática previamente descrita [16].

Desde que fue lanzado BLE, se consideró que podrá establecer una comunicación punto a punto y una comunicación por broadcast. Estas topologías permiten la creación de pequeñas redes de sensores basadas en IoT; sin embargo, en aplicaciones mas extensas (o de grándes de coberturas), como aquellas relacionadas a ciudades inteligentes, resultan ser limitadas. Es por ello que, se opta por la creación de redes BLE basados en mallas, las cuales aprovechan las capacidades de broadcast que ofrece BLE para crear redes de mallas inundadas [17].

La red malla BLE puede comunicarse con los demás dispositivos de esa red siempre y cuando la topología física lo permita. Las redes malla BLE se diseñaron con la finalidad de admitir redes de múltiples saltos a gran escala, por lo tanto, es ideal para la automatización de redes de sensores, seguimiento de activos y cualquier otro caso de

uso que requiera la comunicación segura de diez a miles de dispositivos conectados [18].

2.4. Sigfox

Sigfox es estándar de comunicación LPWAN que ofrece conectividad entre dispositivos finales en redes IoT el cual esta basado en tecnologías patentadas. Sigfox emplea estaciones base equipadas con software cognitivo de radios definidos y conectados a servidores finales basados en redes IP. Los dispositivos finales están comunicados a las estaciones base empleando modulación por desplazamiento de fase binaria (BPSK) en una ancho de banda ultra estrecha (100 Hz). Sigfox emplea anchos de banda no licenciados, reservados para uso industrial, científico y médico (ISM), para diferentes regiones, en Europa el ancho de banda es 868 MHz, 915 MHz en Norte América y 433 MHz para Asia. En un primer momento, Sigfox únicamente soportaba comunicación unidireccional, sin embargo, este ha evolucionado hasta ser de comunicación bidireccional con comunicación asimétrica. La longitud de carga útil en los mensajes uplink de Sigfox es de 12 bytes. [13]
Sigfox ha ganado amplia popularidad debido a su bajo consumo energético, su amplia cobertura y un largo alcance de la señal. De acuerdo con el artículo [19] Sigfox posee un potenciales aplicaciones en el campo de la generación de energía, implementación de redes inteligentes, ciudades inteligentes, edificios inteligentes, tecnologías de comunicación, producción de alimentos, etc. [20]

2.5. Métricas de desempeño

Una vez revisados los estándares ampliamente utilizados para el desarrollo de aplicaciones de IoT-Industria 4.0, es importante remarcar cuales son los parámetros de desempeño a considerar en una red de sensores inalámbricos y a estudiar en este trabajo. Acorde a [13], se pueden considerar los siguientes:

- **Calidad en el servicio:** LoRa junto a Sigfox emplean espectros sin licencias y

protocolos de comunicación asíncronos. Estos estándares de comunicación pueden contrarrestar los efectos negativos ante interferencias, trayectorias múltiples y desvanecimientos de la información ante ciertos escenarios de propagación. Sin embargo, su calidad en el servicio tiende a ser mas bajo que NB-IoT que cuenta con un espectro licenciado por LTE y un protocolo de comunicación síncrono. Comparativamente hablando, la calidad en servicio de NB-IoT es mucho mayor que LoRa, sin embargo, que la calidad sea mas alta produce que los costos de producción de nodos NB-IoT sean mas caros. Es por ello que se recomienda únicamente emplear NB-IoT en aplicaciones donde se tenga que garantizar la calidad en el servicio, mientras que se recomienda utilizar LoRa para aplicaciones donde la QoS no sea un factor determinante y de impacto para el desempeño del sistema.

- **Consumo de la batería y latencia:** Sigfox, LoRa y NB-IoT se encuentran la mayor parte del tiempo en modo de suspensión cuando los dispositivos finales se encuentran fuera de operación. Lo anterior reduce el consumo de energía y amplía la vida útil de las baterías. Sin embargo, debido a que NB-IoT maneja comunicación síncrona y una mejor QoS que LoRa y Sigfox conlleva a que los dispositivos finales de NB-IoT tengan que consumir una mayor cantidad de energía adicional, lo cual afecta la vida útil de las baterías. El consumo adicional en NB-IoT se debe principalmente a que se desea una latencia mucho menor en comparación con Sigfox y LoRa. Por otro lado, LoRa ofrece la alternativa de comunicación de clase C, en donde se realiza un consumo adicional de energía en favor de reducir la latencia. En este sentido, para lograr consumos de energía menor es deseable utilizar Sigfox o LoRa Clase A. Sin embargo, si es necesario reducir la latencia en el sistema de transmisión de datos, es preferible utilizar NB-IoT o LoRa Clase C.

- **Escalabilidad y longitud de carga útil:** Tanto Sigfox, LoRa, NB-IoT permiten la escalabilidad de sus redes siendo NB-IoT la que cuenta con un límite de hasta 100 mil dispositivos conectados a una misma celda, mientras LoRa y Sigfox tienen un límite de 50 mil. NB-IoT permite una transmisión de datos de hasta 1600 bytes, mientras que LoRa solo permite hasta 243 bytes y Sigfox solo 12 bytes lo que limita su uso en aplicaciones donde se requiere hacer envíos con grandes tamaños de datos.

- **Cobertura y alcance de red:** Para este apartado se considera a Sigfox como

la mejor opción ya que tiene una cobertura de mas de 40 km, por lo que podría cubrir una amplia zona, En este sentido Bélgica un país cuya extensión territorial es de 30500 km^2 podía ser cubierto en su totalidad con solo 7 estaciones Sigfox. Para el caso de LoRa se tiene una extensión menor a los 20 km y NB-IoT alcanza los 10 km de extensión. Además, para NB-IoT la principal limitación se dará a medida de la disponibilidad de estaciones LTE, por lo que no se recomienda su uso en zonas rurales o suburbanas donde la cobertura LTE no sea alcanzable.

- **Modelo de implementación:** El despliegue e implementación de sistemas basados en NB-IoT inicia apenas en 2016, por lo que es una tecnología que requerirá de tiempo para madurar. En este sentido Sigfox y LoRa ya cuentan con sistemas maduros y su comercialización se da en varios países del mundo. En este caso LoRa permite una implementación mucho mas flexible que NB-IoT y Sigfox, además, LoRa permite la implementación de red LAN con puerta de enlace LoRa.

- **Costo:** En este apartado hay que considerar el costo de implementación y el costo del uso de la licencia del espectro así como los costos de los dispositivos. Si bien el costo de implementación varía de acuerdo a las consideraciones mencionadas, Sigfox y LoRa poseen una licencia libre de uso del espectro, por lo que reduce los costos de implementación. Ahora bien, de acuerdo con [13] el costo de implementación de Sigfox es mas alto que LoRa, por lo que la implementación de estaciones LoRa es mucho mas rentable.

De acuerdo con [21] una forma de comparar el rendimiento de sistemas de comunicaciones es considerando los parámetros como retardo de la red, rendimiento, tasa de éxito, latencia, consumo de energía y tiempo de vida de la red.

- **Retardo de la red:** El retardo de la red es una métrica que mide el promedio del tiempo que tarda un paquete en ser enviado de un dispositivo a otro y cuanto tiempo se tarda en obtener una respuesta de recibido. Básicamente, con este parámetro podemos identificar cuanto tarda en enviarse y en recibirse un paquete.

- **Rendimiento:** El rendimiento (o performance) define a la cantidad de paquetes que es posible recibir por segundo. Éste se puede considerar como una medida externa de la eficiencia de un protocolo.

- **Tasa de éxito:** Este parámetro mide el número de paquetes enviados contra el número de paquetes recibidos, un ejemplo para entender mejor este parámetro es con el comando «ping» el cual envía un número de paquetes a un destino y si éste contesta apropiadamente será un indicativo de que existe una comunicación correcta entre ambos dispositivos.
- **Latencia:** La latencia promedio mide la cantidad de tiempo que existe entre el inicio de un dato por parte del dispositivo de envío, y su arribo al dispositivo destino.
- **Consumo de energía:** En el consumo de energía se mide cuanta energía consume cada nodo considerando cuando el nodo está activo censando datos, activo y enviando datos, en reposo, etc.
- **Tiempo de vida de la red:** Esta métrica hace referencia al momento desde que se inicia la comunicación entre el dispositivo emisor y el receptor, también se considera el tiempo en el que la red se mantiene conectada de entre los diferentes dispositivos finales.

Ahora bien, otro aspecto importante a considerar es el diseño de la red de sensores, la cual será de gran importancia para el análisis de práctico de los diferentes modelos de canal, siendo para este caso de estudio LoRa. Según [22] para poder conseguir un buen diseño de nuestra red de sensores debemos considerar dos aspectos: las características de la red y los objetivos de diseño.

Dentro de las características de red podemos considerar, entre otras cosas, aspectos ya evaluados como el consumo de energía de los nodos y datos como la latencia y el desempeño de la red. De igual forma se anexan elementos como las especificaciones de aplicaciones, la facilidad del cambio de topología, la configuración de los nodos, entre otros.

En los objetivos de diseño podemos considerar el costo de los nodos, el tamaño de los nodos, escalabilidad, adaptabilidad, fiabilidad, tolerancia a fallos, seguridad, calidad en el servicio, entre otros.

Ahora bien, estas propuestas de métricas son alternativas que se tienen para realizar el análisis del desempeño de LoRa. De forma adicional se puede considerar otro tipo de parámetros (o métricas) como lo es el RSSI (Received Signal Strength Indication) y el SNR (Signal-to-Noise Ratio).

El RSSI se puede resumir como la capacidad que tiene un dispositivo para «escuchar» el mensaje desde otro que lo envía [23]. Este valor nos permite indicar que tan «fuerte» es la señal de nuestro nodo y así poder identificar su rango de alcance.

La figura 2.1, extraída del artículo [15], ilustra como se comporta la señal del RSSI.

Figura 2.1: Comportamiento del RSSI de débil a fuerte.

Analógicamente, el RSSI en LoRa es como si midiéramos en alcance de un módem WiFi desde nuestra computadora, por lo que este indicador nos permite saber que tan fácil es que se pierdan los paquetes debido a una baja intensidad de la señal.

El RSSI se mide en dBm, siendo éste un valor negativo. Por ejemplo, cuando la señal RSSI se encuentre muy próxima a cero, será un indicativo que se tendrá una buena calidad de señal, mientras que, si se aleja de cero (negativamente), la señal estará con una baja calidad. Para el caso de LoRa el valor mínimo de RSSI es de -120 dBm indicando que se tiene un valor de señal débil [23].

El SNR por otro lado se puede definir la intensidad de una señal con respecto al ruido ambiental. Si la señal es mas fuerte que el ruido, se puede decir que no existirán interferencias en la comunicación de los dispositivos, por otro lado, si la señal es más débil que el ruido ambiental, entonces existirán interferencias [23].

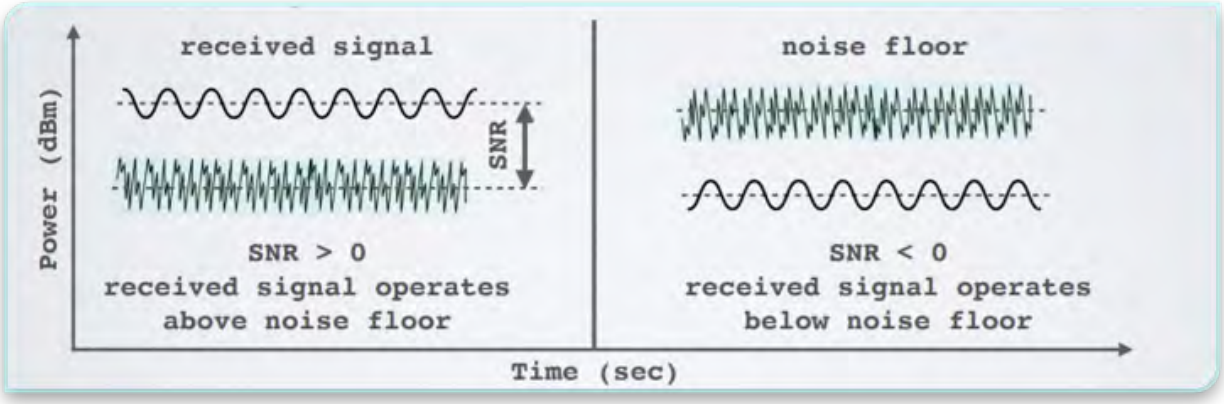

Figura 2.2: Funcionamiento del SNR.

En la Figura 2.2 podemos observar los valores que puede tener el SNR. Si éste es mayor a cero, la señal estará por encima del ruido de piso, es decir, no habrán interferencias, pero si el SNR es menor que cero, entonces la señal estará por debajo del ruido de piso y existirán interferencias.

Existen diferentes formas de medir estos parámetros; sin embargo, mediante el programa Energía [24], que se puede especificar como un IDE modificado de Arduino, se pueden obtener tales valores dentro de las librerías disponibles del software. De igual forma en el sitio web The Things Network [25] es posible realizar estas mediciones.

