Universidad Nacional Autónoma de México

Facultad de Ingeniería

Implementación de un algoritmo para la detección del ataque wormhole a partir de características topológicas

T E S I S

que para obtener el título de
Ingeniera en Computación

P R E S E N T A

Alejandra Abigail Hernández Vanegas

y para obtener el título de
Ingeniero en Telecomunicaciones

P R E S E N T A N

Brandon Jaimes Amador
Irving Jose Luis Hernández Márquez

Director de Tesis:

Dr. Luis Francisco García Jiménez

Ciudad Universitaria,Cd.Mx.,2023

Agradecimientos

Me gustaría expresar mi agradecimiento a la Universidad Nacional Autónoma de México y la Facultad de Ingeniería por todo lo aprendido durante el curso de mi licenciatura y a DGAPA-PAPIIT IA102822 por el apoyo durante el desarrollo de este proyecto.
A nuestro asesor, el Dr. Luis Francisco García Jiménez por compartirnos sus conocimientos y apoyarnos en cada etapa de este trabajo.
A mis compañeros de equipo Brandon e Irving, con los que tuve la oportunidad de compartir esta experiencia, muchas gracias por su compromiso, empeño y dedicación para la realización de esta Tesis.
A mis padres y hermanas que me han amado y apoyado siempre a pesar de la distancia; sepan que los amo y sin ustedes jamás hubiera llegado tan lejos.
A mi abuelita por brindarme su casa y su amor.
A Alfred por creer siempre en mí, por motivarme y acompañarme a lo largo de estos años.
A Mari, Fabio y a todos mis amigos que siempre han estado para compartir nuestros buenos y malos momentos.

Alejandra Abigail Hernández Vanegas

A quien me dio la vida, su amor, su confianza, me apoyo durante este viaje y lo sigue haciéndolo, a mi madre.
A Christopher porque siempre hay algo que celebrar, A Silvia por su apoyo y A Hatziry que siempre tiene un abrazo para dar.
A todos mis amigos. A Armando e Irving, por todo lo compartido, proyectos y experiencias todo este tiempo. A Aurora que nunca me dejo solo y los tiempos de risas.
A Alejandra e Irving por formar parte de este trabajo, por su esfuerzo y dedicación para hacerlo posible.
A mi asesor y profesor el Dr. Francisco que con su conocimiento y asesoría hizo posible este trabajo.
Gracias a la Universidad Nacional Autónoma de México y la Facultad de Ingeniería por todo lo brindado, sus instituciones por haberme permitido formarme en ellas, gracias a cada docente quienes con su apoyo y enseñanzas constituyen la base de mi vida profesional y finalmente el apoyo del proyecto DGAPA-PAPIIT IA102822.

Brandon Jaimes Amador

Quiero agradecer en primera instancia a la Universidad Nacional Autónoma de México, a la Facultad de Ingeniería y a DGAPA-PAPIIT IA102822 por el apoyo y todos los recursos otorgados por estas instituciones para hacer posible este trabajo de investigación.
Agradezco a mis compañeros y profesores que me acompañaron en este recorrido y en especial a el Dr. Luis Francisco García Jiménez por su guía, su constancia y su apoyo durante este proyecto.
A Brandon y Alejandra, por su empeño, su tiempo, su colaboración y la ambición para concluir esta Tesis.
A mis compañeros Armando y Brandon que siempre estuvieron ahí para trabajar en cualquier reto que se nos presentara.
A Iridian, porque me ayudo a obtener una diferente perspectiva de todas las cosas.
A mis hermanos, que aunque son menores, pude aprender muchas cosas de ellos.
Y finalmente a mi madre, gracias a ella pude llegar tan lejos y jamás se rindió con su apoyo, su comprensión y su confianza, que constantemente, siempre me brindo.

Irving Jose Luis Hernández Márquez

Índice general

Índice de figuras

Índice de tablas

Índice de acrónimos

ABR: Associativity Based Routing

AoA: Angle of Arrival

AODV: Ad hoc On-Demand Distance Vector

B.A.T.M.A.N.: Better Approach To Mobile Adhoc Networking

DoS: Denial of Service

DSR: Dynamic Source Routing

DV-Hop: Distance Vector-Hop

ETT: Expected Transmission Time

ETX: Expected Transmission Count

GPS: Global Positioning System

IoT: Internet of Things

MAC: Medium Access Control

MWNL: Multi-Wormhole-Node-Link

OOB: Out-of-band

RIP: Routing Information Protocol

RSSI: Received Signal Strength Indicator

TDoA: Time Difference of Arrival

ToA: Time of Arrival

TTL: Time to Live

VANET: Vehicular Ad-Hoc Network

WCETT: Weighted Cumulative Expected Transmission Time

WFDV: Wormhole-Free DVHop

WRL: Wormhole Resistent Localization

Índice de términos

Blackhole: Agujero negro.
: Se produce cuando un intermediario captura y programa un conjunto de nodos de la red para que bloqueen o eliminen los paquetes y generen mensajes falsos en lugar de reenviar la información correcta en la red de sensores inalámbricos.

Breakpoint: Punto de interrupción.
: Es un nodo en donde se detiene la construcción de árbol de expansión (spanning tree).

Denial of Service: Denegación de servicio.
: Este ataque se presenta cuando se interrumpe el tráfico de datos, denegando el servicio desde (o hacia) el origen o destino, lo que afecta total o parcialmente a la red.

Distance Vector-Hop: Salto vectorial de distancia.
: Protocolo de encaminamiento que utiliza la distancia o conteo de saltos, como métrica para determinar la mejor ruta de envió de información.

Eavesdropping: Escuchando a escondidas.
: Es la intercepción de información no autorizada en tiempo real.

Energy harvesting: Recolección/Cosecha de energía.
: Está relacionado con el uso, almacenamiento y gestión de la energía del ambiente, la cual es convertida principalmente en energía eléctrica para su uso en aplicaciones que requieren pequeña potencia.

Flooding: Inundación.
: Es un proceso dentro del cual se envían paquetes a todos los vecinos.

Grayhole: Agujero gris.
: Son ataques que se llevan a cabo en redes inalámbricas, dentro de los cuales un nodo malicioso logra atraer hacia sí mismo los paquetes de sus vecinos con el fin de eliminarlos.

Hook point: Punto de gancho.
: En esta tesis utilizamos el término para indicar que el nodo dentro de la red cuenta con GPS, lo cual facilitará llevar a cabo la localización de los demás en el algoritmo DV-Hop.

Hopsize:
: Es el tamaño del salto, el cual se usa como métrica para llevar a cabo la localización dentro el algoritmo DV-Hop.

Internet of Things: Internet de las cosas.
: Tecnología que describe la red de objetos físicos que llevan incorporados sensores, software y otras tecnologías con el fin de conectarse e intercambiar datos con otros dispositivos y sistemas a través de Internet.

Multicast: Multidifusión.
: Hace referecia a la entrega de paquetes de manera simultánea a un grupo de nodos destino.

Multilateración:
: Método mediante el cual un nodo desconocido obtiene la información de tres o más distancias de los nodos ancla para estimar sus coordenadas.

Overhearing: Escuchando.
: En el caso de las redes inalámbricas es un concepto que se refiere a que los nodos están esperando recibir información todo el tiempo, pero no es óptimo debido al uso desmedido de recursos como la energía.

Red Ad-Hoc:
: Tipo de red descentralizada que se caracteriza por funcionar de punto a punto, de modo que los nodos pueden comunicarse entre sí independientemente de una infraestructura fija.

Sinkhole: Sumidero/Pozo negro.
: Es un ataque dentro del cual los nodos maliciosos se encargan de enviar información inexacta para el encaminamiento, con el objetivo de engañar a los demás nodos pensando que se tiene una ruta óptima hacia el destino.

Spanning Tree: Árbol de expansión.

Sybil:
: Ocurre cuando un nodo atacante asume diferentes identidades en la red o reemplaza la identidad de uno o más nodos para atraer información.

Wormhole: Agujero de gusano.
: En este caso un nodo atacante se encarga de capturar los paquetes que pasan por él y los reenvía a otro nodo de la red con el que está colaborando con el fin de engañar a los nodos legítimos con las tablas de encaminamiento (vecinos).

Resumen

Los continuos avances tecnológicos dentro de las redes inalámbricas de sensores (Wireless Sensor Networks) han facilitado su incorporación y aplicación en múltiples entornos. Por ejemplo, en el monitorización ambiental, fortalecimiento de la domótica, aplicaciones militares, médicas, industriales y comerciales. Estos dispositivos tienen la capacidad de monitorizar una o más variables físicas, donde la información recolectada es enviada a un nodo especial llamado nodo sumidero o *sink*. Una red inalámbrica de sensores se compone de un conjunto de nodos, donde cada uno de ellos se comunica directamente con aquellos nodos que están dentro de su cobertura local. Por lo que si desean comunicarse con sensores más lejanos requieren implementar mecanismos que hagan posible la comunicación entre las diferentes áreas de la red, esto es debido a que este tipo de redes no cuentan con un nodo central que administre las comunicaciones.

Una de las mayores preocupaciones en este tipo de redes es la protección de la información, debido a que es el recurso más importante que se genera al monitorizar las variables físicas, y de ser interceptada por un agente malicioso, éste podría beneficiarse de algún modo. Actualmente, se conoce de la existencia de varios ataques dirigidos a afectar el correcto funcionamiento de las redes de sensores, entre ellos se encuentran el ataque *blackhole* y el *grayhole*, los cuales son ataques de interrupción de la información, donde un nodo malicioso se encarga de capturar el tráfico que pasa por él con la finalidad de que no llegue a su destino. Aunque existen varios ataques en este tipo de redes, uno de los ataques particularmente difícil de contrarrestar es el ataque *wormhole*, en el que dos nodos maliciosos que se encuentran a cierta distancia y conectados por un medio de alta tasa de transmisión pueden vulnerar la seguridad de la red e interceptar la información. Para ello, las transmisiones inalámbricas se graban en un lado de la conexión y se reproducen en el otro lado del ataque, creando un enlace virtual bajo el control del atacante. Esto afecta a los protocolos de localización, así como a los protocolos de encaminamiento, generando daños en la topología de la red y de los protocolos que están en capas superiores.

Actualmente existen diversas propuestas para contrarrestar el ataque *wormhole*. Algunas de ellas utilizan técnicas como la sincronización de relojes, o la utilización de *hardware* especializado para detectar el ataque *wormhole*, sin embargo, tienden a incrementar el costo de los dispositivos. Otras propuestas se basan en la técnica de *overhearing*, la cual afecta severamente el consumo de energía. También existen propuestas que están orientadas a detectar longitudes de rutas más cortas y estructuras anormales en la topología de la red. Sin embargo, estas requieren del conocimiento del vecindario a cuatro o cinco saltos, lo que incrementa el procesamiento y el consumo energético. En esta tesis, a diferencia de estas propuestas, se plantea un algoritmo para la detección del ataque *wormhole* basado en características topológicas, que solo utiliza el conocimiento de los nodos que se encuentran a un salto de distancia y del uso de algunos nodos que conocen su posición geográfica. Específicamente, en

esta tesis se implementa el algoritmo propuesto en [1] (el cual fue propuesto en un contexto teórico) sobre un ambiente real, el cual se compone por las siguientes etapas; la primera etapa localiza las posibles zonas en las que se encuentran los nodos maliciosos mediante el uso de un algoritmo distribuido llamado *Distance Vector-Hop* (DV-Hop). En la segunda etapa, se busca cuáles de estas zonas forman un grafo bipartito entre los nodos maliciosos, ya que una de las características del ataque *wormhole* es que las conexiones entre los nodos maliciosos siempre forman un grafo bipartito. Posteriormente, con las zonas localizadas, se forma un árbol de expansión (*spanning tree*) que elimina las aristas del grafo bipartito contrarrestando el ataque. Para la implementación de este algoritmo se hace uso del algoritmo de encaminamiento B.A.T.M.A.N, implementado en una red *Ad-hoc* construida con Raspberries Pi.

Capítulo 1

Introducción

A medida que las necesidades de la población aumentan, la tecnología lo hace con ellas, ya que su objetivo es mejorar la calidad de vida y facilitar las tareas cotidianas. Un ejemplo de estas tecnologías son las redes de sensores inalámbricas, o Wireless Sensor Networks (WSN), las cuales han tenido un auge considerable debido a su bajo costo de despliegue, y a su facilidad de monitorizar un amplio espectro de parámetros como la detección de agentes químicos, sonidos, temperatura, y vibraciones; también es posible encontrarlas en detección de incendios, en aplicaciones militares que cubren diversos territorios, así como para tener una respuesta rápida ante emergencias por desastres naturales. También son utilizadas en los vehículos mediante las redes *Vehicular Ad-Hoc Network (VANET)* permitiendo difundir información sobre tráfico, prevención de accidentes y condiciones ambiéntales; además son consideradas una de las principales tecnologías en la implementación del paradigma *Internet of Things (IoT)*.

Dentro de las WSN, los nodos se encuentran distribuidos en un área geográfica que no tiene una arquitectura de red definida, además de no contar con un nodo central que administra la comunicación. Para ello, los dispositivos trabajan en conjunto recibiendo y retransmitiendo información a través de la red mediante mecanismos que les permite encontrar las rutas más óptimas. Un ejemplo es el protocolo de encaminamiento *Better Approach to Mobile Ad-hoc Networking* (B.A.T.M.A.N). Además, una de las principales características de estas redes es el bajo consumo de energía, dado que estos dispositivos utilizan pequeñas baterías que pueden ser recargadas mediante tecnologías como *energy havesting* [21, 22], prolongando la vida útil y la autonomía de la red.

Las redes inalámbricas presentan diversas ventajas como movilidad, rentabilidad, flexibilidad, ahorro de costos, etc; sin embargo, también presentan desventajas como baja disponibilidad, baja tasa de transferencia de datos o rango de trasmisión limitados, interferencias o pérdida de paquetes, recursos de memoria y procesamiento limitados. Sin embargo, la seguridad es una característica a la que se le debe prestar especial atención, debido a que estos sistemas usan transmisiones inalámbricas, lo que las hace más vulnerables, debido a la escucha de usuarios no autorizados, especialmente porque estos dispositivos transmiten información en todas direcciones. Desafortunadamente, no es posible utilizar los mecanismos convencionales de seguridad, debido a los recursos limitados que presentan los dispositivos en cuanto a su alcance de transmisión, procesamiento, capacidad de almacenamiento y suministro de energía [2]. Por lo que el desarrollo de estudios que permitan incrementar la seguridad se convierte en un punto esencial dentro de esta área.

La información que se transmite entre las redes de sensores genera un gran interés para ciertos grupos de personas que, de interceptarla, estos podrían beneficiarse de algún modo. Se han reportado diferentes tipos de ataques en la literatura, entre los más relevantes está la alteración de parámetros vitales para el funcionamiento de la red, el análisis de tráfico, el espionaje mediante un nodo malicioso, inundación de mensajes [3, 4, 5], negación de servicios (DoS) como *evaesdropping, sinkhole, sybil, blackhole, greyhole y wormhole*. Un ataque *DoS* afecta la disponibilidad del servicio. Este tipo de ataque puede afectar en diferentes capas, por ejemplo a nivel físico, capa de enlace de datos (capa Mac) o capa de red, con el objetivo de sobrecargar el sistema. El ataque *evaesdropping* es un proceso pasivo a nivel de red, donde un agente capta paquetes o información sin autorización. El ataque *sinkhole* rompe la estructura de la red a través de propagar una mejor ruta a un dispositivo, lo que implica que muchos nodos enviarán su información a través de él. El ataque *sybil*, por otro lado, se presenta como un nodo con múltiples ubicaciones, provocando un mal funcionamiento en algoritmos de encaminamiento con base en la topología de la red. El ataque *blackhole* se realiza a través de un nodo malicioso donde todo el tráfico de paquetes que recibe es destruido por este nodo y si el nodo malicioso se hace pasar como el nodo sumidero, el ataque es mucho más agresivo.