2.6. Algoritmos de ruteo: estado del arte.

Un aspecto importante a destacar sobre el análisis del desempeño de los estándares de comunicación es el algoritmo de ruteo que va a emplear el nodo sensor para comunicarse con diferentes dispositivos dentro de la red.

Dentro del artículo [26] se menciona que el diseñar algoritmos de ruteo para las redes de sensores inalámbricos (WSN por sus siglas en ingles) es un proceso complejo ya que se requiere que se consideren las necesidades industriales tales como la latencia, escalabilidad y ahorro de energía.

Dentro del artículo [26] se hace mención de como diversos autores han desarrollado propuestas diferentes para el diseño de algoritmos de ruteo para redes de sensores. Cada propuesta mostrada en dicho artículo contiene bases distintas, lo que hace que cada propuesta sea diferente entre si.

De acuerdo con [27] el mayor reto de los protocolos de enrutamiento para WSN es el garantizar un consumo de energía reducido entre cada uno de los nodos mientras estén en constante comunicación. Esto con el fin de garantizar que la vida útil del sistema se prolongue. De igual forma considera que los nodos sensores conectados a una WSN cumplan con dos funciones primordiales, la primera que es el recopilar información del mundo físico al cual fue conectado, y segundo, el enrutamiento de datos.

Existen diferentes protocolos de enrutamiento para redes de sensores, en [28] se hace un estudio de los protocolos de enrutamiento de múltiples rutas basados en tablas Hash y como éstas ayudan a mejorar la eficiencia energética, reducir el retardo y elevar el rendimiento de los nodos sensores. De igual forma en dicho articulo se realiza un análisis de los protocolos MDART (protocolo de enrutamiento de direcciones dinámicas de múltiples rutas, por sus siglas en ingles), el cual es una adaptación del protocolo DART pero para múltiples rutas. El protocolo propuesto emplea el paradigma de tablas dinámicas Hash para facilitar la escalabilidad entre redes móviles. También, se menciona que este protocolo no contabiliza cargas adicionales para rutas repetidas.

Por otra parte, en [29] se detalla el desarrollo de diferentes propuestas de algoritmos de ruteo para LoRa empleando la comunicación de multi ruta. Cabe mencionar que LoRaWAN no permite la comunicación multi ruta entre los dispositivos finales, por lo que en el mismo artículo se proponen diferentes algoritmos que puedan soportar este tipo de comunicación.

En este mismo sentido, el protocolo LoRaBlink, propuesto en el articulo [10], es un protocolo de comunicación multi ruta para dispositivos LoRa. Este protocolo es diferente a LoRaWAN y busca mejorar algunos aspectos que este protocolo no soporta, como son los multi saltos; es decir, que para alcanzar un nodo, éste pueda «saltar» entre cada nodo de la red hasta alcanzar su destino. LoRaBlink trabaja dentro de la

capa física de LoRa ya que incluye la dirección MAC y el enrutamiento en la misma capa en un protocolo de inundación síncrona. Para proveer un consumo de energía bajo, LoRaBlink propone el uso de mensajes de baliza (igualmente conocido como beacon message), los cuales proveen una señalización de inicio de la sincronización entre los dispositivos, y que éstos sean enviados a través de bajas frecuencias. Esto se consigue debido a que no se requiere un tiempo de sincronización estricto entre los dispositivos a comunicarse.

Así mismo, en [30] se detalla de forma práctica el uso de un protocolo de enrutamiento para dispositivos LoRa empleando diversas topologías como la de anillo, punto a punto y mesh. El protocolo fue desarrollado en el lenguaje de programación Python.

Capítulo 3

Diseño metodológico

La metodología propuesta para el logro de los objetivos propuestos en este trabajo es:

1. **Realizar un estudio del estado del arte de los estándares IoT.** Para este paso, se realizará un análisis de la literatura relacionada al tema con la finalidad de documentarse sobre los trabajos previos, aplicaciones, entre otros. También se seleccionarán los mejores trabajos que puedan enriquecer el proyecto.

2. **Estudiar los protocolos de ruteo IoT.** Se realizará una búsqueda de los estándares IoT, se estudiarán en cuanto a su funcionamiento e implementación, y como éstos han sido reportados/comparados en el estado del arte. Con lo anterior, se elegirán los elementos claves a evaluar en el desempeño de sistemas IoT Industria 4.0.

3. **Definir los parámetros de importancia a medir.** Definir los parámetros que permitirán conocer el desempeño del sistema y posteriormente definirlas como métricas de desempeño (por ejemplo, nivel de la batería, latencia, consumo energético de los nodos, entre otros). Los parámetros pueden ser elegidos acorde a las necesidades de la industria con la finalidad de lograr un funcionamiento óptimo en la red de sensores.

4. **Proponer un algoritmo de comunicación entre nodos WSN-IoT.** En esta etapa se implementará un algoritmo de ruteo que pueda ser utilizado para lograr una eficiencia acorde a una métrica establecida en entornos industriales.

5. **Implementar un escenario de pruebas para un sistema WSN-IoT.** Se implementará un escenario industrial real para la realización de pruebas y comparativas de los algoritmos desarrollados.

6. **Análisis de resultados.** Realizar un análisis de los resultados obtenidos de los algoritmos desarrollados.

Para corroborar la metodología se busca emplear una topología sencilla, la cual emplee 3 nodos LoRa conectados a un Gateway, el cual a su vez estará conectado a la red para que se puedan acceder a los datos que los nodos estén generando. En la figura 3.1 se muestra el diseño de la topología propuesta.

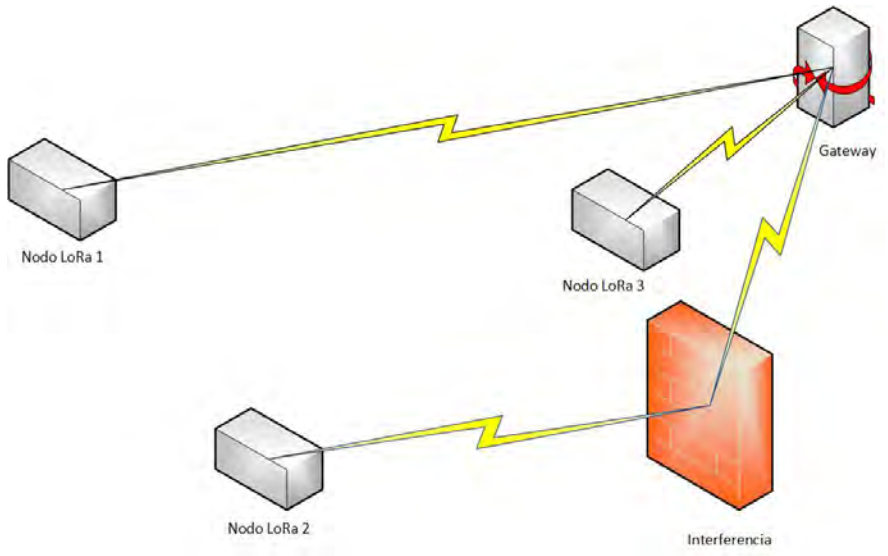

Figura 3.1: Topología propuesta para el estudio.

Como puede observarse, la topología considera una interferencia física entre el nodo 2 y el Gateway; esto es, se busca medir el desempeño del sistema ante un escenario que es muy parecido a un entorno Industrial real, donde las máquinas/dispositivos/nodos no se encuentran en línea de vista con el Gatweay. En este sentido, se busca proponer un algoritmo de ruteo alternativo que pueda evitar la perdida de paquetes en un futuro.

Recordemos que en entornos industriales reales, los nodos sensores se encuentran en lugares poco habituales o de difícil acceso, por lo que es muy común tener problemas a la hora de que el nodo sensor establezca comunicación con el Gateway.

Lo anterior para el protocolo de enrutamiento LoRaWAN puede representar un problema, es por ello que se busca proponer un algoritmo que pueda mitigar este tipo de problemas que recae en el uso de esquemas de enrutamiento multi salto.

Capítulo 4

Resultados

Primeramente se realizó la comunicación base entre un nodo LoRa y un dispositivo Gateway con la finalidad de verificar la conectividad entre ellos y la de su operación de manera correcta.

El nodo sensor implementado se compone de un dispositivo de radio LoRa RFM9x, el cual es acondicionado en una tarjeta de desarrollo launchpad MSP430FR5969 de Texas Instruments. Los dispositivos LoRa permiten ser programados con la herramienta Energía (un IDE basado en Arduino), y con ello, realizar las mediciones del consumo energético del nodo durante su operación.

El Gateway está implementado utilizando un dispositivo Raspberry Modelo B, el cual ha sido configurado para poder realizar una comunicación con los nodos sensores de red. Para poder revisar el estado de trabajo de nuestros nodos deberemos conectarnos remotamente al Gateway empleando el software PuTTY. Esta aplicación, permite revisar los mensajes que el Gateway recibe de los nodos.

Para comunicarnos con el Gateway se ha creado una red de cobertura móvil utilizando una laptop con Windows 11, la cual tiene el objetivo de proporciona la dirección IP a nuestro Gateway. La figura 4.1 ejemplifica el esquema desarrollado.

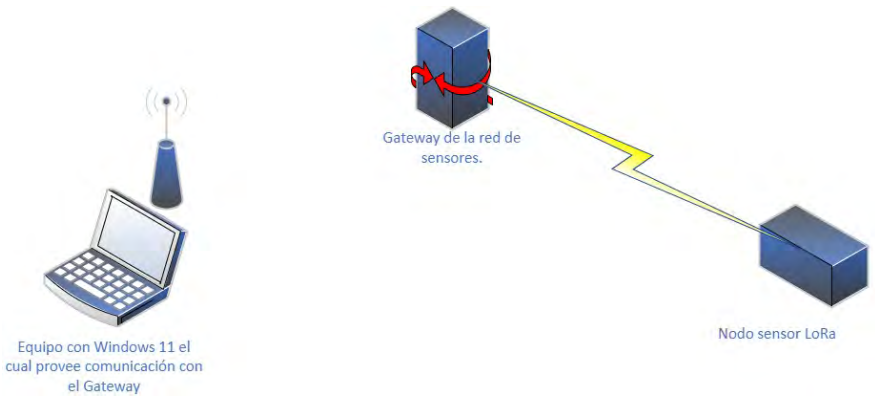

Figura 4.1: Topología de comunicación hecha entre el nodo sensor y el Gateway.

Una vez configurados los nodos y el Gateway, se continuó a realizar el análisis de consumo energético del nodo sensor bajo operación en un escenario industrial. Para ello, se eligió el laboratorio de Ingenierías de la Universidad Autónoma del Estado de Quintana Roo, el cual tiene similitud a un espacio de trabajo industrial real. Dicho laboratorio tiene equipo industrial en operación de forma constante. La figura 4.2 muestra el croquis del laboratorio de ingenierías.

Figura 4.2: Ubicación del laboratorio de ingenierías de la UAEQROO.

Las ubicaciones dentro del laboratorio se muestran en la figura 4.3, en dicha figura

e pueden observar los puntos «G», el cual señala al Gateway, «N», el cual señala el nodo sensor en el segundo piso, y «T» para indicar al túnel del viento del laboratorio de ingeniería de la UAEQROO.

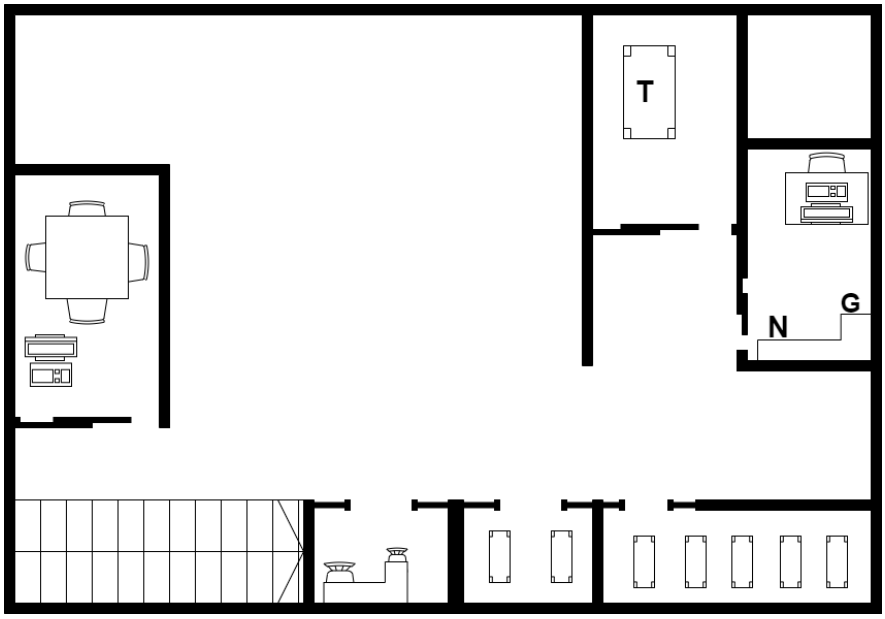

Figura 4.3: Segundo piso del laboratorio de ingeniería de la UAEQROO.