A pesar de que los ataques mencionados representan un gran reto y preocupación para la seguridad de las WSN, el ataque *wormhole* es uno de los ataques con mayor dificultad para ser contrarrestado, especialmente porque no afecta la comunicación y los nodos no son capaces de detectar perdida de información. Para ejemplificar este ataque, en la figura 1.1 se observa que los nodos X y Y son los dos extremos del enlace *wormhole*, ejemplificados de color rojo. Todos los paquetes recibidos de uno de los extremos del ataque son repetidos en el otro, de tal suerte que todas las transmisiones generadas en el vecindario de X serán escuchadas en el vecindario de Y y viceversa. El efecto generado por el ataque es que todos los nodos en la región A asumen que los nodos en la región B son sus vecinos a un salto y viceversa. Por ejemplo, el trafico entre los nodos a y e, por efecto de ataque, toma solo un salto mientras que en ausencia de este se necesitarían de múltiples saltos para lograr la conexión, por lo que las tablas de encaminamiento de los nodos en las regiones A y B se verán alteradas.

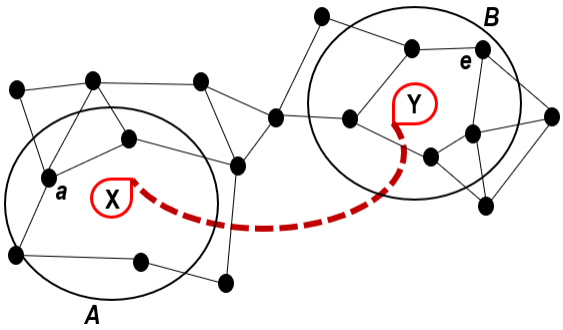


Figura 1.1: Descripción del ataque *wormhole*.

El ataque *wormhole* puede explotar los datos de distintas formas, desde intentar romper el cifrado, modificar los mensajes o incluso alterar paquetes [8]. Esto hace que este ataque sirva como partida para muchos otros ataques más agresivos y severos [14], por ejemplo, la

red podría estar bajo otros ataques como *sinkhole* y *sybil*.

1.1. Definición del problema

El ataque *wormhole* suele ser difícil de detectar debido a que la integridad de la comunicación no se ve afectada. Sin embargo, este ataque puede perjudicar la estructura de las rutas e interferir en la transmisión de datos, lo que provocaría fallas en los algoritmos de localización y encaminamiento. Aunque existen varias propuestas para contrarrestar este ataque, muchas de ellas tienen varias desventajas. Algunas requieren del conocimiento de la topología parcial o total. Otras suponen que todos los nodos tienen una cobertura uniforme, o deben calcular parámetros previamente para estimar dónde está el ataque. Otras propuestas requieren que se utilice *hardware* especializado, como antenas direccionales o relojes altamente precisos, sin embargo, esto eleva el costo de los dispositivos. Algunos trabajos implican un despliegue homogéneo en los sensores, por lo que no funcionan para todo tipo de topologías. También existen propuestas que estudian cómo el ataque *wormhole afecta* al algoritmo de localización *DV-Hop* [10, 11, 12, 13, 15, 16], sin embargo, ninguna de ellas contrarresta el ataque.

Si no se contrarrestan los problemas de seguridad en este tipo de redes, sus aplicaciones podrían impactar directamente a vidas humanas, por ejemplo, cuando los dispositivos son sensores que miden algún parámetro biológico, un atacante podría bloquear o incluso cambiar la información que está destinada hacia alguna institución de salud para la monitorización de los signos vitales de una persona, incluso, si es un sensor que mide los parámetros del correcto funcionamiento de un dispositivo, como un motor, una válvula o algún otro similar que necesita estar bajo vigilancia, el bloqueo de esta información o algún ataque a la integridad de esta, podría comprometer una o más vidas humanas. Y aunque estos serian casos donde la falta de seguridad afecta vidas humanas, también existen casos donde la sociedad puede verse afectada desde un punto de vista económico, por ejemplo, si un campo de cultivo utiliza una red de sensores para supervisar el estado del suelo para garantizar el bienestar del producto cultivado, un ataque a la información transmitida a través de esta red podría derivar en perdidas económicas, si la información es incorrecta, o no ha llegado a su destino para ser procesada. Por lo que es necesario buscar soluciones que se adapten a los recursos disponibles de los dispositivos que las componen, así como viables desde el punto de vista económico, para garantizar una ventaja tecnológica al implementar este tipo de tecnologías.

1.2. Metas.

1.2.1. Meta general.

Implementar el algoritmo propuesto en [1] en un escenario que contempla variables que no son tomadas en cuenta en una simulación, como es la interferencia, las fluctuaciones de la señal inalámbrica, cobertura no homogénea, variaciones de la señal debido a la presencia de obstáculos, con la finalidad de cuantificar la efectividad del algoritmo en un escenario real.

1.2.2. Metas particulares.

- Crear una red Ad-hoc de sensores bajo el protocolo de encaminamiento B. A. T. M. A. N. implementada en placas *Raspberry* Pi.
- Implementar el conjunto de algoritmos propuestos en [1], y verificar su funcionamiento mediante pruebas físicas a partir de la creación de diferentes escenarios.
- Monitorizar el comportamiento de la red con y sin la presencia del ataque *wormhole* y comparar los resultados con los obtenidos en la simulación en [1] para concluir la eficacia del algoritmo.

1.3. Metodología.

Para llevar a cabo de forma satisfactoria los puntos planteados en la sección anterior, se proponen las siguientes etapas del proyecto:

Primero, se hace el análisis de los algoritmos para su implementación en esta tesis: *DV-Hop* y *Spanning Tree*, utilizando el algoritmo de encaminamiento B.A.T.M.A.N.

Posteriormente se establece un escenario real en el que se analice el comportamiento de la red con y sin ataque.

Finalmente, con la implementación del algoritmo, se ejecuta la contra-medida y se cuantifica el número de veces que el algoritmo es capaz de detectar y eliminar el ataque bajo diferentes escenarios.

1.4. Justificación

Si bien existen múltiples opciones para intentar detectar un ataque *wormhole*, muchas de estas pueden significar un costo extra en recursos como el tiempo de procesamiento o el uso de hardware adicional, entre algunos otros que se han mencionado anteriormente en la definición del problema. A diferencia de las propuestas existentes, este algoritmo no representa un costo elevado en procesamiento, ya que solo requiere del conocimiento del 1-vecindario y de algunos elementos en la red que sepan su ubicación.

Este proyecto tiene la finalidad de implementar físicamente la solución propuesta en [1], la cual muestra que es capaz de detectar un 98 % de casos cuando se tienen 5 nodos ancla. Las simulaciones presentadas en [1], de igual forma fueron escritas en Python y constaron de diferentes escenarios en los cuales las entradas que variaron fueron el número de nodos (desde 10 hasta 100), así como el número de *hook points* presentes en la red, que iban desde 3 *hook points* hasta 10. Además, se realizaron las simulaciones con la longitud del ataque *wormhole* fija (como es el caso para este trabajo) y de igual forma se realizaron simulaciones en las que la longitud del ataque es variable.

1.5. Contribución.

A diferencia del algoritmo propuesto en [1], en el cual las pruebas se realizaron bajo simulaciones en un ambiente ideal, en esta tesis se propone la implementación de dicho algoritmo bajo los efectos de un ambiente real, en el cual se presentan problemas como la atenuación de la señal, las fluctuaciones lentas, las multitrayectorias y obstáculos. Además,

de las implicaciones del uso de dispositivos reales a los que se debe adaptar la contra medida del ataque *wormhole*.

1.6. Estructura de la tesis.

Con el fin de alcanzar las metas planteadas dentro de este trabajo de investigación, se propone seguir la siguinte estructura:

En el capítulo 2, se presentan los trabajos más relevantes para mitigar el ataque *wormhole*.

En el capítulo 3, se tratan los algoritmos que fueron implementados, así como su funcionamiento en conjunto para hacer frente al ataque *wormhole*. También se incluye información al respecto de las redes inalámbricas. Específicamente, las redes Ad-hoc y los protocolos de encaminamiento.

En el capítulo 4, se incluyen los puntos más relevantes para la configuración de las tarjetas Raspberry Pi, la implementación, la definición del espacio de pruebas y consideraciones adicionales.

En el capítulo 5, se presenta el análisis de los resultados y se muestran los cálculos realizados para conocer la efectividad del algoritmo implementado.

Finalmente, en el capítulo 6 se incluyen las conclusiones, y las perspectivas de investigación.

Capítulo 2

Antecedentes

En este apartado se realiza una revisión de los trabajos más importantes sobre como contrarrestar el ataque *wormhole*, y de manera especial aquellas investigaciones en las que se utilizan el algoritmo de localización *Distance Vector-Hop*.

2.1. Estado del arte para el ataque *wormhole*

Actualmente existen diversas propuestas sobre cómo hacer frente a las amenazas que implica un ataque de tipo *wormhole*, sin embargo, algunas de ellas requieren tener en consideración factores como suponer que la red sólo puede estar bajo un único ataque *wormhole*, por lo que se terminan descartando aquellos escenarios en los que dos o más ataques trabajan conjuntamente, debido a que es imposible detectarlos. Otras propuestas involucran un alto grado de procesamiento [8, 14], otras más requieren conocer el modelo de comunicación dentro de la red, la sincronización de relojes y/o necesitan elementos específicos como protocolos de encaminamiento (por ejemplo *AODV, DSR*) [8].

TTM, por ejemplo [6], propone un mecanismo que se basa en el tiempo de transmisión y diseñado específicamente para el protocolo *AODV*. Es desplegado durante la fase de encaminamiento y calcula el tiempo de transmisión entre dos nodos vecinos que están dentro del radio de comunicación uno del otro, de forma que al existir dos nodos falsos que formen parte del ataque, se presentará un tiempo de transmisión considerablemente mayor que el de los vecinos reales.

En [14] se propone un algoritmo para la detección del ataque en redes multisalto [7], sin embargo, requiere la ayuda de los modelos de comunicación y la distribución de los nodos.

En *TrueLink* [8] se propone un método donde un nodo dentro de la red puede verificar la existencia de un enlace directo con otro nodo. La verificación en este método consiste en dos etapas: la primera etapa se lleva a cabo mediante el uso de la sincronización de relojes bajo restricciones de tiempo, dentro de las cuales es imposible que reenvíen el intercambio de mensajes con nodos lejanos. La segunda fase es de autentificación, en la cual se utilizan mensajes firmados entre los nodos involucrados. Para este método es necesario hacer uso de un protocolo de encaminamiento seguro, donde *TrueLink* es independiente a este pero aprovecha esta ventaja.

El mecanismo *WormCircle* [9] se basa en términos de dominios geométricos, actuando de manera similar a la propagación de una onda en la superficie del agua, donde la presencia del ataque se generara por la difracción alterando la topología de conexión de la red. Y realiza

la detección a partir del uso de la información de conexión local sin requerir hardware especializado, sin embargo, detectar más de dos ataques es muy difícil para el algoritmo.

2.2. Estado del arte para el ataque wormhole usando *Distance Vector-Hop*

Esta sección contiene una breve descripción de los trabajos más cercanos con el algoritmo *DV-Hop*. Por ejemplo, en [15, 16] requieren que los nodos ancla estén dentro de la zona del ataque, o que el radio de cobertura sea homogéneo [12, 15], incluso en [11, 16] requieren que los nodos se distribuyan de manera uniforme en un área geográfica. Otras se basan en suposiciones como que el canal de comunicación entre los nodos maliciosos es pequeño o la necesidad de realizar cálculos adicionales para la detección del ataque [10, 12, 16]. Existen propuestas que tienen la necesidad de monitorear los cambios en la red, por lo que demandan un alto grado de procesamiento a pesar de únicamente utilizar información local [14]. Otras propuestas requieren de elementos adicionales como un valor determinado de potencia recibida ($RRSI$), tiempos (TTL) en el que un mensaje es válido dentro de la red [10], y algunos trabajos no son capaces de localizar la zona en que el ataque se efectúa a pesar de detectar anomalías en la red [11].

Una característica que tienen en común todas estas propuestas es que la red en su etapa inicial no existe el ataque , por lo que, si existiera un ataque en esta etapa, sería muy complicado lograr la detección de manera oportuna.

En *WormHole-Free DVHop (WFDV)* [10], se propone un algoritmo compuesto de dos fases. La primera fase tiene como objetivo evitar la contaminación del ataque, para ello cada nodo crea su propia lista de vecinos e intenta encontrar enlaces sospechosos, para que de esta forma cada nodo pueda ir eliminando los enlaces que considera como sospechosos. Mientras que la segunda fase puede establecer el algoritmo de localización, ya que los enlaces han sido eliminados.

DWDV [11] es un mecanismo que tiene la finalidad de mitigar el ataque al reducir el error de localización del algoritmo *DV-Hop* de la mayoría de nodos, a través del calculo previo de la métrica "salto normal". Si entre dos nodos vecinos se obtiene un salto mayor a esta medida, se considera que la red se encuentra bajo ataque. Sin embargo, esta propuesta no toma en cuenta las fluctuaciones típicas de una señal inalámbrica.

El algoritmo *AWLDV-Hop* [12] propone una contra medida al *Multi-Wormhole-Node-Link (MWNL)*. Para ello es necesario crear una lista de vecinos *NL* que es usada para encontrar nodos ancla sospechosos calculando la distancia a otros nodos ancla. Con esto los nodos atacados transmiten un mensaje para distinguir las áreas atacadas.

En WRL [13] se propone un algoritmo que consta de cuatro etapas. En la primera etapa los nodos ancla envían su localización a toda la red, mientras que en la segunda etapa se inicia el proceso para la detección del ataque, durante la detección los nodos ancla confían en el conteo de saltos. La tercera etapa consiste en obtener un conteo de saltos a los demás nodos ancla y se estima un valor promedio de salto. Finalmente, se hace una inundación controlada de paquetes con la información generada en las etapas anteriores.

También se ha propuesto un método que se basa en eliminar los bordes del ataque [14], provocando cambios considerables en las rutas cortas. El mayor problema con esta propuesta es utilizar un nodo raíz para el monitoreo, lo cual implica el incremento de recursos a cabo un solo nodo.

En [15] se propone un esquema de localización con base de etiquetas, además del principio

de localización del algoritmo DV-Hop. Este mecanismo genera una lista de pseudo-vecinos para cada uno de los nodos ancla, estas listas son analizadas para clasificar a los nodos atacados en grupos y luego etiquetar a los nodos vecinos reales. De esta forma, dependiendo de la etiqueta se le permite o niega la comunicación.

Por último, el algoritmo *AWDV-hop* [16], es un algoritmo de localización seguro basado en *DV-Hop*. Consta de dos etapas. En la primera etapa se realiza un *floodig* para generar una lista de vecinos relacionados, a partir de este proceso es posible encontrar a los nodos ancla sospechosos a partir del numero de vecinos en las listas. Durante la segunda fase, estos nodos sospechosos calculan la distancia con los otros nodos ancla en la lista marcándose con un 1 o 2 si es que son afectados por el ataque. Finalmente, los nodos marcados con 1 son desconectados.