La figura 4.4 muestra el comportamiento del consumo energético del nodo sensor ubicado en la misma habitación del Gateway. La ubicación de dicho Gateway, como pudo observarse en la figura 4.3, está de forma contigua al nodo sensor siendo la separación entre ellos de 2.5 metros.

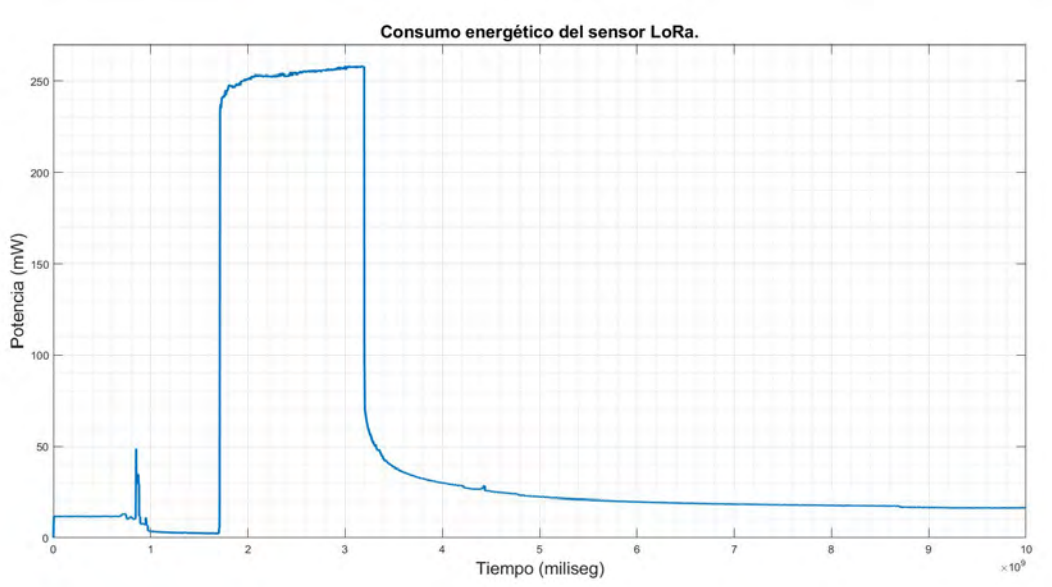

Figura 4.4: Consumo de energía del nodo sensor a una distancia de 2.5 metros.

Las siguientes pruebas han sido realizadas para conocer el desempeño del nodo sensor a nivel de consumo energético ubicando el nodo sensor a diferentes distancias del Gateway.

Las pruebas fueron realizadas en la planta baja del edificio del laboratorio de ingenierías. El croquis se muestran en la figura 4.5 considerando el punto «A» a una distancia de 10 metros, el punto «B» a una distancia de 20 metros y el punto «C» a una distancia de 30 metros del Gateway. La ubicación del Gateway se mantuvo en la misma posición que la especificada previamente en la figura 4.3.

Figura 4.5: Primer piso del laboratorio de ingeniería de la UAEQROO.

Las gráficas mostradas de la figura 4.6 a la figura 4.8 fueron obtenidas sin considerar ruido industrial. Es decir, maquinaria eléctrica en el laboratorio sin operar (maquinaria apagada).

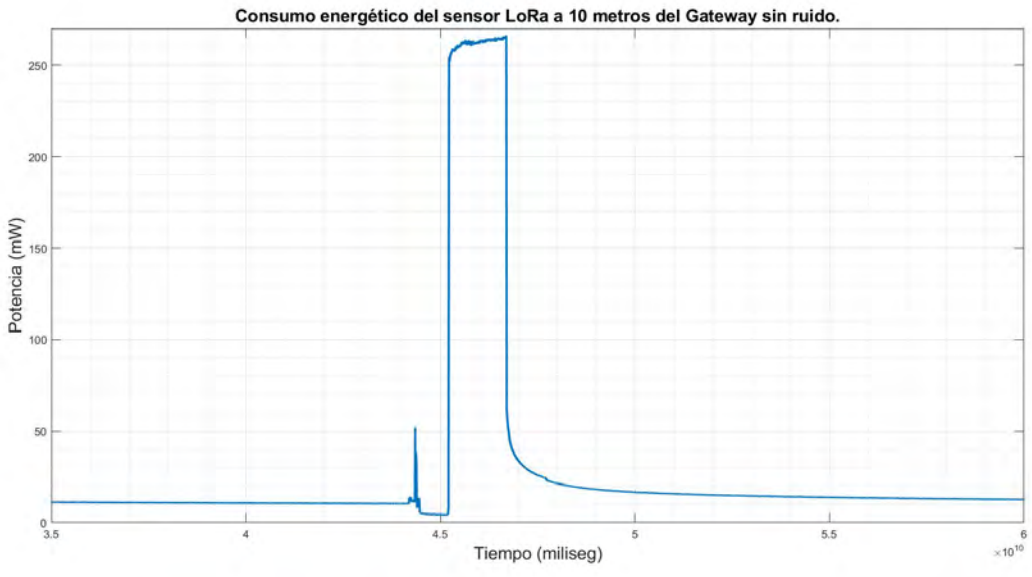

Figura 4.6: Gráfico de consumo de energía a 10 metros sin ruido.

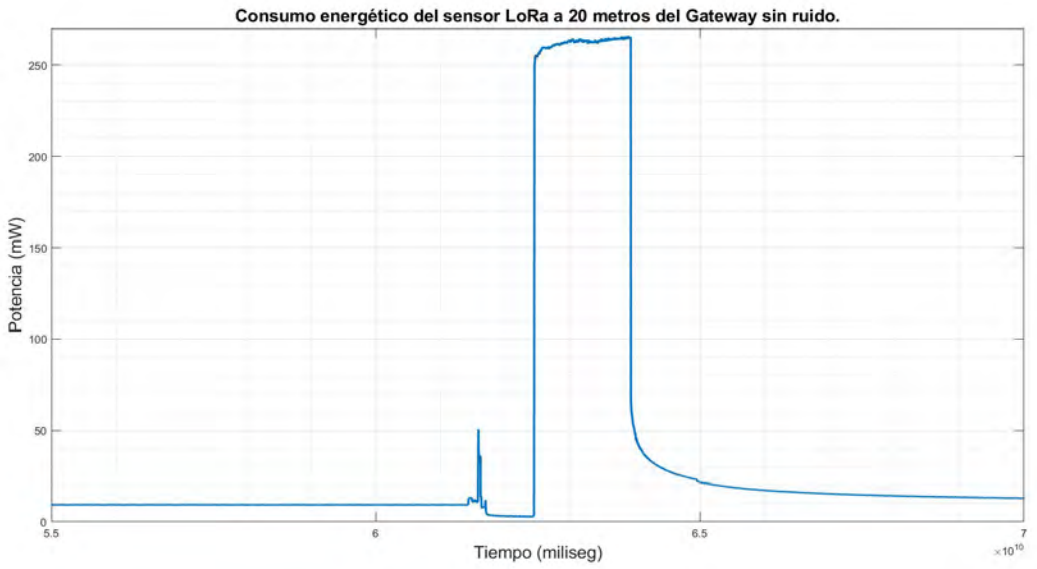

Figura 4.7: Gráfico de consumo de energía a 20 metros sin ruido.

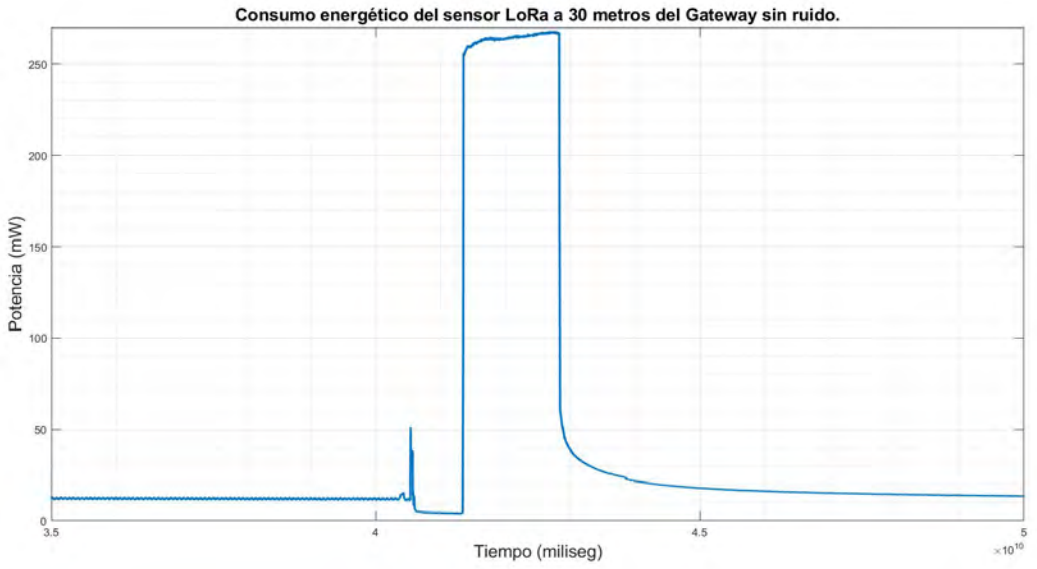

Figura 4.8: Gráfico de consumo de energía a 30 metros sin ruido.

De las figura 4.9 a la figura 4.11 se presentan las gráficas cuando si existe ruido industrial en las mismas ubicaciones.

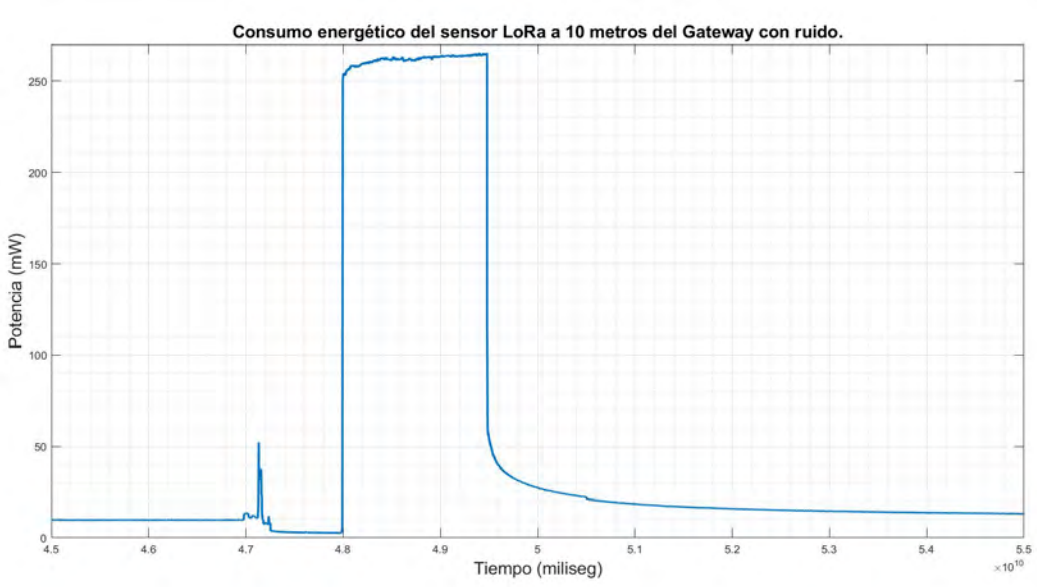

Figura 4.9: Gráfico de consumo de energía a 10 metros con ruido.

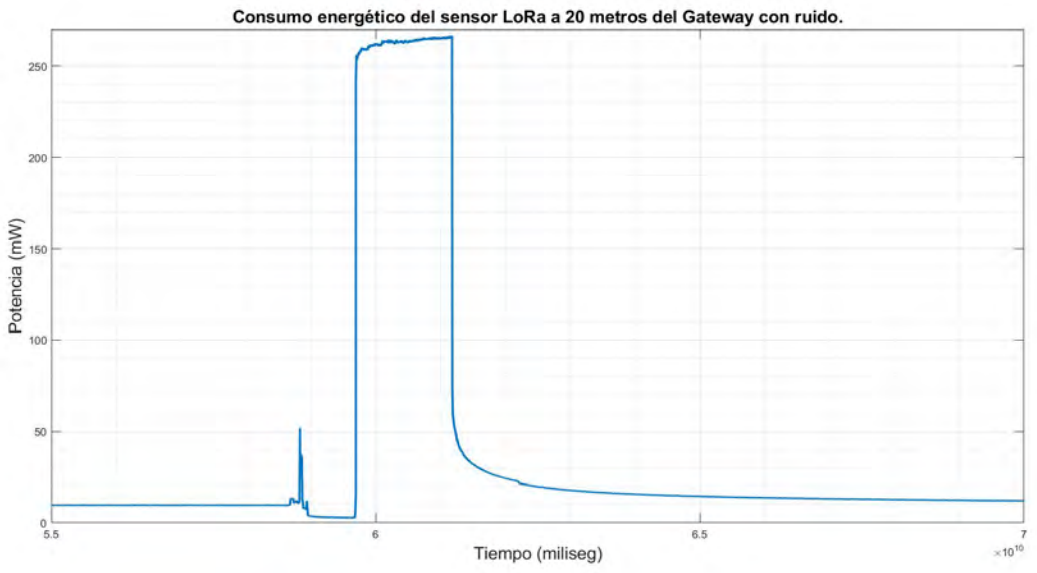

Figura 4.10: Gráfico de consumo de energía a 20 metros con ruido.