En este trabajo de tesis se realiza una implementación física que en contraste con las propuestas e investigaciones mencionadas no asume que la red no está atacada desde su inicio, tampoco asume que los nodos tiene el mismo rango de transmisión como es el caso de [10, 11, 13, 16], tampoco requiere de cálculos previos o constantes que permitan identificar el ataque, no necesita de mediciones de parámetros como *RSSI y TTL*, solo es necesario el conocimiento del uno vecindario, además puede usar nodos con radios de cobertura no homogéneos, ni deben estar localizados de manera uniforme en el área geográfica, además de que se requiere de poco procesamiento a nivel local, lo que aporta a conservar recursos de procesamiento y de energía.

Capítulo 3

Marco Teórico

En este capítulo se explica el funcionamiento de los algoritmos DV-Hop y *spanning tree*. También se explica como son utilizados para hacer frente al ataque *wormhole*.

3.1. *Distance Vector-Hop (DV-Hop)*

Actualmente existen varias aplicaciones en las que se encuentran involucradas las redes de sensores y, muchas de ellas requieren que los datos recolectados se relacionen con una posición geográfica, lo que hace necesario que los sensores conozcan o calculen su ubicación. Las técnicas de localización en las redes de sensores se pueden clasificar en dos grandes categorías. La primera está basada en estimaciones de distancias (*range-based*), mientras la segunda usa métodos no basados en distancia (*range-free*). Los métodos *range-based* se basan en técnicas como *AoA, ToA, TDoA* para estimar la distancia entre dos nodos, sin embargo, es importante mencionar que necesitan contar con *hardware* especializado que suele ser costoso. Mientras que los métodos *range-free* utilizan métricas como la proximidad, la probabilidad de conectividad o el conteo de saltos. Esto los hace mucho más accesibles de utilizar, ya que no requieren de *hardware* especializado. Sin embargo, suelen tener un mayor error que los métodos *range-based*. Para el desarrollo de esta tesis se hace uso de un algoritmo *range-free* llamado *DV-Hop*.

El algoritmo *DV-Hop* estima la posición de los nodos basado en un conteo de saltos. Dentro de este algoritmo se encuentran dos tipos de nodos: los nodos ancla o también llamados *hook points*, que son aquellos que conocen su posición dentro de la red, gracias a que cuentan con un *GPS* o que previamente se les asignó una posición respecto a un punto de referencia, y los nodos denominados *nodos desconocidos*, donde estos tratarán de estimar su posición a partir de las coordenadas de los nodos ancla. Es importante tomar en cuenta que no todos los nodos cuentan con *GPS* por razones de costo y consumo energético.

DV-Hop funciona de la siguiente manera: primero se realiza un proceso llamado "inundación" o "*flooding*". Cada nodo ancla transmite un mensaje que contiene su identificador (en este caso el identificador es la dirección MAC de la tarjeta Raspberry Pi), sus coordenadas (x,y) y la información del número de saltos, que al comienzo este valor es cero. Posteriormente, cada vez que un nodo (ancla o desconocido) recibe un mensaje, el número de saltos se incrementa en 1 y se pregunta si este nodo ya lo tenía en su lista, si no es así, lo guarda en la lista de nodos conocidos, si ya lo conocía, entonces compara la información del número de saltos, si ésta es menor, entonces se actualiza la información del nodo ancla. Cuando termina

este proceso, todos los nodos (desconocidos y anclas) de la red saben a cuantos saltos están de cada nodo ancla. Posteriormente, cada uno de los nodos ancla calculan la distancia media por salto *average hop size*, esta distancia se obtiene promediando las distancias a los demás nodos ancla sobre los conteos de saltos como se muestra a continuación:

$$\overline{hopsize} = \frac{\sum_{j \neq i} \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{\sum_{j \neq i} hops_{ij}}, \tag{3.1}$$

donde (x_i, y_i) y (x_j, y_j) son las coordenadas de los nodos ancla X_i y X_j, respectivamente, mientras que $hops_{ij}$ es el número de saltos entre X_i y X_j. Con este cálculo, los nodos ancla envían a la red el valor estimado para que todos los nodos usen el valor del *hopesize* como medida de un salto. Finalmente, los nodos desconocidos utilizan la información recolectada (número de saltos a cada nodo ancla y el *hopesize*) para aplicar un método de multilateración que se resuelve a partir de un sistema de ecuaciones, con el fin de establecer una posición geográfica. El método de multilateración se observa en la figura 3.1, donde N1, N2 y N3 representan los nodos ancla, mientras el nodo desconocido está representado en rojo. Es importante mencionar que la información de la distancia (d1, d2, y d3) se obtiene al multiplicar el *hopsize* por el número de saltos a un nodo ancla. Para esta tesis no se requiere calcular las coordenadas de los nodos desconocidos, solo se requiere saber si existe un área de intersección entre las circunferencias del método de multilateración. Por ejemplo, en la figura 3.2a se muestra el caso en el que no existe intersección entre todas las circunferencias pertenecientes a cada nodo ancla, mientras en la figura 3.2b no hay suficientes nodos anclas para estimar la posición.

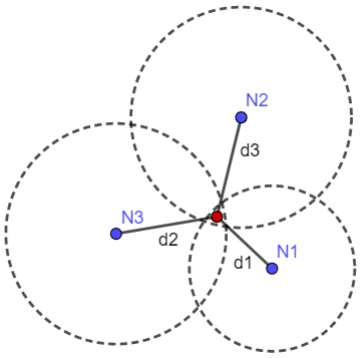
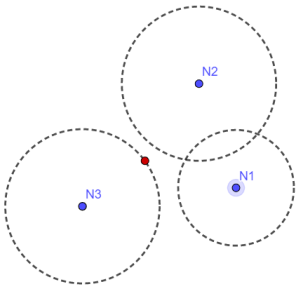
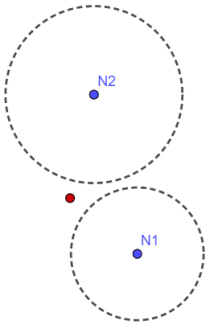


Figura 3.1: Método de trilateración.

Con la finalidad de ejemplificar el funcionamiento del algoritmo *DV-Hop* se plantea el escenario de la figura 3.3 en donde los nodos azules representan a los nodos desconocidos, mientras los verdes representan a los nodos ancla (H1, H2, H3), respectivamente. Bajo este escenario, se muestran dos casos. La figura 3.4 muestra a la red libre de ataque, mientras la figura 3.5 muestra a la red bajo un ataque *wormhole*. En ambas figuras se cuenta con dos nodos desconocidos que intentan auto-localizarse, los cuales están representados con color cian y magenta. Cuando la red se encuentra libre de ataque (figura 3.4), el nodo cian se encuentra a 4, 4 y 1 saltos con respecto a H1, H2 y H3, respectivamente, mientras que el nodo magenta se encuentra a 1, 2 y 2 saltos, respectivamente.



(a) Error por falta de intersección entre las circunferencias



(b) Error por nodos ancla insuficientes

Figura 3.2: Errores en la localización de nodos desconocidos.

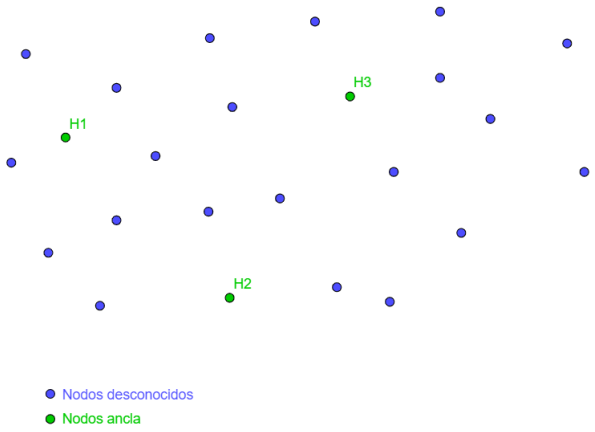


Figura 3.3: Escenario de ejemplo DV-Hop.

En cambio, cuando la red se encuentra bajo un ataque *wormhole* como es el caso de la figura 3.5, el nodo magenta mantiene el mismo número de saltos hacia los nodos ancla (1, 2 y 2), debido a que se encuentra fuera del rango de ataque (representado en la misma figura en color rojo), por lo que la cuenta de saltos no se ve afectada. Mientras que para el nodo cian, el nodo ancla H1 está a dos salto, cuando en realidad está a 4 saltos de él. Para las anclas H2 y H3 no se ven afectadas y mantiene el mismo número de saltos. Esto produce un error en la auto-localización. Este error se debe a que el cálculo de los saltos también afecta la intersección entre las circunferencias en el método de trilateración. Provocando un comportamiento irregular en el algoritmo *DV-Hop* cuando está bajo el ataque *wormhole*.

Tomando el mismo escenario de la figura 3.3, se plantea un escenario en el que se muestra la alteración de las rutas debido al ataque *wormhole*. La figura 3.6 muestra en color cian los saltos que daría el nodo A para alcanzar al nodo B cuando la red se encuentra libre de ataque, mientras se muestra en color rojo los saltos que daría cuando la red se encuentra bajo el ataque. Cuando la red está libre de ataque, entre el nodo A y el nodo B existen 5 saltos, sin embargo, cuando la red está bajo ataque existen 3 saltos, lo que provoca que ambos nodos realicen una estimación incorrecta para el tamaño del *hopsize*. Al mismo tiempo el

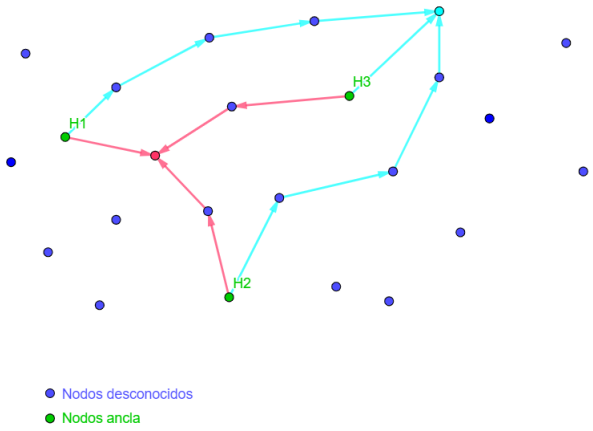


Figura 3.4: Ejemplo DV-Hop sin ataque.

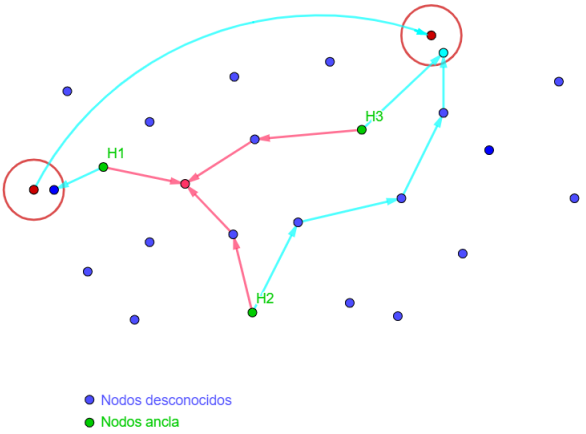


Figura 3.5: Ejemplo DV-Hop con ataque.

ataque afecta la información de los nodos cercanos a ellos, ya que se asumirá un número de saltos menor entre los dos nodos.

Es importante hacer notar que el algoritmo *DV-Hop* se utiliza como primera etapa para localizar posibles zonas de riesgo. Esto se debe al hecho de que los nodos cercanos a los nodos maliciosos presentan fallas en su auto-localización.

3.2. *Spanning Tree*

Un árbol de expansión, también conocido como *Spanning Tree* es un subgrafo de un grafo conectado no dirigido. Este subgrafo incluye todos los vértices del grafo con el mínimo número de aristas. Es posible calcular el número total de árboles de expansión que un grafo puede crear con n vértices de la siguiente manera:

$$n^{n-2}$$

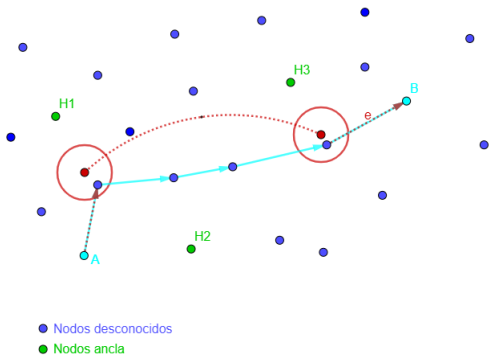


Figura 3.6: Ejemplo simple de rutas con ataque y sin ataque.

(a) Grafo conectado no dirigido (b) Árbol de expansión

Figura 3.7: Representación de un grafo y su árbol de expansión.

Existe una propiedad que un árbol de expansión siempre debe cumplir: si existen n vértices en un grafo, entonces el árbol de expansión debe tener $n-1$ aristas. La figura 3.7 muestra la representación de un grafo conectado no dirigido en la figura 3.7a y uno de los posibles árboles de expansión en la figura 3.7b.

Todo grafo conectado y no dirigido tiene al menos un árbol de expansión. Un árbol de expansión puede ser denominado como un árbol de expansión mínimo cuando presenta el mínimo peso en todas sus aristas. Este peso puede representar diferentes métricas que representan costos entre dos vértices, por ejemplo, puede representar la distancia entre dos nodos o el ancho de banda en el canal.

Las propiedades generales del árbol de expansión son las siguientes:

- Un grafo conectado puede tener más de un árbol de expansión.
- Todos los árboles de expansión posibles tienen el mismo número de aristas y vértices.
- El árbol de expansión no tiene ningún ciclo (bucles).
- Eliminar una arista del árbol de expansión hará que el grafo se desconecte.
- Agregar una arista al árbol de expansión creará un ciclo.

Es posible realizar el cálculo de un árbol de expansión de forma distribuida sin la necesidad de tener conocimiento del grafo completo, para lograrlo se requiere tener dos conjuntos:

- El *conjunto A* esta formado por aquellos nodos miembros del 1-vecindario del nodo que se esta analizando, es decir, aquellos nodos que están conectados directamente.

(a) Árbol de expansión inicializado por A

(b) Árbol de expansión inicializado por E

Figura 3.8: Obtención de Spanning Tree diferente en la misma red.

- El *conjunto B* contiene el nodo por el que fue descubierto, llamado *padre*, y la resta entre el conjunto A y B.

Para llevar a cabo la creación del árbol de expansión se debe considerar que para el nodo que inicia la creación del árbol, su conjunto B está vació, ya que este nodo no tiene un padre que lo descubra. Cada nodo debe enviar un mensaje que se compone del resultado de restar los conjuntos $A - B$, el cual contiene aquellos elementos que están en el conjunto A, pero no están en B. Si esta resta resulta en un conjunto vacío, este nodo no transmitirá el mensaje. En esta tesis cuando un nodo obtiene un conjunto vacío se le conoce como *breakpoint (BP)*.

En la figura 3.8 se muestra un escenario que ejemplifica estas propiedades a partir de un grafo con 9 nodos, lo que implica que es posible crear hasta $9^{(9-2)}$ árboles de expansión, o lo que es lo mismo 4782969, formados por 8 aristas. Primero, se obtiene el árbol de expansión a partir del nodo A (ver figura 3.8a) y posteriormente a partir del nodo E (ver figura 3.8b), esto con la finalidad de mostrar que es posible obtener diferentes resultados para el árbol de expansión en la misma red.

En la figura 3.9 se muestran dos árboles de expansión obtenidos a partir del algoritmo distribuido, en ellos se muestran todas las conexiones existentes entre los nodos A, B, C, D, E, F, G, H e I. Las aristas que forman el árbol de expansión se ven representadas con líneas continuas, mientras que las líneas que se eliminaron durante el proceso se han señalado con líneas punteadas, únicamente con la finalidad de mostrar que alguna vez existieron conexiones.