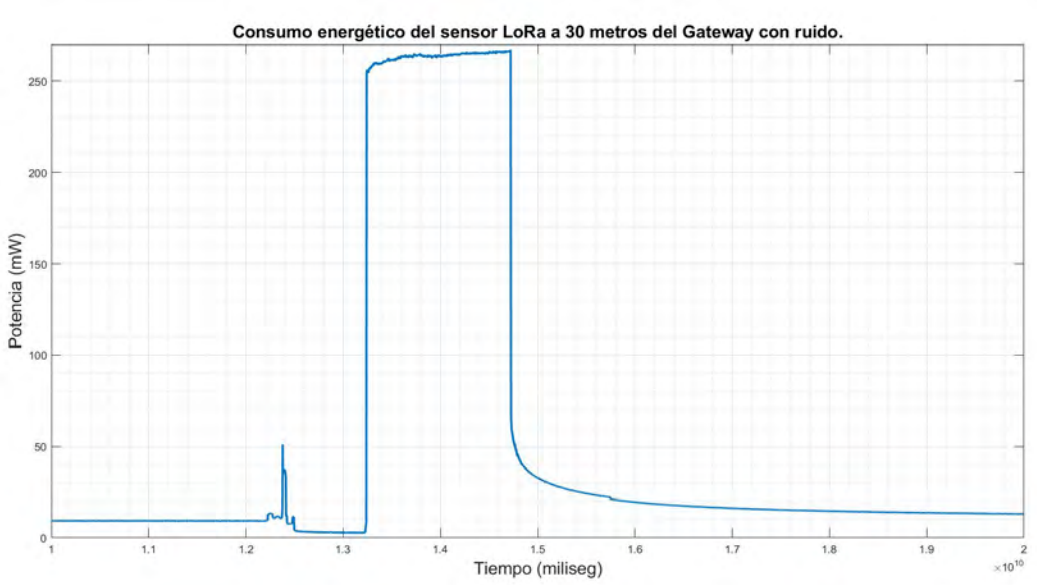

Figura 4.11: Gráfico de consumo de energía a 30 metros con ruido.

El punto «T» en la figura 4.3 es la posición del equipo túnel de viento; de igual forma, en dicha ubicación se realizaron pruebas en condiciones de operación del túnel (es decir, ruido industrial generado por el túnel del viento y maquinaria industrial) y sin ruido industrial. Los resultados obtenidos pueden observarse en las figuras 4.12 y 4.13 respectivamente.

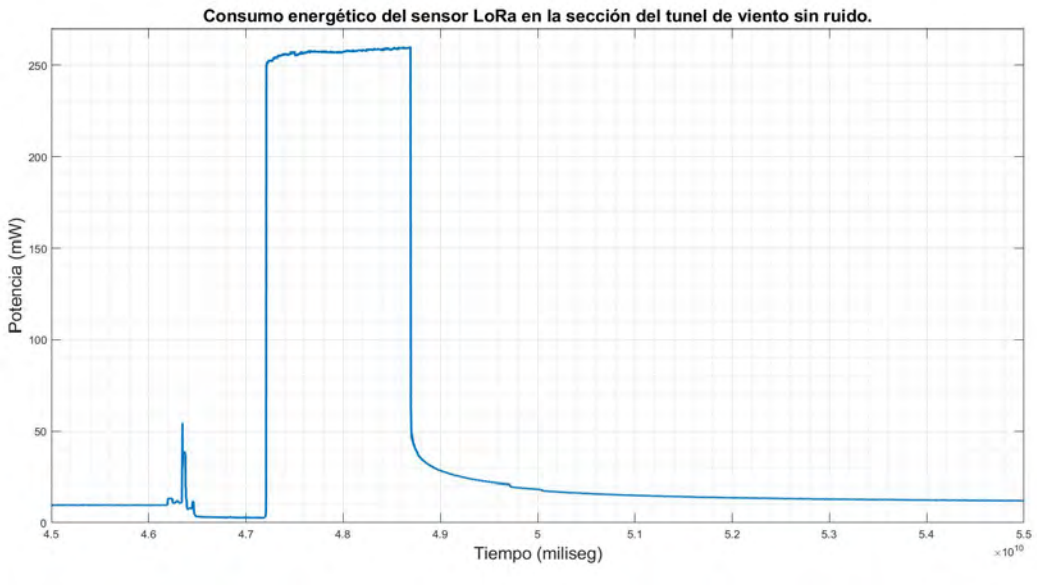

Figura 4.12: Gráfico del consumo energético en la sección del túnel de viento.

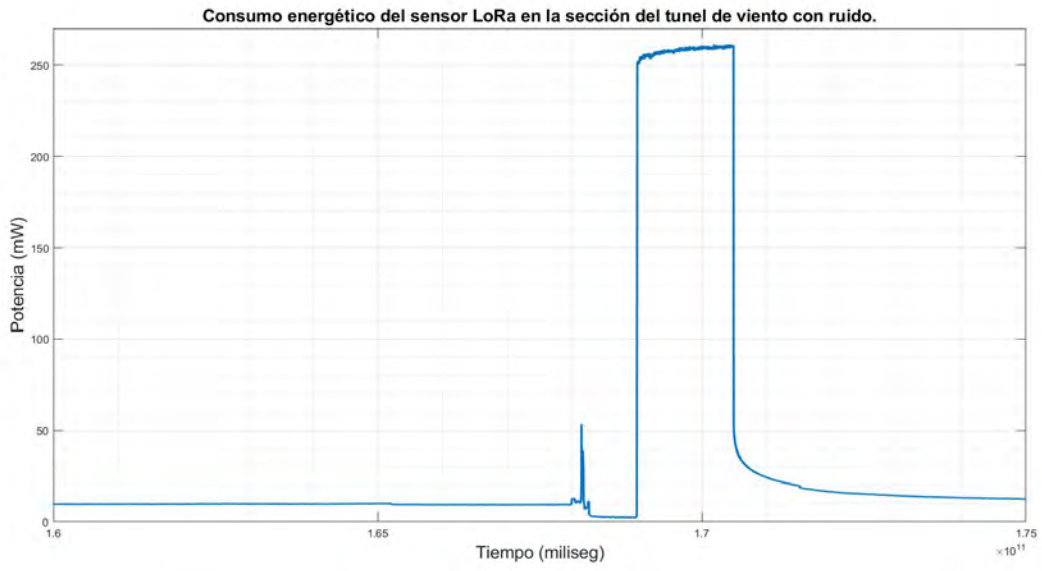

Figura 4.13: Gráfico del consumo energético en la sección del túnel de viento.

Una vez que los perfiles de potencia fueron presentados de la figura 4.4 a la figura 4.13, se continuó a enriquecer el estudio a través de mediciones del RSSI y SNR. La Tabla 4.1 muestra los resultados de la medición de la métrica de RSSI obtenidos considerando un escenario sin ruido. A pesar de que las gráficas de consumo energético no reflejaron cambios significativos por la presencia del ruido, el conocimiento de la métrica de RSSI y SNR nos confirmar el impacto del ruido sobre el escenario de propagación bajo estudio.

Distancia	RSSI del paquete	RSSI	SNR
10 metros	-61	-110	14
20 metros	-68	-114	11
30 metros	-78	-114	10

Tabla 4.1: Métricas sin ruido obtenidas del Gateway del nodo sensor.

Distancia	RSSI del paquete	RSSI	SNR
10 metros	-60	-112	11
20 metros	-66	-112	10
30 metros	-76	-113	9

Tabla 4.2: Métricas con ruido obtenidas del Gateway del nodo sensor.

Como podemos observar, en las tablas 4.1 y 4.3 los valores de RSSI y SNR si son afectados por el ruido industrial.

Finalmente, se hizo una prueba de desempeño del nodo estando cerca de una fuente de generación de ruido industrial. Para ello, el nodo fue ubicado de forma próxima a un túnel de viento. La ubicación del túnel de viento (señalado con el punto T) y el nodo sensor (señalado con el punto N) se puede observar en la figura 4.14. La distancia aproximada entre el nodo sensor y el túnel de viento ha sido de 5 metros.

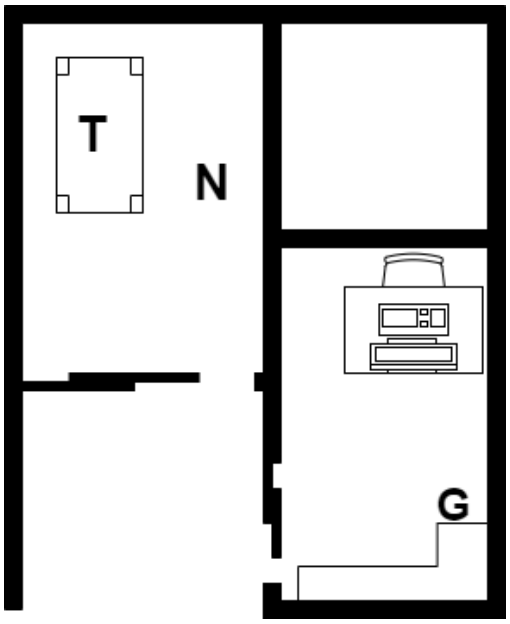

Figura 4.14: Ubicación del nodo sensor en la habitación del túnel de viento.

En la tabla 4.3 se pueden apreciar los resultados de RSSI y SNR del túnel de viento cuando existe la presencia de ruido y cuando no existe la presencia de ruido industrial.

Situación	RSSI del paquete	RSSI	SNR
Sin ruido	-60	-112	11
Con ruido	-71	-108	8

Tabla 4.3: Métricas con ruido obtenidas por el Gateway del nodo sensor.

La presencia de ruido industrial afecta los parámetros de RSSI y SNR, en consecuencia puede concluirse que el desempeño del sistema también se verá afectado. Esto último es muy importante debido a que en aplicaciones industriales, los nodos no se ubican en sitios de fácil acceso (incluso más complicados a los presentados en este caso de estudio), o en sitios cercanos al Gateway. Es es por ello que es necesario estudiar y desarrollar nuevos algoritmos de ruteo, los cuales permitan que un nodo sensor pueda alcanzar a su destino a través de múltiples rutas o acorde a un criterio de desempeño determinado.

4.1. Resultados de implementación de algoritmos

Dentro de la propuesta a desarrollar, se desea que los algoritmos de ruteo seleccionados consideren lo siguiente para la toma de decisiones:

1. Disponibilidad continua de la información que hay entre los nodos y el Gateway (RSSI, SNR, otros a considerar).
2. Capacidad para distinguir al sensor que posee una mejor recepción con respecto al Gateway.
3. Considerar los niveles de batería los nodos sensores con la finalidad de crear rutas alternativa en caso de insuficiencia energética para operar.

Ante estas necesidades se busca proponer algoritmos de ruteo que puedan satisfacer tales requerimientos. Para ello, se ha seleccionado la implementación de un primer algoritmo denominado Dijkstra [31], el cual es un algoritmo que permite encontrar la ruta mas corta entre un nodo y otro con el fin de trazar un camino óptimo para el envío

y recepción de paquetes de información.

4.2. Algoritmo de Dijkstra

El algoritmo de Dijkstra considera la longitud que existe entre cada nodo y esta longitud puede estar asociada a los parámetros que deseemos incluir como RSSI, SNR, nivel de batería, retardo de arribo de paquetes, entre otros. El algoritmo revisa los caminos que existen entre los nodos, y con base a ello, determina la ruta mas corta entre el punto de origen al punto de destino. En aquellos puntos donde no se cumpla la condición de la longitud más corta, el algoritmo se detendrá.

Considere el grafo que se presenta en la figura 4.15, la cual tiene contiene 6 nodos etiquetados de A–F. Dicho grafo nos permitirá mostrar el comportamiento del algoritmo bajo dos consideraciones: a) encontrar la ruta mas corta entre un vértice y otro y 2) obtener los costos de todas las rutas posibles desde un punto de partida hasta cualquier otro nodo.

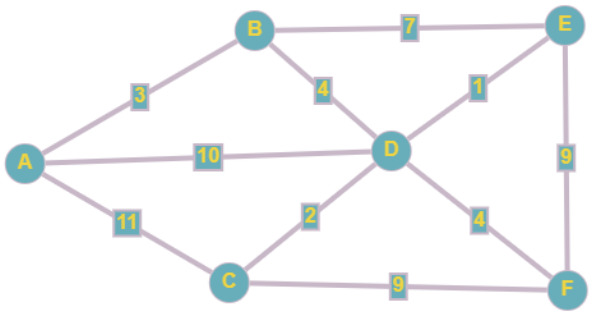

Figura 4.15: Grafo de ejemplo.