(a) Primer árbol de expansión inicializado por G

(b) Segundo árbol de expansión inicializado por G

Figura 3.9: Obtención del Spanning Tree a partir del mismo nodo inicializador.

3.2.1. Ejemplo de construcción del algoritmo *Spanning Tree*

Dentro de este ejemplo no se toma en cuenta la existencia del ataque *wormhole*. La figura 3.10 muestra una red conformada por 11 nodos, los cuales son marcados del N1 al N11. El árbol de expansión estará formado por 10 aristas, y existirán un total de 11^9 posibles árboles.

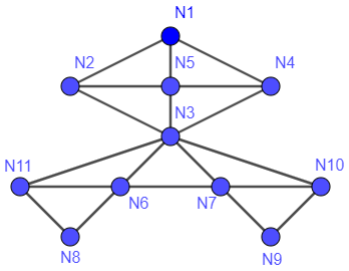


Figura 3.10: Red inicial.

Para este ejemplo se toma al nodo N1 como el inicializador del algoritmo y aleatoriamente se propone el siguiente orden: N1, N2, N5, N4, N3, N11, N6, N7, N10, N8, N9. En el caso del nodo N1, el cual inicia el algoritmo, se tiene que:

$$A = \{N2, N4, N5\} \qquad B = \emptyset \qquad A - B = \{N2, N4, N5\}$$

Se puede ver que el conjunto B es vacío, debido a que el nodo N1 está iniciando el árbol de expansión y no fue descubierto por alguien más. El resultado de la resta de los conjuntos A y B indica a qué conjunto de nodos se les enviará el mensaje $(A - B)$ por medio de multicast. Es posible visualizar esto en la figura 3.11, donde el nodo N1 está en rojo, y los nodos en azul reciben el mensaje $A - B$.

Para el nodo N2:

$$Padre = N1 \qquad A = \{N1, N3, N5\} \qquad B = \{N1, N2, N4, N5\} \qquad A - B = \{N3\}$$

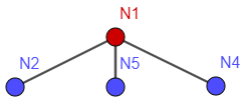


Figura 3.11: N1 inicializa el algoritmo *Spanning Tree*.

Es importante ver que para el nodo N2, su conjunto B son los nodos vecinos del padre y el padre (N1) enviados previamente por multicast, es decir, el mensaje $(A - B)$.

Para el nodo N5:

$$Padre = N1 \qquad A = \{N1, N2, N3, N4\} \quad B = \{N1, N2, N4, N5\} \qquad A - B = \{N3\}$$

Para el nodo N4:

$$Padre = N1 \qquad A = \{N1, N3, N5\} \quad B = \{N1, N2, N4, N5\} \qquad A - B = \{N3\}$$

Para el nodo N3:

$$Padre = \{N2, N4, N5\} \qquad A = \{N2, N4, N5, N6, N7, N10, N11\}$$

$$B = \{N2, N4, N3, N5\} \qquad A - B = \{N6, N7, N10, N11\}$$

En la figura 3.12 se observa que el nodo N5 y los nodos N2 y N4 son padres del nodo N3, lo cual incumple la propiedad de que un nodo no puede tener más de un padre (bucles), por lo que es necesario eliminar la conexión que existe entre el nodo N3 y N4, y la conexión que existe entre N3 y N5 ya que fueron las últimas conexiones que se hicieron. Es importante observar que para crear el conjunto B se consideran los mensajes enviados por N2, N5 y N4.

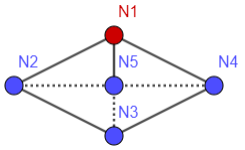


Figura 3.12: Aparición de un bucle en el árbol de expansión.

En la figura 3.13 el árbol de expansión ya no cuenta con el bucle que antes existía y el nodo N3 ha agregado nuevas ramas.

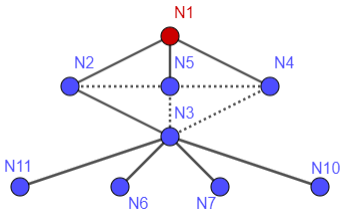


Figura 3.13: Eliminación del bucle entre N4 y N5.

Para el nodo N11:

$$Padre = N3 \qquad A = \{N3, N6, N8\}$$
$$B = \{N3, N6, N7, N10, N11\} \qquad A - B = \{N8\}$$

Para el nodo N6:

$$Padre = N3 \qquad A = \{N3, N7, N8, N11\}$$
$$B = \{N3, N6, N7, N10, N11\} \qquad A - B = \{N8\}$$

Para el nodo N7:

$$Padre = N3 \qquad A = \{N3, N6, N9, N10\}$$
$$B = \{N3, N6, N7, N10, N11\} \qquad A - B = \{N9\}$$

Para el nodo N10:

$$Padre = N3 \qquad A = \{N3, N7, N9\}$$
$$B = \{N3, N6, N7, N10, N11\} \qquad A - B = \{N9\}$$

En la figura 3.14 se muestra la aparición de nuevos bucles en el árbol de expansión. Los nodos N8 y N9 cuentan con dobles padres, por lo que hay que eliminar una de las conexiones existentes, de tal manera que ambos nodos cuenten con un único padre, similar a como ocurrió con el nodo N3.

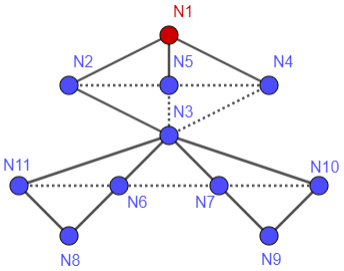


Figura 3.14: Aparición de nuevos bucles en el árbol de expansión.

Para el nodo N8:

$$Padre = \{N11, N6\} \qquad A = \{N6, N11\} \qquad B = \{N11, N6, N8\} \qquad A - B = \{\emptyset\}$$

Para el nodo N9:

$$Padre = \{N7, N10\} \qquad A = \{N7, N10\} \qquad B = \{N7, N10, N9\} \qquad A - B = \{\emptyset\}$$

La figura 3.15 muestra el árbol una vez que ya no existen bucles en él. Hasta este punto el árbol de expansión se encuentra completo, las conexiones eliminadas se observan con líneas punteadas y las conexiones que persisten con líneas continuas.

Finalmente, el árbol de expansión resultante para este ejemplo se puede visualizar en la figura 3.16, en la cual únicamente se encuentran las conexiones persistentes.

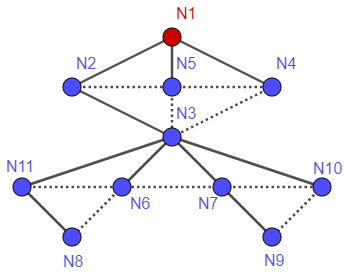


Figura 3.15: Eliminación de bucles en el árbol de expansión.

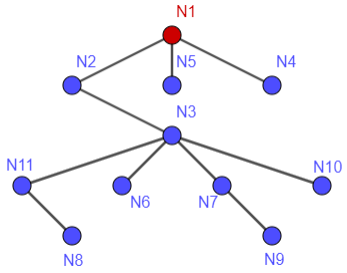


Figura 3.16: Árbol de expansión resultante.

3.2.2. Ejemplo de *Spanning Tree* bajo un ataque *wormhole*

En este apartado se presenta un ejemplo de construcción de un *Spanning Tree* considerando que existe el ataque *wormhole*. El radio de cobertura del *wormhole* ha sido identificado con una circunferencia de color rojo que representan las antenas del ataque. En la figura 3.17 se observa que los nodos que se ven afectados por el ataque son los nodos N2, N7, N9 y N10. Para la realización de este ejemplo se sigue el mismo orden que se ha seguido para el caso anterior, en el que no existe un ataque en la red.

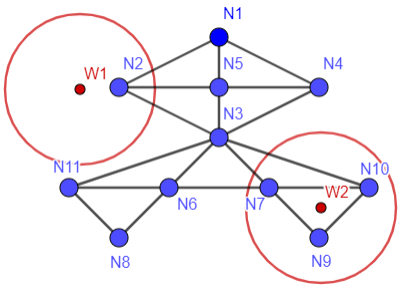


Figura 3.17: Red inicial bajo ataque *wormhole*.

Para el nodo N1:

$$A = \{N2, N4, N5\} \qquad B = \emptyset \qquad A - B = \{N2, N4, N5\}$$

Al calcular los conjuntos del nodo N1 obtenemos los mismos conjuntos que se han obtenido en el caso sin ataque *wormhole*, por lo que, hasta ahora la construcción del árbol de expansión es correcta.

Para el nodo N2:

$$Padre = N1 \qquad A = \{N1, N3, N5, N7, N9, N10\}$$

$$B = \{N1, N2, N4, N5\} \qquad A - B = \{N3, N7, N9, N10\}$$

Para el nodo N5:

$$Padre = N1 \qquad A = \{N1, N2, N4, N3\}$$

$$B = \{N1, N2, N4, N5\} \qquad A - B = \{N3\}$$

Para el nodo N4:

$$Padre = N1 \qquad A = \{N1, N3, N5\}$$

$$B = \{N1, N2, N4, N5\} \qquad A - B = \{N3\}$$

Dado que el nodo N2 se encuentra dentro del rádio de cobertura de la antena W1, ha calculado un conjunto A erróneo, ya que ha considerado como vecinos a un salto a los nodos N7, N9 y N10, los cuáles, en la red original se encuentran a más de un salto de él. En la figura 3.18 se aprecia de manera gráfica lo explicado.

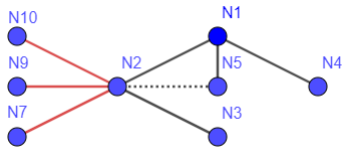


Figura 3.18: Conexiones falsas entre los nodos que se encuentran dentro del radio de cobertura de las antenas W1 y W2.

Para el nodo N3:

$$Padre = \{N2, N4, N5\} \qquad A = \{N2, N4, N5, N6, N7, N10, N11\}$$

$$B = \{N2, N4, N3, N5, N7, N9, N10\} \qquad A - B = \{N6, N11\}$$

Para el nodo N11:

$$Padre = N3 \qquad A = \{N3, N6, N8\}$$

$$B = \{N3, N6, N11\} \qquad A - B = \{N8\}$$

Para el nodo N6:

$$Padre = N3 \qquad A = \{N3, N7, N8, N11\}$$

$$B = \{N3, N6, N11\} \qquad A - B = \{N7, N8\}$$

Para el nodo N7:

$$Padre = \{N2, N6\} \qquad A = \{N2, N3, N6, N9, N10\}$$

$$B = \{N2, N3, N6, N7, N8, N9, N10\} \qquad A - B = \{\emptyset\}$$

Para el nodo N10:

$$Padre = N2 \qquad A = \{N2, N3, N7, N9\}$$

$$B = \{N2, N3, N7, N9, N10\} \qquad A - B = \{\emptyset\}$$

Para el nodo N8:

$$Padre = \{N11, N6\} \qquad A = \{N6, N11\}$$

$$B = \{N11, N6, N7, N8\} \qquad A - B = \{\emptyset\}$$

Para el nodo N9:

$$Padre = N2 \qquad A = \{N2, N7, N10\}$$

$$B = \{N2, N3, N7, N9, N10\} \qquad A - B = \{\emptyset\}$$

En el caso del Nodo N11 y N6, ambos envían el mensaje con la información al nodo N8, por lo que éste tiene doble padre y uno de ellos se elimina, como ya se ha explicado en el ejemplo anterior. Tanto el nodo N7, N9 y N10 se encuentran dentro del radio de cobertura de la antena W2, por lo que el cálculo de sus conjuntos es incorrecto, a diferencia del ejemplo anterior en el que la red no se encontraba bajo ataque, el nodo N7, N8, N9 y N10 ahora son *BP* en el árbol de expansión. Se puede ver la estructura final del árbol de expansión en la figura 3.19.

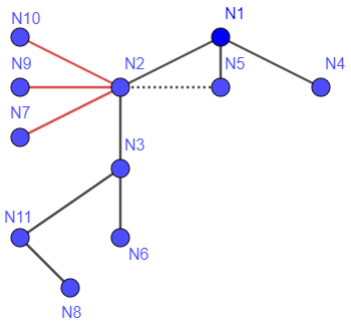
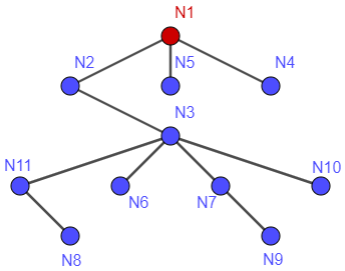
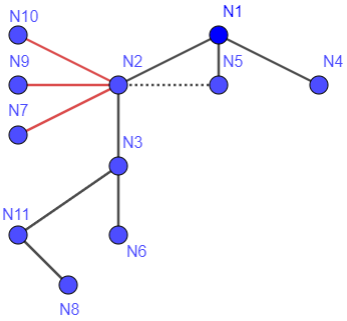


Figura 3.19: Árbol de expansión bajo el ataque *wormhole*.

En la figura 3.20 se observa el árbol de expansión. La figura 3.20a muestra el árbol de expansión para la misma red cuando esta se encuentra sin ataque *wormhole* 3.20a, mientras que en la figura 3.20b, se muestra bajo ataque *wormhole*.



(a) Árbol de expansión sin ataque *wormhole*.



(b) Árbol de expansión con ataque *wormhole*.

Figura 3.20: Comparación de los árboles de expansión obtenidos.

3.2.3. Definición de grafo bipartito.

Un punto importante a considerar es que un *wormhole*, se convierte en un grafo bipartito en términos topológicos.

Dado un grafo G, G es considerado un grafo bipartito si de su conjunto de vértices V se pueden obtener dos conjuntos de vértices independientes $V1$ y $V2$, de forma que cada una de las aristas de G une un vértice del conjunto $V1$ con otro de $V2$. Además estos conjuntos deben de cumplir con:

- $V1 \cap V2 = 0$
- $V1 \cup V2 = V$

Consideramos un grafo bipartito completo si todos los elementos de $V1$ (cian) están unidos a los elementos de $V2$ (magenta), así como se muestra en la figura 3.21.

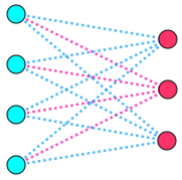


Figura 3.21: Grafo bipartito completo.

Para el ejemplo previo de la construcción del árbol de expansión (ver figura 3.17), se puede observar que el conjunto $V1$ está compuesto por el nodo N2, mientras el conjunto $V2$ está compuesto por el conjunto {N7, N9, N10}.

3.3. *Distance Vector-Hop (DV-Hop)* y *Spanning Tree* como contra-medida para el *ataque wormhole.*

Para contrarrestar el ataque *wormhole*, primero se debe inicializar un *spanninig tree* desde algún nodo desconocido (puede ser elegido de manera aleatoria, o bajo cualquier otro parámetro) con la intención de que conforme se vaya expandiendo el árbol, éste ubique las posibles zonas del ataque al preguntarle a cada nodo si falló su auto-localización. Si esto es cierto, entonces la información del uno-vecindario debe ser enviada al nodo que inicializó el algoritmo de *spanninig tree* a través del uso de los BP que van almacenando la construcción del árbol. Una vez que el nodo iniciador tiene completo el árbol, deberá buscar si entre estos nodos se forma un grafo bipartito, si es así, entonces se inicia un segundo árbol de expansión que avisa a los nodos que son afectados por el ataque, esto se lleva a cabo con la intención de que no tomen en cuenta las aristas que forman al grafo bipartito. De esta manera se suprime el ataque *wormhole*. Para ambos árboles de expansión, los nodos *break-points* (BP) regresan al nodo que inició el árbol la parte que ellos conocen con el fin de que éste reconstruya el *spanning tree*.