Primeramente es necesario definir una matriz de mapeo que conformará las distancias entre cada uno de los nodos. La matriz deberá ser llenada con base a las siguientes condiciones: si el nodo al que vamos a evaluar posee una distancia adyacente con otro nodo, esto se deberá especificar en la matriz (el valor dependerá de los criterios definidos por los parámetros a considerar RSSI, SNR, otros); de lo contrario, si no posee una distancia adyacente con un nodo en especifico, el valor especificado

dentro de la matriz será infinito. Si el nodo es el mismo, la distancia será definida como cero. Por ejemplo, observando la figura 4.15, la distancia entre el nodo A con el nodo B es 3, pero la distancia entre el nodo A y el nodo E es infinito debido a que no hay una distancia adyacente entre ellos.

Una vez analizada la figura 4.15, la matriz de los nodos resultante es la siguiente:

$$\begin{pmatrix} 0 & 3 & 11 & 10 & \infty & \infty \\ 3 & 0 & \infty & 4 & 7 & \infty \\ 11 & \infty & 0 & 2 & \infty & 9 \\ 10 & 4 & 2 & 0 & 1 & 4 \\ \infty & 7 & \infty & 1 & 0 & 9 \\ \infty & \infty & 9 & 4 & 9 & 0 \end{pmatrix} \tag{4.1}$$

Una vez que la matriz de nodos ha sido obtenida, es posible ejecutar el algoritmo Dijsktra. A continuación se detallan los pasos que sigue el algoritmo para encontrar la ruta mas corta:

1. Marcar el nodo inicial con una distancia actual igual a cero. Los nodos restantes deberán ser definidos igual a infinito.
2. Se establece el nodo no visitado con la menor distancia actual al nodo de origen.
3. Para cada vecino que haya con el nodo V (visitado) se le debe sumar la distancia del nodo inicial y su distancia local con los otros nodos no visitados. Si la distancia obtenida en la suma es mayor que la distancia de los nodos nodos no visitados, entonces la distancia entre el nodo V y el nodo no visitado será la nueva distancia que se tendrá en el recorrido.
4. Se marca al nodo V como el nuevo nodo de origen.
5. Se repiten los pasos del 2 al 4 hasta que se complete el recorrido completo.

Teniendo esto en cuenta, se desarrollaron dos programas, el primero de ellos calcula las distancias desde un nodo de origen hacia todos los demás nodos del grafo, mientras que el segundo solo considera la distancia entre el dos nodos previamente establecidos, los cuales serían origen y destino.

A continuación realizará una prueba para comprobar la distancia que existe entre el nodo A y los nodos restantes del grafo. De acuerdo con el primer programa se tienen las siguientes distancias desde el nodo A a los nodos restantes del grafo:

- **Al nodo B:** 3
- **Al nodo C:** 9
- **Al nodo D:** 7
- **Al nodo E:** 8
- **Al nodo F:** 11

De acuerdo con este código, las distancias obtenidas nos proporcionan la ruta mas corta hacia algún nodo deseado; sin embargo, no nos muestran los nodos que debemos seguir para obtener dichas rutas. Para ello se emplea un segundo código el cual nos muestra la ruta que debemos seguir así como la distancia total de recorrido. De acuerdo con el código 2, la ruta para obtener la distancia entre el nodo A y el nodo B es la siguiente.

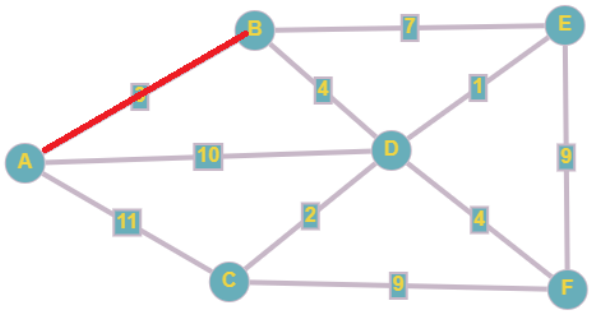

Figura 4.16: Ruta del nodo A al nodo B.

Para la ruta del nodo A al nodo C tenemos una distancia de 9 de acuerdo al primer código y respecto al segundo, la ruta que tenemos para tener esa distancia es la siguiente:

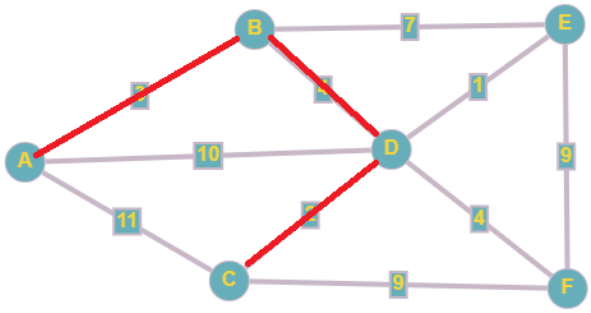

Figura 4.17: Ruta del nodo A al nodo C.

Tal y como podemos observar en la figura 4.17, la ruta mas corta para llegar al nodo C no es precisamente la ruta directa entre el nodo A y el nodo C. Puede verse que la distancia para alcanzar el nodo C es menor recorriendo el camino de los nodos A–B–D–C (con una distancia igual a 9) en lugar de tomar una ruta directa del nodo A al nodo C (con una distancia igual a 11). El valor de la distancia (o costo) igual a 11 puede estar asociado a una ausencia de visibilidad del nodo A al nodo C de forma directa, o un bajo valor de RSSI, o un bajo valor de las reservas de energía del nodo, entre otros factores o parámetros a considerar.

Ahora la distancia que tenemos del nodo A al nodo D es de 7 y su trayectoria es la siguiente:

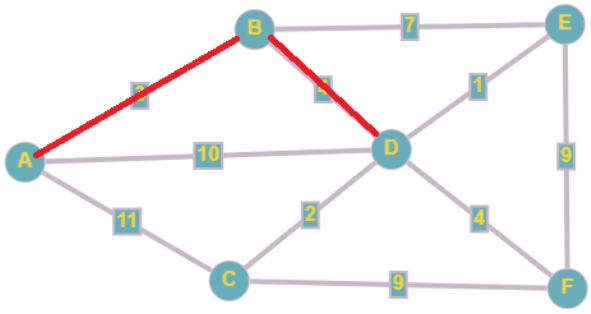

Figura 4.18: Ruta del nodo A al nodo D.

Nuevamente, como en el caso anterior, en la figura 4.18 podemos observar que a pesar de que existe una adyacencia directa entre los nodos A y D, la distancia re-

sulta ser mucho mas larga de que si tomamos la ruta establecida entre los nodos A y B, en donde la distancia total es menor que la distancia adyacente entre ambos nodos.

Para el caso del nodo E, no tenemos una adyacencia directa con el nodo A, por lo que se busca que crear una ruta que tenga menos peso entre ambos nodos, cabe mencionar que la distancia entre el nodo A y el nodo E de forma predeterminada es infinita, por lo que se considera que no existe adyacencia entre ambos nodos. La ruta que se debe seguir desde el nodo A al nodo E con una distancia total de 8 es la siguiente:

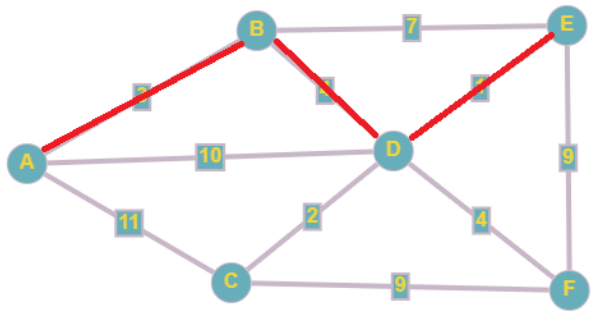

Figura 4.19: Ruta del nodo A al nodo E.

En este caso podemos observar que hay diferentes formas para llegar al nodo E, ya sea desde el nodo B, el nodo F pasando por el nodo D y directamente desde el nodo D, como se ha dicho con anterioridad, se busca que la ruta sea la mas corta, por lo que la ruta mas óptima es la que se tiene en la figura 4.19.

Para finalizar, se muestra la ruta que existe para el nodo A y el nodo F, los cuales no poseen una distancia de adyacencia directa, por lo que por defecto es igual a infinito.

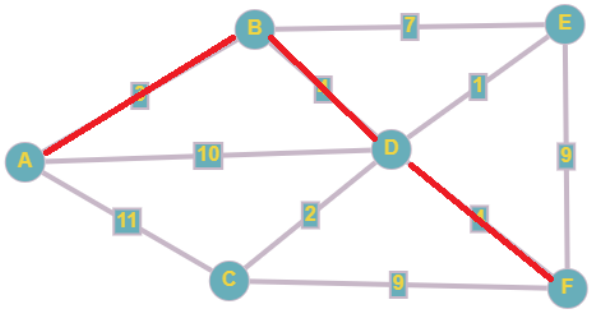

Figura 4.20: Ruta del nodo A al nodo F.

Como podemos observar esta ruta sigue el mismo camino inicial que los vistos en las figuras anteriores, partiendo del nodo A, pasando por el B y de ahí al nodo D, por lo que se puede decir que la mejor forma de llegar a los nodos extremos (E y F) debemos pasar si o si, por esta ruta si queremos obtener el camino con menor peso. Otro detalle importante es que esta ruta es de acuerdo a las distancias declaradas para cada nodo, si estas distancias son cambiadas, es claro que las rutas también lo serán.

Seguidamente se realizaron diferentes pruebas del desempeño energético del algoritmo de Dijkstra en la launchpad, el algoritmo fue puesto a prueba en simulación con 4, 8, 6, 10 y 16 nodos. A continuación se muestran los resultados obtenidos de la figura 4.21 a la figura 4.25. Se comenta que los primeros dos escalones de las gráficas corresponden a la inicialización de la tarjeta launchpad; para el caso de la figura 4.21 el tiempo de procesamiento del algoritmo ocurre entre los 7 y 7.8 segundos.

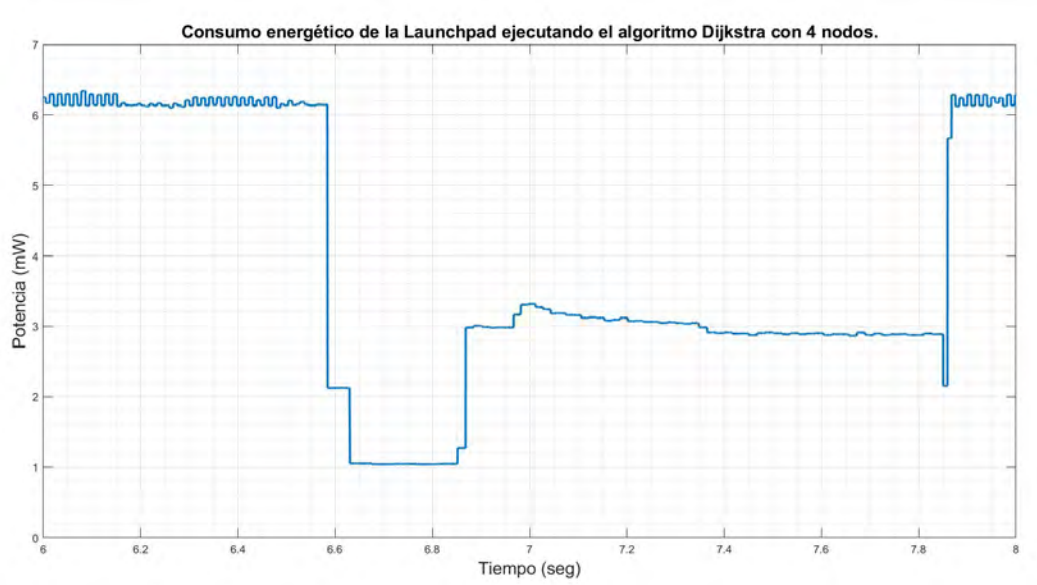

Figura 4.21: Consumo energético de la launchpad ejecutando Dijkstra con 4 nodos.

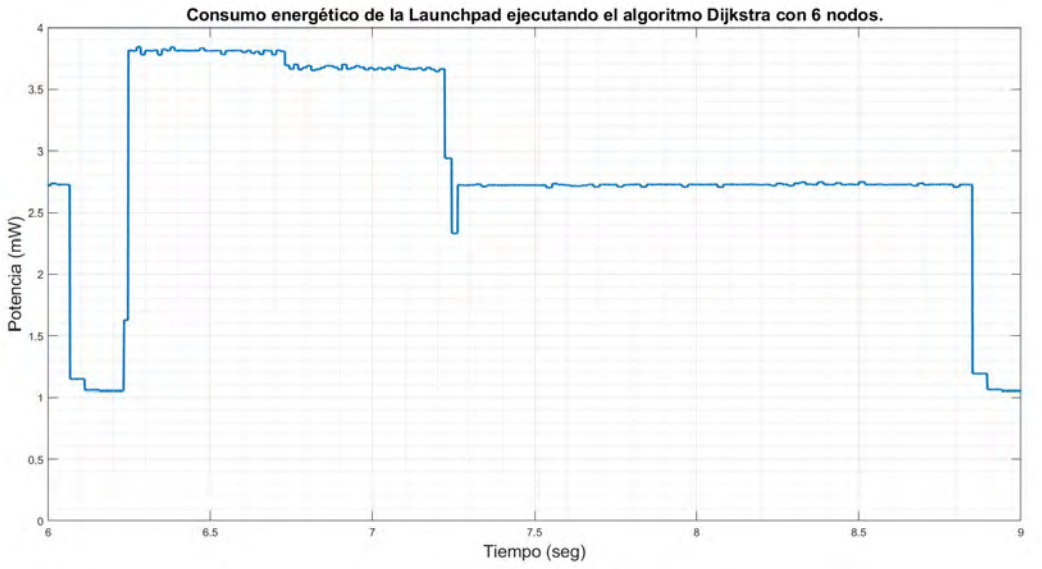

Figura 4.22: Consumo energético de la launchpad ejecutando Dijkstra con 6 nodos.