La falla en la auto-localización brinda conocimiento de un posible grafo bipartito entre los nodos que se encuentran en las zonas atacada por el *wormhole*, ya que estos presentan mayor probabilidad de estar bajo la influencia del ataque, lo que indica la posible existencia de un grafo bipartito entre los nodos que no pudieron auto-localizarse.

3.4. Protocolos de encaminamiento en redes de sensores

Los protocolos de encaminamiento utilizados en las redes cableadas no son posibles de usarse en las redes inalámbricas *ad-hoc*, debido a que éstas suelen cambiar dinámicamente su topología, y además el medio de propagación inalámbrico es inherente a la interferencia. Estas redes son diseñadas tomando en cuenta parámetros de servicio que se pueden resumir en:

- **Fiabilidad:** Es la capacidad que tiene una red para reconfigurarse rápidamente, de tal suerte que los servicios no se vean afectados en caso de caídas. Los protocolos de encaminamiento deben ser capaces de encaminar los paquetes de la manera más rápida y transparente posible.
- **Transparencia:** El usuario debe poder conectarse a la red por medio de distintos puntos de acceso de forma simple.
- **Escalabilidad:** El número de nodos de una red está limitado a la sobrecarga que introduzca el protocolo de encaminamiento. Es necesario encontrar un compromiso entre la información del usuario y la información de control que circula por la red.
- **Calidad de Servicio:** Los nodos encaminan sus paquetes por las mejores rutas basadas en una métrica. Es necesario definir tipos de tráfico para ofrecer un mejor servicio al usuario final (QoS).

Los protocolos de encaminamiento en las redes de sensores se pueden dividir en dos grandes áreas, los protocolos estáticos y los protocolos dinámicos. El encaminamiento estático no suele ser una opción viable para redes con un gran número de nodos. Por este motivo, los protocolos de encaminamiento para este tipo de redes suelen ser protocolos dinámicos.

En general éstos pueden ser clasificados, de acuerdo con el momento en el cual actualizan sus tablas de encaminamiento, como protocolos proactivos y protocolos reactivos:

- Los protocolos **proactivos** son aquellos que buscan mantener permanentemente las tablas de encaminamiento actualizadas, de tal manera que las solicitudes de transmisión de paquetes sean atendidas inmediatamente y exista un conocimiento total sobre el estado de la red. Para ello se requiere de la transmisión periódica de paquetes o como respuesta a algún evento, con información de actualización de rutas o estados. Ejemplos de este tipo de protocolos son *RIP* y *B.A.T.M.A.N.*

- Los protocolos de encaminamiento **reactivos** son aquellos que incluyen nuevas rutas en las tablas de encaminamiento en el momento en el que se vayan a utilizar, por lo que se agrega un retardo significativo adicional cuando pretende encaminar un paquete cuya ruta no se conoce previamente. Estos protocolos envían información de control solo en el proceso de descubrimiento de una nueva ruta, y al no requerir del intercambio permanente de paquetes sobre la red, son más adecuados en entornos en los cuales el ahorro de energía es vital, como por ejemplo las redes de sensores, donde los sensores suelen ser alimentados por pequeñas baterías. Ejemplos de este tipo de protocolos son *AODV, DSR y ABR.*

Las métricas usadas por los protocolos de encaminamiento suelen estar basadas en el número de saltos, el retardo, la estabilidad del enlace, el ancho de banda, las pérdidas de paquetes e incluso en el costo económico. Algunas de las métricas más usadas en protocolos de encaminamiento de redes *ad-hoc* son *Hop Count* (basada en el número de saltos, considerando como salto a un enlace inalámbrico) u otras métricas relacionadas con la calidad del enlace *ETX (Expected Transmission Count), ETT (Expected Transmission Time), WCETT (Weighted Cumulative Expected Transmission Time)*. La elección de la métrica o la definición de su cálculo es un aspecto muy importante que debe considerar las condiciones del entorno en el cual la red se desplegará.

También se pueden clasificar los protocolos de encaminamiento según el nivel al que pertenecen en el modelo OSI (*Open System Interconnection*), específicamente en el nivel 3 o nivel 2. El protocolo B.A.T.M.A.N. trabaja sobre capa 2 de la siguiente forma:

- **Capa 2:** esta capa fue diseñada en un origen para la gestión del acceso al medio, la detección de errores en los enlaces, el control de flujo y la distribución ordenada de los paquetes. Sin embargo, actualmente se han añadido funcionalidades de encaminamiento en este nivel en algunos protocolos, dotando a este tipo de redes de algunas características especiales. El nivel 2 es transparente para las capas superiores, por lo que los protocolos de encaminamiento que trabajen en la capa de enlace también lo serán para la capa de red. Esto simplifica la comunicación con nivel *IP*, donde cualquier topología se convierte en un solo *hub* o segmento *LAN 802.x* ya sea por cable o inalámbrico. Esta transparencia u ocultación de la red física, se ofrece tanto para tramas *unicast, multicast y broadcast* dentro y fuera de la red. El hecho de que un protocolo de encaminamiento trabaje en la capa de enlace, hace que el acceso a la información de la capa física y capa *MAC (Media Access Control)* sea más rápido, mejorando la eficiencia de las decisiones de encaminamiento y reduciendo el tiempo de reacción ante posibles fallos en la red. Además, esto añade la ventaja de ser independiente a esquemas de direccionamientos concretos de nivel 3, por lo que no existen problemas de incompatibilidad con IPv4 o IPv6. También permite a los nodos con

pocos recursos formar parte de la red sin necesidad de implementar todo el nivel *IP*. Sin embargo, una red cuyo protocolo de encaminamiento trabaje en el nivel 2 resulta poco escalable, debido a la falta de jerarquía en el direccionamiento *MAC*, por lo que generalmente este tipo de redes suelen ser de tamaño pequeño a medio.

3.4.1. B.A.T.M.A.N advanced

Better Approach To Mobile Ad-hoc Networking Advanced (B.A.T.M.A.N-adv). Este protocolo de encaminamiento fue diseñado como una mejora del protocolo *OLSR (Optimized Link State Routing)*, ya que éste tenía un alto consumo de recursos. En *B.A.T.M.A.N*, ningún nodo tiene la información completa de la topología de la red puesto que esto no es necesario. Para construir su tabla de encaminamiento, cada nodo mantiene la información sobre el mejor *siguiente salto* hacia el resto de los nodos de la red. Para construir esta tabla, cada nodo transmite mensajes *broadcast* llamados *Originator Messages (OGM)*, con el fin de informar a los nodos vecinos sobre su existencia en la red. Estos vecinos vuelven a difundir los *OGM* basados en reglas específicas con el fin de informar a sus vecinos sobre la existencia del nodo que origino el envió. De esta manera, la red es inundada de forma selectiva con los mensajes *OGM*. Los *OGM* son paquetes de 52 bytes, y están compuestos de la siguiente información:

- La dirección del nodo que inicio el envió del paquete
- La dirección del último nodo que transmitió el paquete
- *TTL(Time-to-live)*
- Un número de secuencia

Con esta información, todos los nodos de la red tendrán certeza de la existencia del resto de nodos, pero no tendrán información topológica acerca de la ruta hacia ellos. Para saber cuál es el mejor enlace o ruta hacia un nodo particular, se hace uso de una ventana deslizante para almacenar los números de secuencia recibidos de cada destino. Este número de secuencia permite diferenciar entre información nueva y obsoleta. La ventana deslizante tiene un tamaño máximo y la cantidad de paquetes recibidos no repetidos. Esta ventana usa una métrica para determinar la calidad de las rutas hacia el resto de los nodos en la red conocida como *TQ (transmission Quailty)*. En la figura 3.22 se puede apreciar un ejemplo de cómo se eligen las rutas, se puede ver que en la figura 3.22a, la calidad del link entre A y B no es buena, por ello el algoritmo elige otra ruta a pesar de que tiene más saltos. Para llegar a esta decición el nodo A envía periódicamente mensajes OMG. Los nodos B, C y D, recibe los OGM de A y calculan la ruta óptima hacia A. En el caso 3.22b, B escoge el enlace B-A como mejor camino hacia A, ya que por este enlace el OGM de A ha llegado más rápido (enlace con mayor calidad). En el 3.22b, cambia su decisión y escoge el enlace B-D como mejor camino hacia A, ya que ahora ha recibido 2 mensajes OGM desde A por el enlace B-D, mientras que sólo ha recibido un mensaje OGM de A por el enlace B-A, es decir, existe mala calidad del enlace entre el enlace A y B.

La primera implementaciones del protocolo *B.A.T.M.A.N* opera en la capa 3 del modelo *OSI*, esta implementación es conocida como *batmand*. Todos los paquetes en esta versión van sobre *UDP (User Datagram Protocol)*, obligando al nodo a manipular la tabla de rutas del kernel. En contraste, la nueva versiones conocida como *batman-adv* utiliza la capa 2 del modelo *OSI*, tanto la información de usuario como el tráfico de control es transportado

(a) (b)

Figura 3.22: Ejemplo de selección de rutas en el protocolo B.A.T.M.A.N.

en tramas *ethernet* utilizando encapsulamiento hasta el nodo destino. Esto hace que todos los nodos parezcan estar conectados con un enlace local de 1 salto desde el punto de vista de IP, ocultando la verdadera topología de red, así como de los cambios en la misma. Al utilizar este segundo nivel hace posible que exista un encaminamiento del flujo de datos más eficiente y una mayor simplicidad a la hora de proveer servicios de *roaming (servicio de itinerancia)* a los equipos clientes.

Para las últimas revisiones del protocolo B.A.T.M.A.N, se ha introducido un nuevo paquete llamado ELP *(Echo Location Protocol*). Estos mensajes son enviados con una periodicidad alta con lo cual es posible comprobar rápidamente el estado de los enlaces locales, por lo que estos mensajes no inundan toda la red.

Los mensajes *OGM* siguen presentes en estas nuevas versiones, ya que la existencia de estos dos tipos de mensajes reduce considerablemente el intervalo de envío de los *OGM* (reduciendo el ancho de banda y la capacidad de procesamiento), lo cual implica una mejora en los tiempos de convergencia y simplificando notablemente el protocolo de encaminamiento.

Actualmente la versión de *batmand* es menos usada en comparación con *batman-adv* que comienza a tener un mayor auge. Sin embargo, debido a la continua evolución que tiene *batman-adv* podemos encontrar muchas versiones, además de contar con una comunidad activa de desarrollo. Ambas implementaciones fueron desarrolladas por la comunidad *OpenMesh*.

Capítulo 4

Desarrollo

Este capítulo abarca la metodología presentada en la introducción de este trabajo de tesis.

Para el entorno de pruebas se requiere contar con los siguientes elementos:

- **Raspberry Pi OS**: anteriormente conocido como Raspbian, es una distribución del sistema GNU/Linux con base en Debian pensada para las placas Rasberry Pi.
- **BATMAN-adv**: Protocolo de encaminamiento en su versión 2019.4.
- **Red ad hoc**: configuración de las tarjetas de red en modo ad-hoc, lo cual permite la conexión entre nodos dentro del rango de alcance.

Primero se debe instalar B.A.T.M.A.N en las Raspberry Pi, sin embargo, primero es necesario instalar *batctl*, el cual es es la interfaz de control y diagnóstico de batman-adv. El procedimiento de instalación y configuración para B.A.T.M.A.N en las Raspberry Pi se encuentra en el Apéndice E. Primero se realizaron pruebas para verificar el envío y recepción de mensajes entre las tarjetas y se configuró la potencia de transmisión en 0 dBm con la finalidad de reducir el área de cobertura deseada para construir la red bajo la topología que se define más adelante.

También se utilizó la herramienta VNC Viewer para monitorizar, con la que fue posible establecer conexiones remotas e identificar el comportamiento de los dispositivos de la red.

4.1. Implementación de los Algoritmos *Distance Vector-Hop* y *Spanning Tree*

En los Apéndices B y C se encuentra el código que contiene los algoritmos *Distance Vector-Hop* y *Spanning Tree*, respectivamente, si bien ambos códigos son muy parecidos, la diferencia entre ellos es que el código del Apéndice B lo ejecuta el nodo *root*, que es un *hook point* que está encargado de iniciar el proceso de *flooding*, por lo que este comienza enviando el mensaje *flooding* a sus vecinos a 1 salto, estos reciben el mensaje con base al código del Apéndice C, a su vez lo reenvían a sus vecinos a 1 salto. A medida de que los nodos reciben los mensajes de otros, se va inundando la red. Los mensajes duplicados son descartados por los nodos una vez que se han recibido, así poco a poco se detiene la inundación. Después

de la inundación, los nodos cuentan con la información acerca de los nodos *hook points* y el número de saltos a los que se encuentra de ellos. Cuando el nodo *root* termina de enviar a sus vecinos a 1 salto los mensajes de *flooding*, este inmediatamente comienza a mandar mensajes para construir el *Spaning Tree*, los demás nodos reciben estos mensajes y van guardando la información a medida de que los reenvían a sus 1-vecinos para comenzarla construcción, posteriormente eliminan los dobles padres que pudieran existir y se reconstruye el *Spaning Tree* hacia atrás mandando mensajes a partir de las hojas hacia la raíz, esto para que al final todos los nodos tengan conocimiento del árbol que se generó. Finalmente, se realiza el cálculo del *average hop size (AHD)*, utilizando toda la información recaba por los mensajes en la inundación de la red. Para calcular dicho valor se promedian las distancias a los *hook points* restantes sobre el conteo de saltos tal y como se muestra en la ecuación 3.1. El nodo *root* al calcular la distancia envía un mensaje a toda la red (comenzando con sus vecinos a un salto) sobre el valor obtenido de *AHD*, a su vez cada uno de los nodos que reciben el mensaje, guardan el valor y lo reenvian, terminan su proceso de escucha en la red, y guardan la información sobre los *hook points*. El último paso a verificar es la presencia de intersecciones dentro de la red, esto es realizado por el nodo *root*, el cual arrojara un mensaje en pantalla si hay o no suficientes intersecciones, considerando que 3 son las suficientes. Los resultados almacenados por cada uno de los nodos se guardan en un archivo de texto que posteriormente son utilizados para el análisis estadístico de las diferentes pruebas que se hicieron.

4.2. Definición del Espacio de Pruebas

Para llevar a cabo las pruebas de los algoritmos implementados se seleccionó como ubicación la explanada del *Museo Universitario Arte Contemporáneo (MUAC)* (ver figura 4.1). En este espacio se esparcieron en el suelo 8 placas Raspberry Pi que fungieron como nodos, de modo que fuera posible formar una red como se muestra en la figura 4.2. Una vez que se colocaron los nodos, se tomaron medidas a partir de un punto de referencia (0,0), para conocer la posición en el espacio seleccionado.

Conocer la posición en la que se encuentran los nodos sirvió para recolectar la información de las coordenadas de los *nodos ancla* (estos nodos se encuentran representados de color verde), en este caso la red contaba con tres nodos de este tipo, el número mínimo necesario para poder llevar a cabo el proceso de auto-localización. Posteriormente, se introducen 2 placas Raspberry Pi adicionales, conectadas mediante un cable *ethernet* para fungir como las antenas del ataque *wormhole*. La red que se encuentra bajo ataque se muestra en la figura4.3. En la figura se muestra el árbol de expansión (representado por las conexiones o aristas con líneas continuas), por lo que las líneas punteadas corresponden a las aristas que han sido eliminadas. También es posible apreciar la posición en la cual se colocó el ataque representado en color rojo.

Figura 4.1: Museo Universitario Arte Contemporáneo (MUAC). [Fotografía], por Museos de México (2016), *https* : *//www.museosdemexico.com/muac*$_1$7.*html*

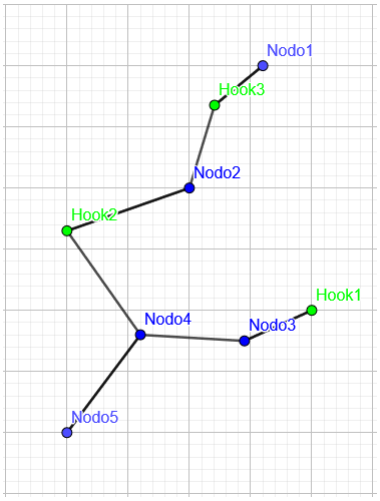


Figura 4.2: Topología de la red sin ataque.

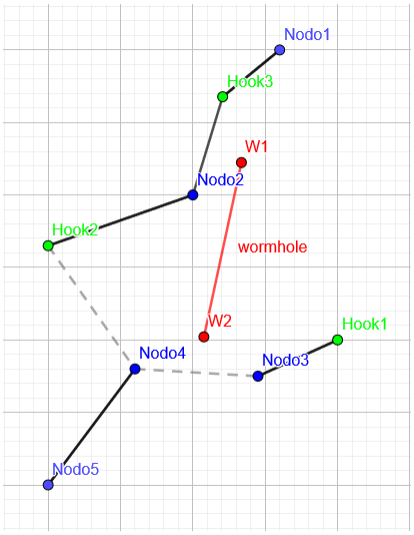


Figura 4.3: Topología de la red con ataque.

Capítulo 5

Resultados

En esta sección se presentan los resultados, así como los elementos necesarios para poner a punto el escenario de pruebas.

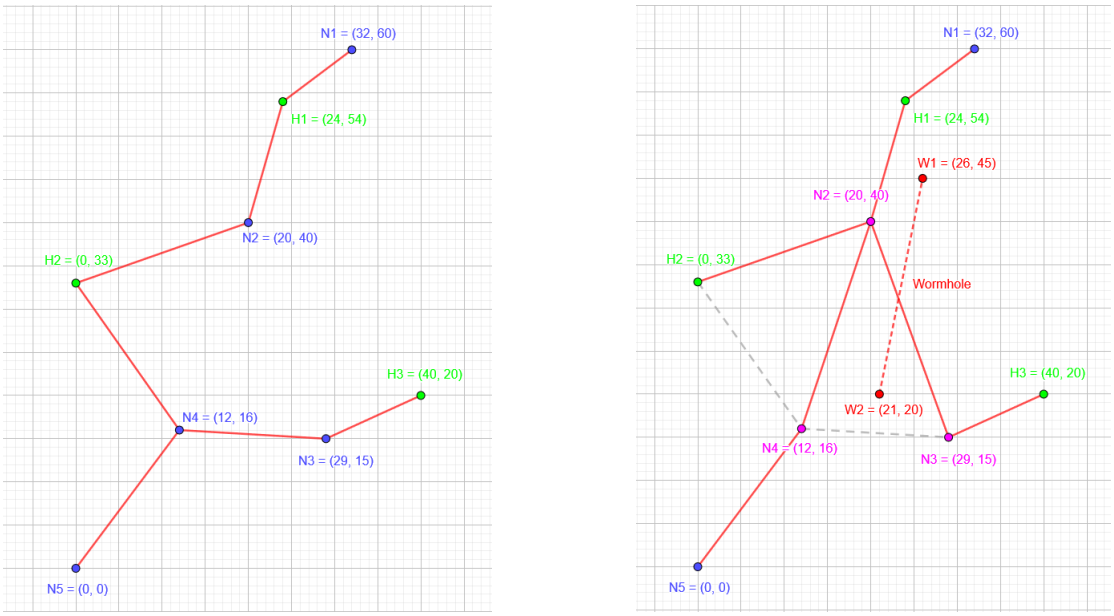
Los parámetros más importantes de los códigos implementados durante la realización de las pruebas tanto para el nodo *root* (nodo que inicia el árbol de expansión), beacon (nodo ancla) y nodo desconocido, se enuncian a continuación (ver Apéndices B y C):

- Mensaje flooding *(msg)*: contiene los datos de los nodos beacon, el cual consta de:
  - IP origen *(origin)*.
  - Dirección MAC dada por el protocolo BATMAN *(mac_ bat)*.
  - Ubicación de beacon *(location)* de la forma (x,y).
- Nodos que lanzan el ataque *(worm)*.
- Registro de los nodos beacon *(reg)*.
- Datos sobre todos los nodos beacon, con sus coordenadas y dirección MAC *(dic_ data)*.
- Medida del *hopsize* calcualdo *(AHD)*.
- Datos para conocer las intersecciones de trilateración *(dic_ inter)*.
- Datos para conocer el tipo de mensaje:
  - Llave *('DELETE')*. Para eliminar dobles conexiones en el *Spanning Tree* se hace uso de este tipo de mensajes. Contiene un diccionario que proporciona la información de quién es el padre del nodo y su dirección MAC.
  - *('REBUILD')*. Para llevar a cabo la reconstrucción del *Spanning Tree* se requiere del envío de mensajes tipo *'REBUILD'*. Contiene un diccionario que proporciona la información de quién es el padre del nodo, su dirección MAC y el conjunto B.

Adicionalmente cada una de las tarjetas *Raspberry* Pi debe contar con un archivo llamado *ip.txt*, el cual contiene su dirección IP, dirección MAC del protocolo *BATMAN-adv*, dirección *broadcast* de la red y su ubicación (este último dato solo es para los nodos *beacon*). Para la identificación de los nodos se asignó la siguiente gama de colores:

- Verde: nodos ancla
- Azul: nodos desconocidos
- Rojo: nodos wormhole
- Rosa: nodos afectados por el ataque

A continuación, se muestran algunas de las pruebas realizadas en el escenario planteado en la figura 4.2, el tamaño físico del escenario fue de 40 m × 80 m, la potencia de transmisión de todas las tarjetas se configuró en 0 dBm, esto hace que las tarjetas tengan un radio de cobertura de aproximadamente 10 m. Se utilizaron tres nodos *beacon*, uno de estos es considerado como nodo *root*, este se encargará de iniciar y reconstruir el *Spanning Tree*. Estos nodos se ubicaron en los puntos (40,20), (0,33) y (23,6,54). Para el caso de los nodos que componen el ataque, como se muestra en la figura 4.3, se ubicaron en los puntos (26,45) y (21,20) conectados entre sí por un cable *ethernet*.



(a) Spanning Tree sin ataque

(b) Spanning Tree con ataque

Figura 5.1: Escenario bajo pruebas.

La figura 5.1a, muestra el escenario en ausencia del ataque, en ella se observa el resultado del *spanning tree* (líneas rojas), el cual es iniciado por el *Hook 3*. Este escenario consta de 8 nodos, de los cuales tres son nodos ancla *(H1,H2 y H3)*. La figura 5.1b muestra el escenario en presencia del ataque, de igual manera, se muestra el *spanning tree* (líneas rojas) que es iniciado por el nodo ancla *H3*. En esta figura se puede ver que los nodos afectados por el ataque son *H3, N2, N3 y N4*. Para este caso, los nodos N2, N3 y N4 son capaces de detectar el ataque debido a que confirman la presencia de una grafo bipartito, dado que son parte de él. Una manera simple de confirmar este ataque es mediante un conteo de saltos entre los dos árboles de expansión. Para ello, se toma un nodo como referencia y se mide a

Nodos	Saltos
H3 $\longrightarrow$ N1	1
H3 $\longrightarrow$ N2	1
H3 $\longrightarrow$ H2	2
H3 $\longrightarrow$ N3	4
H3 $\longrightarrow$ N4	3
H3 $\longrightarrow$ H1	5
H3 $\longrightarrow$ N5	4

Tabla 5.1: Número de saltos sin ataque nodo H3.

cuantos saltos se encuentran todos los demás nodos con respecto del nodo de referencia. Si la desviación estándar varía mucho querrá decir que la topología de la red es muy distinta entre ambos árboles de expansión, lo que indica que existe una gran probabilidad de que la red esta bajo la influencia de un ataque. Para ejemplificar este caso, a continuación se muestra un pequeño estudio estadístico entre la media y la desviación estándar de saltos. Para ello se consideraron los dos árboles de expansión mostrados anteriormente y los nodos que forman parte del 1-vecindario de los nodos dentro del ataque para obtener el promedio de saltos y así verificar si existe un cambio cuando el ataque ya no está presente. Se toma como punto de referencia al nodo H3 dado que se encuentra dentro del ataque.

Análisis estadístico sin ataque:
Tomando como referencia los datos de la tabla 5.1 obtenemos los siguientes datos.

Media de saltos:

$$E(X) = \frac{20}{7} = 2{,}85 \quad \text{saltos} \tag{5.1}$$

Desviación estándar:

$$\sigma^2 = \frac{\sum(X - E(X))^2}{n-1} = \frac{14{,}8575}{7} = 2{,}1215 \tag{5.2}$$

$$\sigma = \sqrt{2{,}1215} = 1{,}45 \tag{5.3}$$

Análisis estadístico con ataque:
Tomando como referencia los datos de la tabla 5.2 obtenemos los siguientes datos.

Media de saltos:

$$E(X) = \frac{14}{7} = 2 \quad \text{saltos} \tag{5.4}$$

Desviación estándar:

$$\sigma^2 = \frac{\sum(X - E(X))^2}{n-1} = \frac{4}{7} = 0{,}5714 \tag{5.5}$$

$$\sigma = \sqrt{0{,}5714} = 0{,}7559 \tag{5.6}$$

Nodos	Saltos
H3 $\longrightarrow$ N1	1
H3 $\longrightarrow$ N2	1
H3 $\longrightarrow$ H2	2
H3 $\longrightarrow$ N3	2
H3 $\longrightarrow$ N4	3
H3 $\longrightarrow$ H1	2
H3 $\longrightarrow$ N5	3

Tabla 5.2: Número de saltos con ataque nodo H3.

Se puede observar que tanto la desviación estándar como la media de los saltos se reducen en presencia del ataque. En cuanto a la desviación estándar se reduce en casi un 50 %, mientras que en el número de saltos casi en un salto. Ambas reducciones indican que la topología en general cambió considerablemente, además que existe un grafo bipartito entre las zonas encontradas por el algoritmo DV-Hop. Para este escenario se realizaron 50 repeticiones para cuantificar cuantas veces el algoritmo es capaz de detectar el ataque bajo condiciones reales de variaciones de la señal, multitrayectorias, interferencia de obstáculos, y personas que pasaban en el escenario mostrado en la figura 4.1. Se consideró un radio de cobertura del ataque *wormhole* de 10 m. La figura 5.2 muestra el total de ejecuciones, de las cuales 30 ocasiones se detecto exitosamente el ataque con la presencia de 3 nodos ancla.

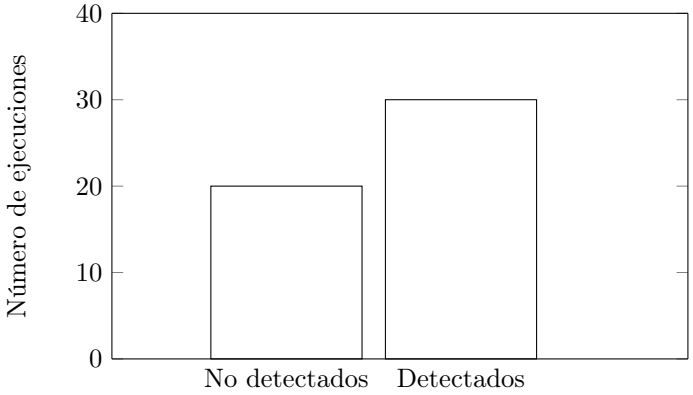


Figura 5.2: Resultados del escenario propuesto.

Capítulo 6

Conclusiones

En este capítulo se incluyen las conclusiones generales a las que se llegaron con la realización de esta tesis, adicionalmente, se agregan las perspectivas de investigación.

6.1. Conclusiones generales

Actualmente, el uso de redes inalámbricas se ha vuelto cada vez más importante en muchas aplicaciones del día a día, ya que existe un gran número de servicios que nos facilitan el trabajo diario, sin embargo, pese a que este tipo de redes tiene muchas ventajas, también tienen desventajas, las cuales son de suma importancia ponerles atención. Una de ellas es que las redes de sensores inalámbricas suelen ser inseguras, por lo que la información que viaja a través de ellas es susceptible a ser interceptada, alterada o modificada. En general, todas las redes inalámbricas o cableadas están propensas a ser atacadas, sin embargo, en el caso de las redes inalámbricas existen ataques muy severos que ponen en riesgo no solo la información si no la estructura de la red. El caso estudiando en esta tesis representa uno de lo ataques más severos, debido a que altera la topología de la red y es la base de múltiples ataques. En esta tesis se han descrito algunos de los trabajos relacionados al ataque *wormhole* y métodos para contrarrestarlo, sin embargo, estas medidas pueden llegar a presentar algunos inconveniencias como son la dificultad para ser implementados, la demanda de recursos e información adicionales, los costos altos al requerir de *hardware* muy especializado. Más aún, todos los trabajas mostrados en esta tesis son simulados y ninguno de ellos se enfrenta a las condiciones reales de un ambiente como el propuesto en este trabajo. Para ello, se implementó el algoritmo propuesto en [1] que es capaz de detectar la presencia de un ataque *wormhole* en una red de sensores mediante estudios topológicos.

Debido a limitaciones en el número de equipo que se tenía disponible para la realización de esta tesis (tarjetas Raspberry Pi, baterías, displays, etc) se realizó un escenario pequeño de pruebas que se adaptó a la cantidad de dispositivos con los que se contaba.

En el capítulo 4 se muestran los resultados obtenidos para esta tesis. Se realizó un total de 50 pruebas en el escenario seleccionado, el cual cuenta con 8 nodos, de los cuales 3 de ellos son nodos *hook points*. En estas pruebas se obtuvo un total de 30 casos detectados y 20 casos no detectados, es decir, en un 60 % de los casos fue posible detectar el ataque.

Haciendo una comparativa con el trabajo presentado en [1], se tomó el escenario más pequeño, el cual cuenta con 10 nodos, de los cuales, 3 son *hook points*. Al igual que en nuestro escenario, en este se realizaron 50 simulaciones de las cuales se obtuvo que 34 casos

Tipo prueba	Nodos Desconocidos	Hook points	% Detectado	% No detectado
Simulación	7	3	68 %	32 %
Física	5	3	60 %	40 %

Tabla 6.1: Comparación de los resultados de las pruebas de simulación con las pruebas físicas

fueron detectados y 16 no lo fueron. Dicho de otro modo, en un 68 % de los casos fue posible detectar el ataque, mientras que en un 32 % no lo fue. La Tabla 5.1 muestra los resultados obtenidos bajo simulación comparados con el experimento real. Se puede observar en esta tabla que los porcentajes de detección y falla son muy similares.

Cabe mencionar que los experimentos en [1] muestran que para un escenario de 100 nodos con 4 nodos ancla, el algoritmo es capaz de detectar un 88 % de los ataques, mientras que para la misma red pero con 5 nodos ancla es capaz de detectar el 92 % de los ataques. Más aún en [1] se muestra que con 10 nodos anclas solo aumenta un 2 % más. Por lo que el número óptimo de nodos ancla está entre 4 y 5 para redes de 10, 20, 50 y 100 nodos.