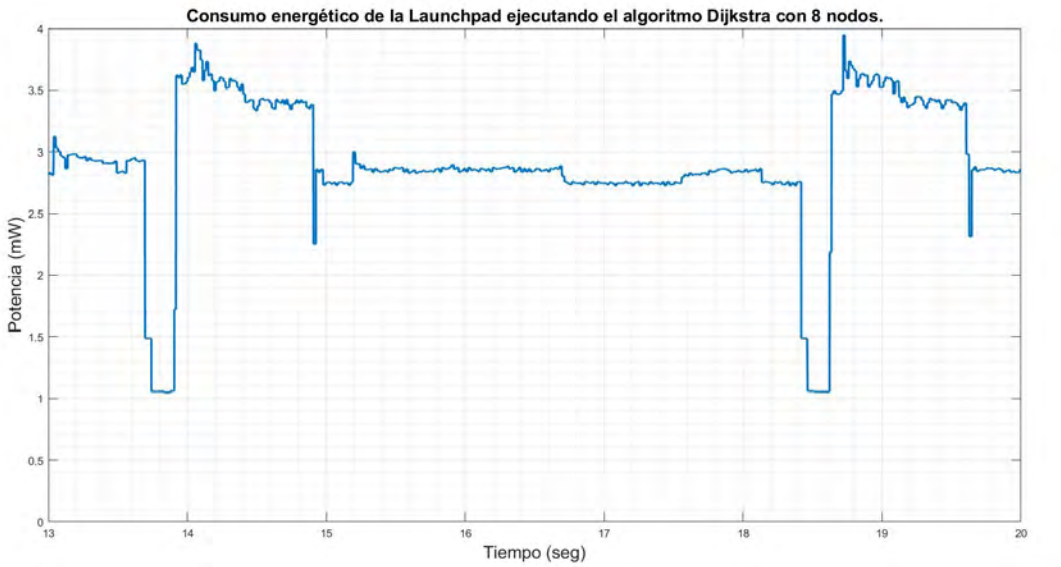

Figura 4.23: Consumo energético de la launchpad ejecutando Dijkstra con 8 nodos.

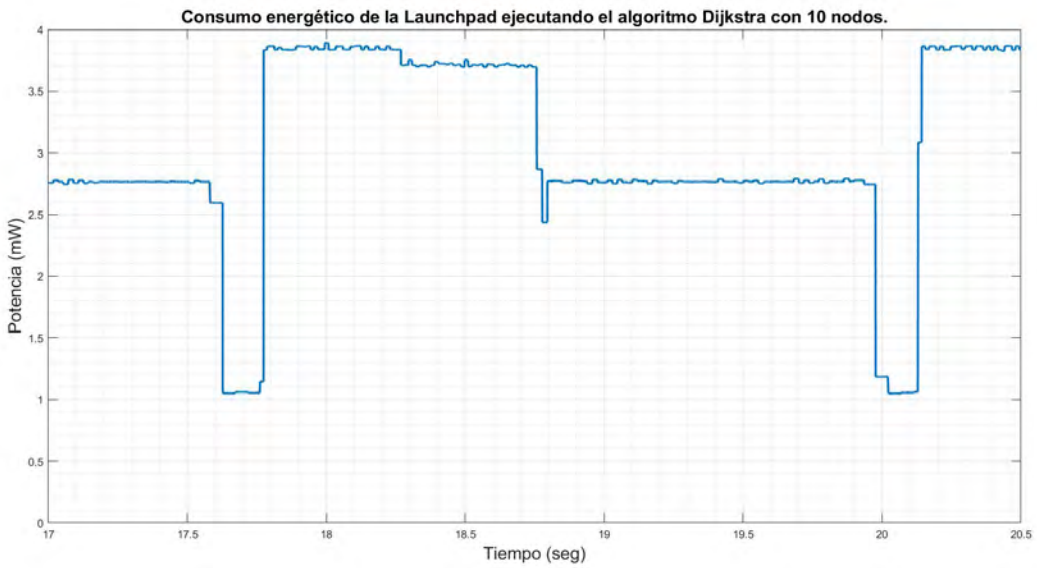

Figura 4.24: Consumo energético de la launchpad ejecutando Dijkstra con 10 nodos.

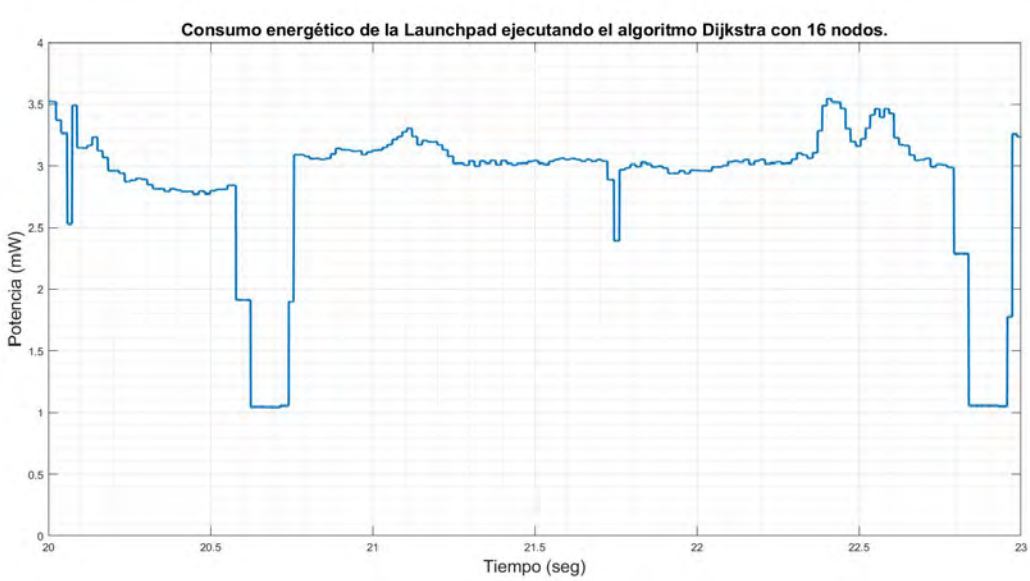

Figura 4.25: Consumo energético de la launchpad ejecutando Dijkstra con 16 nodos.

Estrictamente no existe un consumo abrupto de energía; sin embargo, a medida de que el número de los nodos aumenta, se puede observar un incremento en el tiempo de procesamiento. A pesar de ello, al ser Dijktra un algoritmo de baja complejidad, consumo energético se ve ligeramente afectado debido al tiempo de ejecución. Otro punto importante a mencionar, que la solución del algoritmo también dependerá de los pesos y destino a alcanzar. Los códigos desarrollados correspondiente al algoritmo Dijkstra pueden encontrarse en la **Sección de Anexos** al final del documento.

4.3. Algoritmo AODV

El protocolo AODV (Ad hoc On-demand Distance Vector) es un protocolo reactivo, es decir, crea las rutas de comunicación entre nodos únicamente cuando éstas son requeridas.

AODV emplea mensajes de broadcast llamados “Hello” los cuales manda cada nodo a sus vecinos para indicar la ruta y la existencia de dicho nodo en la red. Si un nodo no recibe una respuesta a su mensaje “Hello” quiere decir que el enlace esta roto o no hay comunicación con un nodo.

Cuando existe información que se tiene que enviar se hace un broadcast con un mensaje Route Request (RREQ) para el destino. Para cada nodo intermedio que reciba un RREQ creará una ruta de origen del paquete. Si el nodo receptor no recibe este RREQ antes, no es el destino y no conoce una ruta actual al destino, por lo que retransmite un RREQ. Si el nodo receptor es el destino o tiene una ruta especifica al destino genera un paquete Route Replay (RREP) [32].

Los mensajes de Route Replay (RREP) son de tipo unicast; es decir, van de un punto a otro de forma dirigida pasando por los nodos de comunicación entre ellos. Conforme se vaya transmitiendo el RREP, cada nodo intermedio irá creando una ruta para llegar al nodo destino, creando así la ruta de alcance al nodo destino. Si el nodo origen recibe varios RREP, indicando diferentes rutas a un mismo nodo, el nodo origen deberá elegir al nodo destino mediante la ruta mas corta con el menor número de saltos entre ambos nodos.

Mientras la información viaja entre los nodos de origen y destino, los nodos intermedios actualizan sus temporizadores para mantener la comunicación en la ruta creada. Si en algún momento la ruta deja de usarse por mucho tiempo, el nodo no puede garantizar que la ruta sea confiable, por lo que procede a eliminarla de su tabla de enrutamiento y realiza el proceso de aprendizaje de la ruta nuevamente.

En caso de que la comunicación se pierda mientras existe el flujo de información, se envía un mensaje Route Error (RERR) el cual es enviado a los nodos intermedios para que invaliden cualquier ruta inalcanzable. Si el nodo de origen es quien recibe un RERR, éste procederá a eliminar la ruta inalcanzable de su tabla y reiniciará el proceso de aprendizaje hacía el destino.

En la figura 4.26 se puede ejemplificar el proceso de rutas del protocolo AODV mediante los diferentes mensajes que utiliza para validarlas. Los códigos desarrollados correspondiente al algoritmo AODV pueden encontrarse en la **Sección de Anexos** al final del documento.

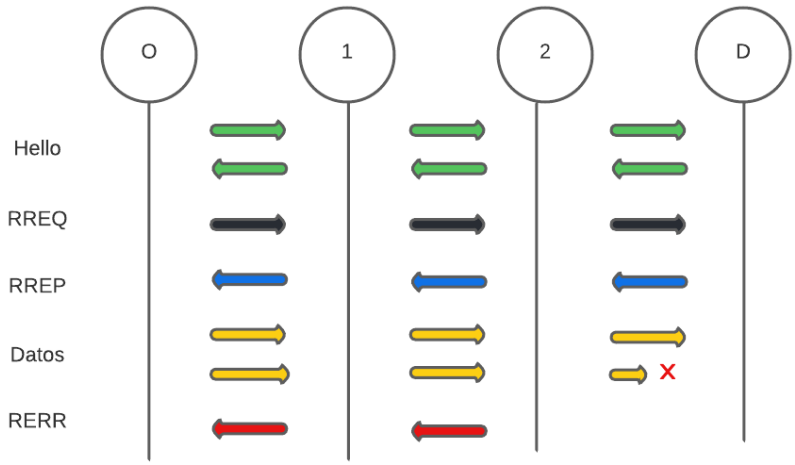

Figura 4.26: Diagrama de comunicación del protocolo AODV de un nodo origen a un nodo destino con dos nodos intermedios.

Con relación a este algoritmo se trabajó con una simulación en Matlab en donde se realizaron pruebas de funcionamiento del algoritmo. En este sentido se creo de forma aleatoria una infraestructura de red con diversos nodos, el reto era observar como se encontraba la ruta mas corta de un nodo a otro. En la figura 4.27 podemos observar la ruta que fue creada.

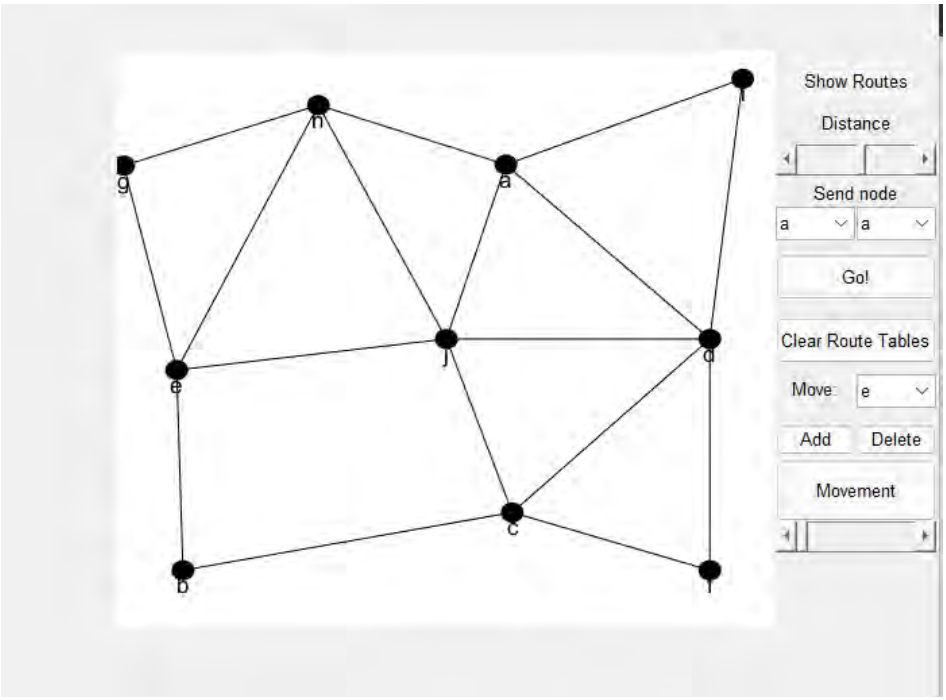

Figura 4.27: Red de nodos para prueba con el algoritmo AODV.

Al ejecutar el algoritmo se solicitó que se trace una ruta del nodo A al nodo F para obtener la ruta mas corta tal como lo muestra la figura 4.28.

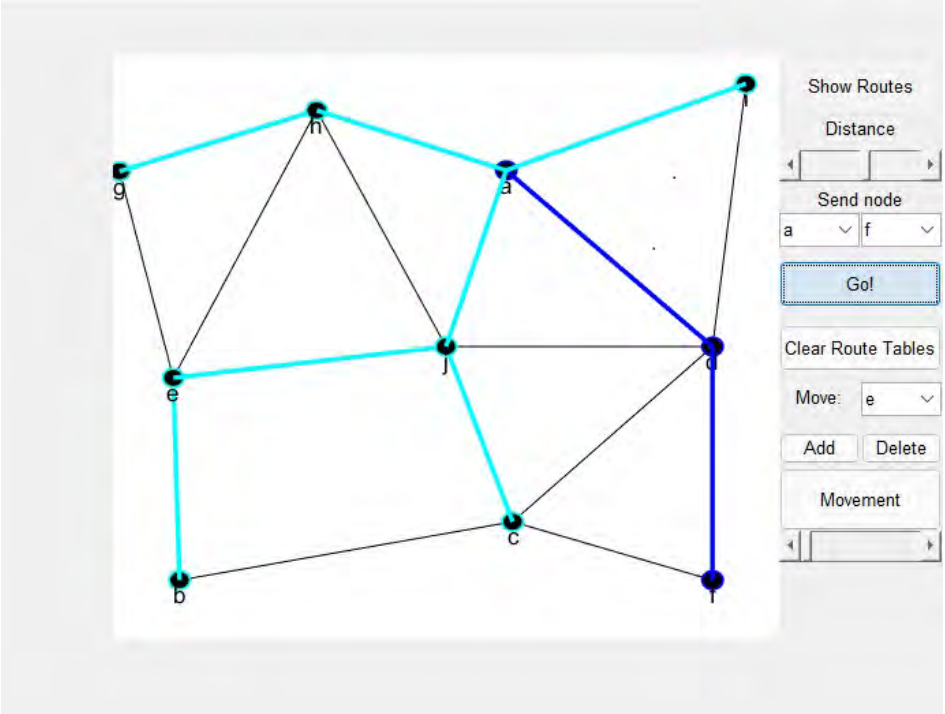

Figura 4.28: Ruta mas corta del nodo A al nodo F.

Como se puede observar la ruta trazada se obtuvo en el momento de ejecutar el algoritmo y esta ruta será usada siempre que el nodo A quiera alcanzar el nodo F. Se comenta que no será necesario calcularla de nuevo, siempre y cuando se desee hacer una actualización de las rutas, o que alguno de los nodos intermedios pierda comunicación hacia el nodo F.

Al ejecutar el algoritmo AODV se obtiene una tabla de enrutamiento la cual nos muestra el proceso de aprendizaje de la ruta entre los nodos A al F.

SeqNum: 1 **Node a**

	dest	nextHop	hopCnt	seqNum	lifeTime
1	f	d	2	1	2

Figura 4.29: Tabla de enrutamiento del algoritmo AODV en el nodo A.

SeqNum: 1 **Node b**

	dest	nextHop	hopCnt	seqNum	lifeTime
1	a	e	3	1	1

Figura 4.30: Tabla de enrutamiento del algoritmo AODV en el nodo B.

SeqNum: 1 **Node c**

	dest	nextHop	hopCnt	seqNum	lifeTime
1	a	j	2	1	1

Figura 4.31: Tabla de enrutamiento del algoritmo AODV en el nodo C.

SeqNum: 1 **Node d**

	dest	nextHop	hopCnt	seqNum	lifeTime
1	a	a	1	1	1
2	f	f	1	1	2

Figura 4.32: Tabla de enrutamiento del algoritmo AODV en el nodo D.

SeqNum: 1 **Node e**

	dest	nextHop	hopCnt	seqNum	lifeTime
1	a	j	2	1	1

Figura 4.33: Tabla de enrutamiento del algoritmo AODV en el nodo E.

SeqNum: 1 **Node f**

	dest	nextHop	hopCnt	seqNum	lifeTime
1	a	d	2	1	1

Figura 4.34: Tabla de enrutamiento del algoritmo AODV en el nodo F.

SeqNum: 1 **Node g**

	dest	nextHop	hopCnt	seqNum	lifeTime
1	a	h	2	1	1

Figura 4.35: Tabla de enrutamiento del algoritmo AODV en el nodo G.

SeqNum: 1 **Node h**

	dest	nextHop	hopCnt	seqNum	lifeTime
1	a	a	1	1	1

Figura 4.36: Tabla de enrutamiento del algoritmo AODV en el nodo H.

SeqNum: 1 **Node i**

	dest	nextHop	hopCnt	seqNum	lifeTime
1	a	a	1	1	1

Figura 4.37: Tabla de enrutamiento del algoritmo AODV en el nodo I.

En las figuras 4.29 a 4.37 podemos apreciar cada una de las tablas de enrutamiento por nodo para que el nodo A aprenda la ruta al nodo F empleando el algoritmo AODV.

Debido a limitaciones técnicas, financieras y de tiempo, las pruebas con el algoritmo AODV no se lograron realizar en un entorno de laboratorio con equipos que puedan ejecutarlo en tiempo real, de tal forma que se pueda obtener un análisis del consumo energético de los nodos que lo ejecutan. Sin embargo, se puede pensar que AODV es una alternativa viable para ser utilizado en aplicaciones IoT industria 4.0 debido a su baja complejidad algorítmica.

Capítulo 5

Conclusiones

Cada día la consideración de conectividad de cosas a través Internet va en aumento. La industria no se queda atrás, es por ello que Industria 4.0 buscar que equipo industrial posea conectividad a Internet a través de nodos sensores de red inalámbricos. Por ejemplo, motores, calderas, arrancadores, entre otros, envíen información de sus estados a unidades remotas. En este sentido, cada uno de los dispositivos podrá hacerlo a través de sus dispositivos de transmisión de datos o nodos sensores de red incorporados al equipo o maquinaria industrial.

Debido a que se requiere que los nodos sensores puedan enviar información de forma continua hacia un nodo destino, es necesario que dichos nodos puedan hacerlo acorde a un nivel de desempeño deseado. Como ha sido planteado en este documento, la ubicación de los nodos sensores dentro de un ambiente de propagación para la Industria 4.0 no necesariamente siempre es el adecuado. Lo anterior se debe primordialmente a que los nodos pueden estar operando con fuentes de ruido adicionales generados por equipo industrial (motores, equipo de soldadura, etc). Además, es común que el área de trabajo de la industria cambie de forma recurrente adicionando barreras donde antes no habían (muros, paredes, otros equipos, entre otros). Lo anterior ocasionará que el desempeño del sistema de nodos sensores desplegado en la industria se vea afectado. En este sentido, es necesario evaluar y proponer el uso de nuevos esquemas de ruteo que ayuden a mitigar tal problemática.

Si bien, existen protocolos propuestos por una comunidad, tal es el caso de LoRa-

WAN, existen limitaciones que pueden provocar que el rendimiento de los sensores en la red no sea el esperado debido a que no fueron diseñados para operar bajo ese entorno de propagación. Es por ello que la comunidad científica busca realizar propuestas de protocolos IoT que hayan sido conceptualizados para operar en condiciones reales industriales.

Hoy en día, la mayoría de las propuestas se encuentran en fases tempranas de desarrollo; incluso, se tienen propuestas que ya se encuentran en etapas de simulación; sin embargo, el objetivo de buscar esa eficiencia es latente en la industria hoy en día. Ejemplos de propuestas alternativas son las presentadas en este trabajo, algoritmo Dijsktra y AODV, los cuales son algoritmos clásicos pero que debido a su baja complejidad algorítmica hoy en día pueden ser conceptualizados para ser utilizados en aplicaciones IoT en general.

Tanto Dijsktra como AODV buscan encontrar la ruta mas corta hacía un nodo destino empleando consideraciones asociadas a métricas de interés como RSSI, latencia, SNR, estado de las baterias del nodo, entre otros. A diferencia de LoRAWAN, estos algoritmos son descentralizados, es decir, que no dependen de un Gateway para poder comunicarse entre nodos.

Las propuestas presentadas son llevadas a un nivel de simulación en donde se busca demostrar que dichos algoritmos pueden ser alternativas viables en el ambiente industrial (caso algoritmo AODV), así mismo, es claro que lo presentado es solo el primer paso para la realización de un estudio más extenso a futuro.

Bibliografía

[1] L. D. Xu, “Internet of Things in industries: A survey,” *IEEE Transactions on industrial informactis*, vol. 10, no. 4, 2014.

[2] E. Sisinni, “Industrial Internet of Things opportunities and directions,” *IEEE Transactions on industrial informactis*, vol. 14, no. 11, 2018.

[3] R. Sánchez-Iborra, J. Sánchez-Gómez, J. Vallesta-Viñas, M. Dolores-Cano, and S. Antonio, “Performance evaluation of LoRa considering scenario conditions.,” *Sensors*, 2018.

[4] W. Wang, “Comparative analysis of channel models for industrial IoT Wireless communication.,” *IEEE Access*, 2019.

[5] A. Carlsson, I. Kuzminykh, R. Fraksson, and A. Liljegren, “Mesuring a LoRa network: Performance, possibilities and limi-tations.,” *Springer Nature Switzerland AG 2018*, 2018.

[6] R. Islam, W. Rahman, R. Rubait, M. H. M. Reza, and M. M. Rahman, “LoRa and server-basedhome automation using internet of things (IoT),” *Journal of King Saud University-Computer and Information Sciences.*, 2020.

[7] E. B., “LoRa documentation.” https://lora.readthedocs.io/en/latest/, 2018. Accessed: Spring 2021.

[8] J. Dias and A. Grilo, “LoRaWAN multi-hop uplink extension,” *Procedia Computer Science*, vol. 130, pp. 424, 431, 2018.

[9] D. L. Mai and M. K. Kim, “Multi-hop LoRa network protocol with minimized latency,” *Energies*, 2020.

[10] M. Bor, J. Vidler, and U. Roedig, “LoRa for the Internet of Things,” *International Conference on Embedded Wireless Systems and Networks*, 2016.

[11] C.-T. Duong and M. K. Kim, “Reliable multi-hop linear network based on LoRa,” *International Journal of Control and Automation*, vol. 11, no. 4, 2018.

[12] R. Jederman, M. Borysov, N. Hartenbusch, S. Jaeger, M. Sellwing, and W. Lang, “Testing LoRa for food applications - example applications for aiflow mesasuraments inside cooled warehouse with apples,” *ELSEVIER*, 2018.

[13] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, “A comparative study of LPWAN techonologies for large-scale IoT deployment,” *The Korean Institute of Communications and Information Sciencies.*, 2018.

[14] J. Peña Queralta, T. Gia N., Z. Zou, H. Tenhunen, and T. Westerlund, “Comparative study of LPWAN technologies on unlicensed band for m2m communication in the iot: beyond lora and lorawan,” *ELSEVIER*, 2018.

[15] R. Sharan Sinha, Y. Wei, and S.-H. Hwang, “A survey: on LPWAN technology: LoRa and NB-IoT,” *The Korean Institute of Communicationsand Information Sciencies.*, 2018.

[16] M. Casar, T. Pawelke, and T. Gabriel, “A survey on Bluetooth Low Energy security and privacy,” *Computer Networks*, vol. 205, p. 108712, 2022.

[17] D. Hortelano, T. Olivares, and M. C. Ruiz, “Reducing the energy consumption of the friendship mechanism in bluetooth mesh,” *Computer Networks*, vol. 195, p. 108172, 2021.

[18] N. Todtenberg and K. Rolf, “A survey on bluetooth multi-hop networks,” *Ad Hoc Networks*, vol. 93, p. 101922, 2019.

[19] P. Ruckebusch, S. Giannoulis, I. Moerman, J. Hoebeke, and E. De Poorter, “Modelling the energy consumption for over-the-air software updates in LPWAN networks: SigFox, LoRa and IEEE 802.15.4g,” *Internet of Things*, vol. 3-4, pp. 104–119, 2018.

[20] I. Smajla, D. Karasalihović Sedlar, L. Jukić, and N. Vištica, “Cost-effectiveness of installing modules for remote reading of natural gas consumption based on a pilot project,” *Elsevier*, 2022.