Es importante mencionar que para fines de esta tesis no fue posible realizar escenarios con un mayor número de nodos, ya que no se contaba con el hardware necesario para poder crecer la red de la forma en la que crece en los escenarios de simulación mostrados en [1]. Por otro lado, tampoco fue posible variar la longitud del cable *ethernet* que conecta las tarjetas que forman el *wormhole* debido a limitaciones de espacio y tiempo disponible para realizar las pruebas en el espacio seleccionado. Esto se debió a que mientras se realizaban las pruebas, las autoridades del museo pedían un permiso por parte de la Universidad para continuar, situación muy compleja, ya que aún era pandemia. Sin embargo, haciendo uso de los recursos y del espacio con el que se contó para la realización de esta tesis fue posible cumplir con los objetivos de implementar el algoritmo y ponerlo a prueba. Por otro lado los objetivos particulares se cumplieron, ya que se implementó una red Ad-Hoc de sensores haciendo uso del protocolo B. A. T. M. A. N. y de placas Raspberry Pi, para lo cual fue necesario realizar una investigación del funcionamiento del protocolo B. A. T. M. A. N. y así poder llevar a cabo la construcción de la red y la comunicación de los nodos mediante *sockets*, de esta forma fue posible monitorizar el comportamiento de la red con y sin la presencia del ataque *wormhole* con el fin de comparar los resultados para concluir la eficacia del algoritmo.

Adicionalmente se detectaron algunos puntos que deben ser considerados importantes desde la creación de la red y el posicionamiento de los nodos en el espacio de pruebas, así como para la creación del ataque *wormhole*. Para el caso de la creación de la red y el posicionamiento en el espacio de pruebas, se contaba con un espacio limitado, por lo que se tuvo que cambiar la potencia de transmisión a 0 dBm para lograr que los mensajes no fueran interceptados por los nodos que se encontraban a una distancia mayor. Otra consideración a tomar en cuenta es que las tarjetas fueron colocadas en el suelo del espacio de pruebas, esto provoca que la comunicación entre los nodos no es confiable debido a que el tipo de antenas utilizado en los dispositivos *Raspberry* son antenas sobre un chip, cuyo patrón de radiación es similar al de una antena de tipo dipolo, es decir, con forma de toroide, esto provoca que, al tener los dispositivos en alturas diferentes se modifica el alcance de la transmisión, ya que al cambiar el plano de elevación, se modifica el modelo de propagación. Esto implica que si se desplaza la altura uno de ellos hacia arriba o hacia abajo, la potencia disminuirá. Como consecuencia de ello, las pruebas se veían afectadas, incluso si se movían los dispositivos para

revisar algún detalle de las *Raspberry*. Además, el colocarlas en el piso genera trayectorias múltiples, dado que el suelo por su composición se considera como un dieléctrico, lo que permite que las señales se reflejen en lugar de ser absorbidas por completo como en el caso de materiales conductores.

Por otro lado, también existieron diferentes causas de interferencia en la comunicación entre los dispositivos, entre ellas están la presencia del espejo de agua que se encuentra frente al edificio del MUAC (Museo Universitario Arte Contemporáneo), la espiga de Rufino Tamayo que se encuentra en la explanada y que, al ser un espacio moderadamente concurrido, el paso de la gente alrededor y entre los dispositivos causó ligeras variaciones en la comunicación de los nodos. Por otro lado, la creación del ataque *wormhole* no fue un trabajo sencillo, ya que retrasmitir los paquetes al otro lado del canal, se deben copiar en el protocolo 802,11 y encapsularlos al protocolo 802,3 para enviarlos por el protocolo *ethernet*. Además, el protocolo BATMAN solo entabla comunicación con nodos propios del protocolo y no con terceros, es por ello que se tomó la decisión de que los nodos que forman parte del *wormhole* también formaran parte de la red, creando así un ataque en el que un par de nodos que forman parte de la red se ven comprometidos y modificados para realizar un ataque dentro de la misma red.

Visto desde el lado del atacante se puede concluir que llevar a cabo este tipo de ataque no resulta en una tarea fácil, esto se debe a que primero se debe falsificar las identidades y entrar como parte de la red, y adicionalmente, infiltrarse en el espacio de pruebas para tender un cable de extremo a extremo entre los nodos. Sin embargo, aunque la implementación del ataque sea complicada no descarta la posibilidad de que este exista.

Dicho lo anterior, a pesar de todas las consideraciones que se deben tomar en cuenta para llevar a cabo la implementación física de la red y de los algoritmos propuestos para contrarrestar el ataque *wormhole*, se considera que el porcentaje de pruebas exitosas en las que fue posible detectar el ataque en la red física es aceptable frente al porcentaje de pruebas exitosas en las que fue posible detectar el ataque en la red simulada [1]. Reafirmando así la idea de que los escenarios físicos comprenden de mayores variaciones que los escenarios teóricos o de simulación.

6.2. Perspectivas de investigación

Se propone la realización de nuevos escenarios contando con una mayor cantidad de nodos (dispositivos Raspberry Pi, baterías y displays) para lograr crecer la red de sensores, y de este modo monitorizar su comportamiento con y sin la presencia de un ataque *wormhole*, además de la creación de un ataque de este tipo con un canal de longitud mayor al realizado en esta tesis. Con esto será posible analizar el comportamiento de la red bajo la presencia del ataque con diferentes variaciones tanto de nodos como de longitud del ataque y poder obtener un porcentaje de efectividad de esta propuesta en diferentes escenarios físicos.

Apéndice A

Pseudocódigo generación de Spanning Tree

```
Inicio:
    Bandera STOP = False, (detener flooding de Beacons)
    Un nodo Root, ultimo en iniciar.
    Regla de envio de mensaje multicast:
      Si nodo es root entonces: ENVIAR(true,conjuntoB)
      Si nodo es beacon o node entonces: ENVIAR(false,conjuntoB)

    Una vez iniciada la red, inicia el flooding de ubicacion Beacons:

        [Como beacon] Si el contador de mensajes count = 0 entonces:
            Enviamos mensaje de ubicacion del beacon con la informacion del
            nodo ip origen,mac_batman,posicion.
            Enviar(IP ORIGEN, MAC BATMAN, POSICION):
            count + 1

    Comenzamos a escuchar mensajes(m) dentro de la red, m = RECIBIR()

      Si longitud de mensaje m es igual a 2 entonces:
            Comienza el envio de conjunto B para ST.
                Si mensaje m es de vecino a UN salto entonces:
                    Propagamos flooding de conjunto B, con multicast de
                    acuerdo a regla.

      Si longitud de mensaje m es igual a 1 entonces:
            Validacion de banderas y reconstruccion de ST.
            Dejamos de propagar mensajes de ubicacion. Bandera STOP = True
            Iniciamos reconstruccion hacia arriba/atras de ST
        Si mensaje m contiene la bandera REBUILD entonces:
            Guardamos informacion de reconstruccion de ST.
            Guardar(SPANNING TREE).
            Propagamos informacion local de reconstruccion a nodos vecinos.
            EnviarRebuild(IP ORIGEN, MAC BATMAN, SPANNING TREE).

        Eliminacion de aristas entre los nodos ocasionadas por dobles padres.
        Si mensaje m contiene la bandera DELETE entonces:
            Se elimina las aristas entre los nodos.
            Si se recibe mensaje m con un segundo padre entonces:
                Eliminar arista con el segundo padre.
```

```
            Sino:
                Ignorar mensaje de eliminacion


Si bandera STOP es True :
  Tomamos informacion del origen del mensaje m
  Si el origen es diferente a mi ip:
      Si esta registrado este origen entonces:
          Inundamos la red con mensaje m. Enviar(IP ORIGEN, MAC BATMAN)
      Sino:
          Registramos informacion de mensaje m
          Inundamos la red con mensaje m. Enviar(IP ORIGEN, MAC BATMAN)
```

Apéndice B

Código principal para nodo root en Python versión 2.7

```
# ######################################################################
#
# This program allows the root node to distribute its location
# information, measure the size of the Hop-size, as well as rebuil
# the Spannig Tree and check intersections.
# A. Hernandez, B. Jaimes, I. Hernandez
#
# ######################################################################


import socket
import pickle
import time
import subprocess
import spaTree

from get_data import get_data
from calDist import calcularAHD
from intersect import  existIntersection

reg = {}

#Node data
data = open('ip.txt','r')
l = []

for line in data:
  l.append(line.rstrip('\n'))
data.close()

ip_device = l[0]
mac_bat = l[1]
ip_broadcast = l[2]
location = l[3]

dest = (ip_broadcast,12345)
HOST = ip_broadcast
UDPSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)#socket to send
```

```
39 UDPSock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
40 mySock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #socket to receive
41 mySock.bind((HOST,12345))
42
43
44 try:
45   print('Start ROOT')
46   count = 0
47   rebuild = 0
48   myST = []
49   temp = {}
50   stop = False # Para detener el flooding
51   B = []
52   padre = True
53   datos, padre  = spaTree.envMultic(True,B)
54   msg = pickle.dumps(datos)
55   UDPSock.sendto(msg,dest)
56
57   while True:
58
59     if count == 0: #number of messages allowed in flooding
60       dic = {'origin': ip_device, 'mac_bat': mac_bat, 'location': location}
61       d = pickle.dumps(dic)
62       UDPSock.sendto(d, dest)
63       print('Beacon Msg sent')
64       time.sleep(0.5)
65       count = 1
66
67     #------Listening to the other beacon nodes----------
68     msg = mySock.recvfrom(1024)
69     d = pickle.loads(msg[0])
70
71     if len(d) == 2:
72       #pass
73       arr = d[0].values()[0]
74       if spaTree.revisar(arr):
75         resultado = spaTree.envMultic(False,d)
76         if len(resultado[0].values()[0]) != 0:
77           print("se enviara el resultado")
78           msg = pickle.dumps(resultado)
79           UDPSock.sendto(msg,dest)
80
81     elif len(d) == 1:
82       stop = True
83       if d.get('REBUILD'):
84         print("Reconstruyendo ST")
85         array = d.get('REBUILD')
86
87         if mac_bat == array[0]:
88           datos = pickle.loads(spaTree.leerB())
89           if rebuild < len(datos[0].values()[0]):
90             if temp.has_key(array[1]) != True:
91               temp[array[1]] = array[2]
92               print(temp)
93               rebuild += 1
94             if rebuild == len(datos[0].values()[0]):
95               myST.append(temp.copy())
96           if rebuild == len(datos[0].values()[0]):
97             aviso = {}
98             aviso['REBUILD'] = datos[1], mac_bat, myST
```

```
99              #print(aviso)
100             print("ST completo!")
101             spaTree.guardaRST(aviso)
102             print(aviso)
103
104     elif stop is not True:
105       d = pickle.loads(msg[0])
106       origin_IG = d.get('origin')
107       origin_ID = d.get('mac_bat')
108
109       if origin_IG != ip_device:
110         if origin_ID in reg.keys():
111           newMsg = pickle.dumps(d)
112           UDPSock.sendto(newMsg, dest)
113           #print("Broadcast_node_B")
114           time.sleep(0.5)
115           #-------------------------------------
116         else:
117           print('Registro_' + origin_ID)
118           d_copy = d.copy()
119           reg[origin_ID] = d_copy
120           newMsg = pickle.dumps(d)
121           UDPSock.sendto(newMsg, dest)
122           print("Broadcast_node_B")
123           time.sleep(0.5)
124       d.clear()
125
126 except KeyboardInterrupt:
127
128   mySock.close()
129   print('Finish . . . ')
130   beacons = []
131
132   for i in reg.keys():
133
134     beacon_to_tr = str(i) #beacon to search
135     comm = 'sudo batctl tr ' + beacon_to_tr
136     text = subprocess.check_output(comm,shell = True).splitlines()
137     hops = 0
138
139     for x in range(len(text)):
140       if x != 0:
141         if text[x][4:21].find('*') == -1:
142           if text[x][4:21] not in worm:
143             hops += 1
144
145
146     beacons.append(' ' + reg.get(i)['origin'] + 7*(' ')  + reg.get(i)['
      mac_bat'] + 6*(' ') + str(reg.get(i)['location']) + 10*(' ') + str(hops) +
       ('\n'))
147     name_file = 'beaconsB.txt' #File to store distance and location data of
      beacons
148     registro_beacons = open(name_file,'wb')
149     registro_beacons.write( 80 *('-') + ('\n') + 4*('\t') + 'Beacons record \
      n' + 80 *('-') + ('\n'))
150     registro_beacons.write((' Beacon\'s IP') + 7 * (' ') + (' Beacon\'s
      BATMAN' ) + 8 * (' ') + ('Beacon\'s Location')+ 5 * (' ') + ('Me to Beacon
       \n\n'))
151
152     for i in beacons:
```

```
153       registro_beacons.write(i+'\n')
154
155     registro_beacons.close()
156
157 finally:
158
159   dic_data = get_data(name_file)
160   print('dic_data: ')
161   print(dic_data)
162
163   #----Take Hop Size ------------------
164   coord = {}
165   hops = {}
166   dist = {}
167
168   for clave in dic_data.keys():
169     coord[clave] = dic_data[clave]['location']
170     hops[clave] = dic_data[clave]['hops-count']
171
172   misCoord = list(map(float,(location).split(',')))
173   AHD=calcularAHD(coord,hops,misCoord)  #Hop Size
174   print('AHD = ' + str(AHD))
175     ##----Hop size flooding------------------------
176   dist['mac_bat'] = mac_bat
177   dist['origin'] = ip_device
178   dist['Distancia'] = AHD
179   cont = 0
180
181   while cont < 1:
182     envDist = pickle.dumps(dist)
183     UDPSock.sendto(envDist,dest)
184     print('Sent hop size')
185     time.sleep(3.0)
186     cont += 1
187
188   UDPSock.close()
189
190    ###---Intersection checking------
191   dic_inter = {}
192   dic_temp = {}
193   dic_inter = dic_data.copy()
194
195   dic_temp = {'location' : misCoord, 'hops-count' : 0 }
196   dic_inter[mac_bat[12:]] = dic_temp.copy()
197   dic_temp.clear()
198   print('dic_inter: ')
199   print(dic_inter)
200
201   '''
202   if existIntersection(dic_inter, AHD) >= 3:
203     print("Existen al menos 3 intersecciones")
204   else:
205     print("No hay intersecciones suficientes") '''
```

Apéndice C

Código principal para nodo en Python versión 2.7

```
# ######################################################################
#
# this program allows the nodes to receive messages from the becons,
# store this information, flood the network with this information
# and send their information to rebuild ST.
# A. Hernandez, B. Jaimes, I. Hernandez
#
# ######################################################################

import socket
import pickle
import time
import subprocess
import spaTree


reg = {}

#Node data
data = open('ip.txt','r')
l = []

for line in data:
  l.append(line.rstrip('\n'))
data.close()

ip_device = l[0]
mac_bat = l[1]
ip_broadcast = l[2]
location = l[3]

dest = (ip_broadcast,12345)
HOST = ip_broadcast
UDPSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)#socket to send
UDPSock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
mySock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)#socket to receive
mySock.bind((HOST,12345))