[21] L. Alazzawi and A. Elkateeb, “Performance evaluation of the WSN routin protocols scalability,” *Journal of Computer Systems, Networks, and Communications*, 2008.

[22] S. K. Singh, M. P. Singh, and D. K. Singh, “Routing protocols in Wireless sensor networks - a survey,” *International Journal of Computer Science and Engineering Survey*, vol. 1, no. 2, 2010.

[23] E. B., “LoRa documentation.” lora.readthedocs.io/en/latest/, 2018.

[24] “Energía.” https://energia.nu. Accessed: Summer 2021.

[25] “The things network.” https://www.thethingsnetwork.org. Accessed: Summer 2021.

[26] S. Reza Nayabi, N. Osati Eraghi, and J. Akbari Torkestani, “WSN routing protocol using multiobjective greedy approach,” *Wiley*, 2021.

[27] A. Vinitha, R. M.S.S., and Dhirajsunehra, “Secure and energy aware multi-hop routing protocol in WSN using taylor-based hybrid optimization algorithm,” *Journal of King Saud University - Computer and Information Sciences*, 2019.

[28] R. Kaur and K. Singh, “An efficient multipath dynamic routing protocol for mobile wsns,” *Procedia Computer Science*, vol. 46, 2015.

[29] J. Rodríguez Cotrim and J. H. Kleinschmidt, “Review: LoRaWAN mesh networks: A review and classification of multihop communication,” *Sensors*, 2020.

[30] D. A. Mejias Rojas, “Diseño de protocolo de redes mesh basado en LoRa,” 2021.

[31] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., 2009.

[32] I. Chakeres and E. Belding, “AODV routing protocol implementation design.,” *Proceedings of the 24th International Conference on Distributed Computingsystems Workshops*, pp. 698–703, 01 2004.

Capítulo 6

Anexos

Anexo A: Código de prueba del algoritmo Dijkstra.

```
    %% Ejemplo

clc, clear all;
       % A   B    C    D   E    F
map = [  0    3   11  10  Inf  Inf
         3    0   Inf  4   7   Inf
         11  Inf   0   2  Inf   9
         10   4    2   0   1    4
         Inf  7   Inf  1   0    9
         Inf  Inf  9   4   9    0]

Dijsktra1(map,1,6)
distances = Dijsktra2(map,1)
```

Dijkstra 1

```
    function Dijsktra1(A,sb,db)
% A (entrada) representa la matriz de adyacencia, sb, la etiqueta del
punto de inicio, db, la etiqueta del punto final
 % B (salida) representa la matriz de tiempo que requiere menos tiempo,
```

```
 dist - el tiempo que requiere menos tiempo, mypath - la ruta más corta
m=length(A);
for i=2:m
    for j=1:(i-1)
      A(i,j)=A(j,i);
    end
end
a=A;
for k=1:(m-1)
    b=[1:(k-1),(k+1):m];
    kk=length(b);
    a_id=k;
    b1=[(k+1):m];
    kk1=length(b1);
    while kk>0
      for j=1:kk1
          te=A(k,a_id)+A(a_id,b1(j));
          if te<A(k,b1(j))
              A(k,b1(j))=te;
          end
      end
      miid=1;
      for j=2:kk
        if A(k,b(j))<A(k,b(miid))
            miid=j;
        end
      end
      a_id=b(miid);
      b=[b(1:(miid-1)),b((miid+1:kk))];
      kk=length(b);
      if a_id>k
      miid1=find(b1==a_id);
      b1=[b1(1:(miid1-1)),b1((miid1+1):kk1)];
```

```
      kk1=length(b1);
      end
    end
    for j=(k+1):m
        A(j,k)=A(k,j);
    end
end

m=size(a,1);
path=zeros(m);

for k=1:m
    for i=1:m
        for j=1:m
            if a(i,j)>a(i,k)+a(k,j)
                a(i,j)=a(i,k)+a(k,j);
                path(i,j)=k;
            end
        end
    end
end

dist=a(sb,db);

parent = path (sb,:);% el vértice precursor de cada vértice en la ruta más
corta desde el punto de inicio sb hasta el punto final db

parent (parent == 0) = sb; %El componente en path es 0, lo que indica que
el precursor del vértice es el punto de partida

mypath=db; t=db;

while t~=sb
```

```
        p=parent(t); mypath=[p,mypath]
        t=p;
end

 fprintf (’matriz de distancia más corta del punto es:’) ;

B=A,dist,mypath
```

Dijkstra2

```
    function distances = Dijsktra2(map, startingpoint)

N = length(map);

distances(1:N) = Inf;

visited(1:N) = 0;

distances(startingpoint) = 0;

while sum(visited) < N
    candidates(1:N) = Inf;
    for index = 1:N
        if visited(index) == 0
            candidates(index) = distances(index);
        end
    end
    [currentDistance, currentPoint] = min(candidates);

    for index = 1:N
        newDistance = currentDistance + map(currentPoint, index);
        if newDistance < distances(index)
            distances(index) = newDistance;
```

```
        end
    end
    visited(currentPoint) = 1;
end
```

Código Dijkstra para 4 nodos.

```
// A C++ program for Dijkstra’s single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

// Number of vertices in the graph
    using namespace std;

#define V 4
#include <limits.h>
   int graph[V][V] = { { 0, 2, 5, 10 },
                       { 2, 0, 8, 0 },
                       { 5, 8, 0, 3 },
                       { 10, 0, 3, 0 } };
// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{

    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed distance array
void printSolution(int dist[])
{
    Serial.println("Vertice");
```

```
    for (int i = 0; i < V; i++)
        Serial.println(i);

    Serial.println("Distancia al vértice");
    for (int i = 0; i < V; i++)
        Serial.println(dist[i]);
}

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array.  dist[i] will hold the shortest
    // distance from src to i

    bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;
```

```
        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }
    // print the constructed distance array
    printSolution(dist);
}

void setup() {
  Serial.begin(115200);
  dijkstra(graph, 0);
  // put your setup code here, to run once:
}

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

Código Dijkstra para 6 nodos.

```
    // A C++ program for Dijkstra’s single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

// Number of vertices in the graph
using namespace std;

#define V 6
#include <limits.h>
   int graph[V][V] = { { 0, 3, 11, 10, 0, 0 },
                        { 3, 0, 0, 4, 7, 0 },
                       { 11, 0, 0, 2, 0, 9 },
                       { 10, 4, 2, 0, 1, 4 },
                       { 0, 7, 0, 1, 0, 9 },
                       { 0, 0, 9, 4, 9, 0 } };
// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{

    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed distance array
void printSolution(int dist[])
```

```
{
    Serial.println("Vertice");
    for (int i = 0; i < V; i++)
        Serial.println(i);

    Serial.println("Distancia al vértice");
    for (int i = 0; i < V; i++)
        Serial.println(dist[i]);
}

// Function that implements Dijkstra’s single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array.  dist[i] will hold the shortest
    // distance from src to i

    bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);
```

```
        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist);
}

void setup() {
  Serial.begin(115200);
  dijkstra(graph, 0);
  // put your setup code here, to run once:
}

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

}
```

Código Dijkstra para 8 nodos.

```
    // A C++ program for Dijkstra’s single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

// Number of vertices in the graph
using namespace std;

#define V 8
#include <limits.h>
    int graph[V][V] = { { 0, 7, 0, 0, 0, 0, 0, 2 },
                        { 7, 0, 11, 0, 0, 0, 0, 0 },
                        { 0, 11, 0, 6, 0, 0, 5, 0 },
                        { 0, 0, 6, 0, 7, 0, 0, 0 },
                        { 0, 0, 0, 7, 0, 8, 0, 0 },
                        { 0, 0, 0, 0, 8, 0, 15, 12 },
                        { 0, 0, 5, 0, 0, 15, 0, 9 },
                        { 2, 0, 0, 0, 0, 12, 9, 0 } };
// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{

    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}
```

```
// A utility function to print the constructed distance array
void printSolution(int dist[])
{
    Serial.println("Vertice");
    for (int i = 0; i < V; i++)
        Serial.println(i);

    Serial.println("Distancia al vértice");
    for (int i = 0; i < V; i++)
        Serial.println(dist[i]);
}

// Function that implements Dijkstra’s single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array.  dist[i] will hold the shortest
    // distance from src to i

    bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
```

```
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist);
}

void setup() {
  Serial.begin(115200);
  dijkstra(graph, 0);
  // put your setup code here, to run once:
}

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

}
```

Código Dijkstra para 10 nodos.

```
    // A C++ program for Dijkstra’s single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

// Number of vertices in the graph
using namespace std;

#define V 10
#include <limits.h>
    int graph[V][V] = { { 0, 3, 11, 5, 0, 0, 0, 0, 0, 0 },
                        { 3, 0, 0, 0, 7, 0, 0, 0, 0, 0 },
                        { 11, 0, 0, 0, 0, 5, 0, 0, 0, 0 },
                        { 5, 0, 0, 0, 0, 3, 0, 0, 0, 0 },
                        { 0, 7, 0, 0, 0, 0, 11, 0, 0, 0 },
                        { 0, 0, 5, 3, 0, 0, 0, 3, 8, 0 },
                        { 0, 0, 0, 0, 11, 0, 0, 9, 0, 11 },
                        { 0, 0, 0, 0, 0, 3, 9, 0, 11, 7 },
                        { 0, 0, 0, 0, 0, 8, 0, 11, 0, 12 },
                        { 0, 0, 0, 0, 0, 0, 11, 7, 12, 0 } };
// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{

    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
```

```
}

// A utility function to print the constructed distance array
void printSolution(int dist[])
{
    Serial.println("Vertice");
    for (int i = 0; i < V; i++)
        Serial.println(i);

    Serial.println("Distancia al vértice");
    for (int i = 0; i < V; i++)
        Serial.println(dist[i]);
}

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array.  dist[i] will hold the shortest
    // distance from src to i

    bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
```

```
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist);
}

void setup() {
  Serial.begin(115200);
  dijkstra(graph, 0);
  // put your setup code here, to run once:
}

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

}
```

Código Dijkstra para 16 nodos.

```
    // A C++ program for Dijkstra’s single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph

// Number of vertices in the graph
using namespace std;

#define V 16
#include <limits.h>
    int graph[V][V] = { { 0, 5, 7, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                        { 5, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                        { 7, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0 },
                        { 2, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0 },
                        { 0, 0, 0, 0, 0, 9, 0, 5, 0, 0, 0, 0, 9, 0, 0, 0 },
                        { 0, 3, 0, 0, 9, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                        { 0, 0, 0, 0, 0, 15, 0, 0, 0, 0, 0, 5, 0, 0, 0, 5 },
                        { 0, 0, 0, 7, 5, 0, 0, 0, 7, 9, 0, 0, 0, 0, 0, 0 },
                        { 0, 0, 5, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 0, 0, 9, 0, 0, 17, 4, 0, 0, 8, 0 },
                        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 5, 7, 0 },
                        { 0, 0, 0, 0, 0, 0, 5, 0, 0, 4, 0, 0, 2, 0, 0, 0 },
                        { 0, 0, 0, 0, 9, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0 },
                        { 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 5, 0, 0, 0, 0, 0 },
                        { 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 7, 0, 0, 0, 0, 6 },
                        { 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 6, 0 } };
// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{

    // Initialize min value
    int min = INT_MAX, min_index;
```

```
    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed distance array
void printSolution(int dist[])
{
    Serial.println("Vertice");
    for (int i = 0; i < V; i++)
        Serial.println(i);

    Serial.println("Distancia al vértice");
    for (int i = 0; i < V; i++)
        Serial.println(dist[i]);
}

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array.  dist[i] will hold the shortest
    // distance from src to i

    bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
```

```
    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist);
}

void setup() {
  Serial.begin(115200);
  dijkstra(graph, 0);
  // put your setup code here, to run once:
}
```

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

}
```

Anexo B: Código main de AODV en MatLab

```
clear global
clear
close all
delete(timerfind)

% Seed random number generator
rng(12345)

% Setup nodes
global nodes distance range routeLifetime;
numNodes = 9;
routeLifetime = 15;
distance = 5.5;
range = 10;
nodes = node();
% for i = 1:numNodes
%     nodes(i) = node(char(i-1+'a'),rand * range, rand * range);
% end
nodes(1) = node("a",5,5);
nodes(2) = node("b",1,1);
nodes(3) = node("c",6,2);
nodes(4) = node("d",9,5);
nodes(5) = node("e",3,6);
nodes(6) = node("f",9,1);
nodes(7) = node("g",0.1,8);
nodes(8) = node("h",1,8.9);
nodes(9) = node("i",9.5,9.5);

% Setup figures
global graphFig showRoutesBtn tableFig;
graphFig = initGraphView();
```

```
updateGraphView()
tableFig = initTableView();
updateTableData()

% Initialize remaining components
initMove()
calcConnections(distance,showRoutesBtn.Value);
for node = 1:numel(nodes)
    nodes(node).seqNum = 1;
end

% Show graph view on top
figure(graphFig)
```