```

```
try:
  print('Start Node')
  count = 0
  rebuild = 0
  myST = []
  temp = {}
  stop = False # Para detener el flooding

  while True:

  #------Listening to the other beacon nodes and other messages----------
    msg = mySock.recvfrom(1024)
    d = pickle.loads(msg[0])

    #Differentiate between Spanning Tree and other messages
    if len(d) == 2:
      #pass
      arr = d[0].values()[0]
      padre = True #nuevo
      if spaTree.revisar(arr):
        resultado, padre = spaTree.envMultic(False,d)
        if len(resultado[0].values()[0]) != 0 and padre == True:
          print("se enviara el resultado")
          msg = pickle.dumps(resultado)
          UDPSock.sendto(msg,dest)
        elif padre != True:
          print("Ya tengo un padre")
          aviso = {}
          datos = pickle.loads(spaTree.leerB())
          padre = datos[1]
          aviso['DELETE'] = padre, mac_bat
          #print(aviso)
          msg = pickle.dumps(aviso)
          UDPSock.sendto(msg,dest)
          print("Envie el aviso")
          time.sleep(5)

        datos = pickle.loads(spaTree.leerB())
        if len(datos[0].values()[0]) == 0:
          time.sleep(10)
          aviso = {}
          # Aviso [Accion] = Padre, Mi MAC, B
          aviso['REBUILD'] = datos[1], mac_bat, datos[0].values()[0]
          #print(aviso)
          spaTree.guardaRST(aviso)
          print(aviso)
          while True:
            msg = pickle.dumps(aviso)
            UDPSock.sendto(msg,dest)
            print("Envie el aviso de reconstruccion B vacio")
            time.sleep(15)

    elif len(d) == 1:
      print("Entre al elif")
      stop = True
      datos = pickle.loads(spaTree.leerB())

      if len(datos[0].values()[0]) == 0:
        time.sleep(10)
        aviso = {}
```

```
99          # Aviso [Accion] = Padre, Mi MAC, B
100         aviso['REBUILD'] = datos[1], mac_bat, datos[0].values()[0]
101         #print(aviso)
102         spaTree.guardaRST(aviso)
103         print(aviso)
104         while True:
105           msg = pickle.dumps(aviso)
106           UDPSock.sendto(msg,dest)
107           print("Envie el aviso")
108           time.sleep(20)
109
110       if d.get('DELETE'):
111         array = d.get('DELETE')
112         print(array)
113         if mac_bat != array[0]:
114           print(datos)
115           b = datos[0].values()[0]
116           print(b)
117           if array[1] in b:
118             datos[0].values()[0].remove(array[1]) #eliminar de la lista
119             spaTree.reescribir(datos)
120           else:
121             print("Mensaje de eliminacion ignorado")
122         else:
123           print("Yo soy el padre")
124
125       elif d.get('REBUILD'):
126         print("Reconstruyendo ST")
127         array = d.get('REBUILD')
128
129         if mac_bat == array[0]:
130           if rebuild < len(datos[0].values()[0]):
131             if temp.has_key(array[1]) != True:
132               temp[array[1]] = array[2]
133               rebuild += 1
134             if rebuild == len(datos[0].values()[0]):
135               myST.append(temp.copy())
136               print("Se agrego informacion a mi ST")
137       if rebuild == len(datos[0].values()[0]) or len(datos[0].values()[0])
    == 0:
138         aviso = {}
139         aviso['REBUILD'] = datos[1], mac_bat, myST
140         #print(aviso)
141         spaTree.guardaRST(aviso)
142         print(aviso)
143         while True:
144           msg = pickle.dumps(aviso)
145           UDPSock.sendto(msg,dest)
146           print("Envie el aviso de reconstruccion")
147           time.sleep(15)
148
149     elif stop is not True:
150       d = pickle.loads(msg[0])
151       origin_IG = d.get('origin')
152       origin_ID = d.get('mac_bat')
153
154       if origin_IG != ip_device:
155         if origin_ID in reg.keys():
156           if 'Distancia' in d: #Message with hop size received
157             distancia = d.get('Distancia')
```

```
158             reg[origin_ID]['Distancia'] = distancia
159             newMsg = pickle.dumps(d)
160             UDPSock.sendto(newMsg, dest)
161             print("Broadcast_node_Distancia")
162             time.sleep(0.5)
163             print('Distancia ok')
164             raise KeyboardInterrupt #Having the distance, we end up
165
166         else:
167           print('Registro_' + origin_ID)
168           d_copy = d.copy()
169           reg[origin_ID] = d_copy
170
171       #-----Flooding the network with incoming messages-----
172       newMsg = pickle.dumps(d)
173       UDPSock.sendto(newMsg, dest)
174       time.sleep(0.5)
175       d.clear()
176
177 except KeyboardInterrupt:
178
179   mySock.close()
180   UDPSock.close()
181   print('Finish . . . ')
182   beacons = []
183
184   for i in reg.keys():
185
186     beacon_to_tr = str(i) #beacon to search
187     comm = 'sudo batctl tr ' + beacon_to_tr
188     text = subprocess.check_output(comm,shell = True).splitlines()
189     hops = 0
190
191     for x in range(len(text)):
192       if x != 0:
193         if text[x][4:21].find('*') == -1:
194           if text[x][4:21] not in worm:
195             hops += 1
196
197     beacons.append(' ' + reg.get(i)['origin'] + 7*(' ')  + reg.get(i)['
      mac_bat'] + 6*(' ') + str(reg.get(i)['location']) + 10*(' ') + str(hops) +
       ('\n'))
198
199     if 'Distancia' in reg.get(i):
200       AHD = reg.get(i)['Distancia'] #we recover the hop size
201
202     name_file = 'beaconsN.txt' #a#File to store distance and location data of
       beacons
203     registro_beacons = open(name_file,'wb')
204     registro_beacons.write( 80 *('-') + ('\n') + 3*('\t') + 'Beacons record \
      n' + 80 *('-') + ('\n'))
205     registro_beacons.write((' Beacon\'s IP') + 7 * (' ') + (' Beacon\'s
      BATMAN' ) + 8 * (' ') + ('Beacon\'s Location')+ 5 * (' ') + ('Me to Beacon
       \n\n'))
206     for i in beacons:
207       registro_beacons.write(i+'\n')
208
209     registro_beacons.close()
210
211 finally:
```

```
212
213   dic_data = get_data(name_file)
214   print(dic_data)
215   print(AHD) #Hop size
```

Apéndice D

Código principal para el ataque wormhole

```
# ######################################################################
#
# This program allows the attack member devices to exchange messages
# using the 802.11 protocol and distribute them to the network using
# the 802.3 protocol.
# A. Hernandez, B. Jaimes, I. Hernandez
#
# ######################################################################

import socket
import pickle
import threading
import time

#---Attack ip's
    ----------------------------------------------------------------------
me = '192.168.5.2'
partner = '192.168.5.3'
#
    ------------------------------------------------------------------------------------------


#--Sockets for B.A.T.M.A.N (802.11)
    ------------------------------------------------
ip_broadcast = '192.168.5.255'
destBatman = (ip_broadcast,12345)
HOST = ip_broadcast
sendBatman = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #socket to send
sendBatman.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
recvBatman = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #socket to
    receive
recvBatman.bind((HOST,12345))
#
    ------------------------------------------------------------------------------------------


#--Sockets for Ethetnet (802.3)
    ------------------------------------------------------
```

```
31 destEthernet = ('192.168.2.3',23456)
32 sendEthernet = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #socket to
       send
33 sendEthernet.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
34 recvEthernet = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #socket to
       receive
35 recvEthernet.bind(('192.168.2.1',23456))
36 #
       ------------------------------------------------------------------------------------


37
38 def funRecvEthernet():
39
40     print('Receiving from Ethernet')
41     while True:
42         msg = recvEthernet.recvfrom(40000)
43         print('eth: ' + str(pickle.loads(msg[0])))
44         sendBatman.sendto(msg[0],destBatman)
45         print(' ---> sent by B.A.T.M.A.N <---')
46
47 try:
48     print('Wormhole Attack begins...')
49     thread = threading.Thread(target = funRecvEthernet)
50     thread.setDaemon(True)
51     thread.start()
52     print('Receiving from B.A.T.M.A.N')
53
54     while True:
55
56         msg = recvBatman.recvfrom(1024)
57         if (me != msg[1][0] and partner != msg[1][0]):
58             print('bat: ' + str(pickle.loads(msg[0])))
59             sendEthernet.sendto(msg[0],destEthernet)
60             print(' ---> sent by Ethernet <---')
61
62 except KeyboardInterrupt:
63     print('Attack finished')
```

Apéndice E

Configuración B.A.T.M.A.N

En este apartado mostraremos los pasos a seguir para configurar las tarjetas rapsberry en una red AD-HOC de B.A.T.M.A.N. Para ello es necesario tener que dicha tarjeta cuente con conexión a internet.

Para poder utilizar este protocolo en el nuestros dispositivos es necesario descargar el *batctl*, que es el controlador y diagnostico de *batman-adv* proporcionado por Open-Mesh. Para poder instalarlo se necesitan dos librerias previas para el correcto funcionamiento, las cuales podemos instalar con el comando:

$ sudo apt install libnl-3-dev libnl-genl-3-dev

Luego es necesario descargar, compilar e intastalar el controlador:

$ git clone https://git.open-mesh.org/batctl.git
$ cd batctl
$ sudo make install

Una vez realizado estos pasos se puede genear un archivo con extensión *.sh* el cual se recomienda se inicie al encendar cada una de los dispostivos. Este archivo debe contener la siguiente información:

sudo modprobe batman-adv
sudo ip link set wlan0 down
sudo ifconfig wlan0 mtu 1532
sudo iwconfig wlan0 mode ad-hoc
sudo iwconfig wlan0 essid my-mesh-network
sudo iwconfig wlan0 ap any
sudo iwconfig wlan0 channel 8
sleep 1s
sudo ip link set wlan0 up
sleep 1s
sudo batctl if add wlan0
sleep 1s
sudo ifconfig bat0 up

sleep 5s
sudo ifconfig bat0 192.168.5.1/24

Es importante recalcar que cada dispotivo debe tener una dirección IP diferente pero dentro de la misma categoria así con el mismo *channel* y *essid*.

Algunos de los comandos que se pueden ejecutar con este controlador son:

Comando	Descripción
neighbors | n	Tabla de depuración que muestra cuáles son los nodos vecinos que los mensajes originadores registraron como activos, con posible conexión y tiempo desde la última vez que se vio.
originators | o	Tabla de depuración de los mensajes originadores registrados en cierto tiempo así como el origen de éste.
traceroute | tr [MAC_address | bathost_name]	Tabla de depuración que contiene la ruta de nodos por los cuáles debe pasar un nodo A (el que lanza el traceroute) a un nodo B (MAC_address | bathost_name).
tcpdump | td [-c][-p filter][-x filter] interface	Se mostrará una tabla de depuración con todos los paquetes que se ven en la(s) interfaz(es) dada(s). Para imprimir sólo los paquetes que coincidan con el número de compatibilidad de batctl especifique la opción c"(compat filter). Para filtrar los tipos de paquetes mostrados puede utilizar p"(mostrar sólo los tipos de paquetes especificados) o x"(mostrar todos los tipos de paquetes excepto los especificados).
ping | p	realiza una función similar al comando ping conocido, solo modificado para el protocolo B.A.T.M.A.N
multicast_mode | mm [0 | 1]	Se puede realizar el multicast con diferentes configuraciones, si no se da una configuración se realiza el flooding de la manera clásica.
interface | if [add | del iface(s)]	Mostrará una tabla con la configuración actual de las interfaces. Para añadir o eliminar interfaces, especifique "add" o "del" como primer argumento y añada los nombres de las interfaces que desea añadir o eliminar. Se pueden especificar múltiples interfaces.
orig_interval | it [interval]	Muestra el ajuste actual del intervalo de mensajes originadores; de lo contrario, el parámetro se utiliza para establecer el intervalo del emisor. El intervalo está en unidades de milisegundos.

Tabla E.1: Comandos controlador *batctl*

Bibliografía

[1] K. Galvan and F. Garcia *Algoritmo para la detección del ataque wormhole en una red de sensores a partir de características topológicas*, Tesis UNAM, 2020, 132.248.9.195/ptd2020/diciembre/0806000/Index.html.

[2] Johan S., R. Rueda ,M. Jesús and P. Talavera, *Similarities and differences between Wireless Sensor Networks and the Internet of Things: Towards a clarifying position*, Revista Colombiana de Computación, 2017, Vol. 18, No. 2, pp. 58–74.

[3] Papadopoulos, K., T. Zahariadis, N. Leligou and S. Voliotis., *Sensor Networks Security Issues In Augmented Home Environment*, IEEE International Symposium on Consumer Electronics, April 2008.

[4] Zia, T. and A. Zomaya., *Security Issues in Wireless Sensor Networks*, International Conference on Systems and Networks Communications, October 2006.

[5] Areitio Bertolín Javier, *Análisis de riesgos y contramedidas en REDES MANET*, REE, Febrero 2012. pp. 62-73

[6] Tran,P. V., L. X. Hung, Y. Lee, S. Lee and H. Lee, *TTM: An Efficient Mechanism to Detect Wormhole Attacks in Wireless Ad-hoc Networks*, Consumer Communications and Networking Conference, 2007, DOI: 10.1109/CCNC.2007.122

[7] Maheshwari, R, J. Gao and S. R. Das, *Detecting Wormhole Attacks in Wireless Networks Using Connectivity Information*, IEEE International Conference on Computer Communications, 2007, DOI: 10.1109/INFCOM.2007.21

[8] Eriksson, S.,V. Krishnamurthy and M. Faloutsos, *TrueLink: A Practical Countermeasure to the Wormhole Attack in Wireless Networks*, International Conference on Network Protocols, 2006, DOI:10.1109/ICNP.2006.320200

[9] Dong, D., M. Li, Y. Liu and X. Liao, *WormCircle: Connectivity-Based Wormhole Detection in Wireless Ad Hoc and Sensor Networks*, International Conference on Parallel and Distributed Systems, 2009, DOI: 0.1109/ICPADS.2009.97

[10] Nabila, L., M. Gueroui and M. Aliouat, *Secure DV-Hop localization scheme against wormhole attacks in wireless sensor networks*,December 2011, DOI: 10.1002/ett.1532.

[11] Zhu Bin, Liao Junguo and Zhang Huifu, *Defending Wormhole Attack in APS DVhop*, IEEE Third International Conference on Communications and Networking in China, Aug. 2008. DOI: 10.1109/CHINACOM.2008.4685007

[12] Jianpo Li and Dong Wang, *The Security DV-Hop Algorithm against Multiple-Wormhole-Node-Link in WSN*, April 2019, KSII Transactions on internet and information systems Vol. 13, No. 4.

[13] Ronghui, He, Ma Guoqing, Fang Lan, Kuang Chunguang and Liu Li, *WRL:A Wormhole-Resistent Localization Scheme Based on DV-hop for Wireless Sensor Networks*, January 2010.

[14] Harsányi, K., A. Kiss and T. Szirányi, *Wormhole detection in wireless sensor networks using spanning trees*, IEEE International Conference on Future IoT Technologies (Future IoT), 2018, DOI: 10.1109/FIOT.2018.8325596

[15] Junfeng Wu, Honglong Chen,Wei Lou, Zhibo Wang, and Zhi Wang, *Label-Based DV-Hop Localization Against Wormhole Attacks in Wireless Sensor Networks*, Fifth IEEE International Conference on Networking, Architecture, and Storage, 2010,DOI 10.1109/NAS.2010.41.

[16] Jianpo Li, Dong Wang and Yanjiao Wang, *Security DV-hop localisation algorithm against wormhole attack in wireless sensor network*,IET Wireless Sensor Systems, June 2017, DOI:10.1049/iet-wss.2017.0075.

[17] Liu, X., C. Liu, W. Liu, Y. Zhang and X. Zeng, *Research on wireless sensor network node localization algorithm*,2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST), Shenzhen, 2017, pp. 170-173, doi: 10.1109/ICFST.2017.8210496.

[18] Liu, B. , Z. Wang, J. Cheng, L. Yu and Y. Chen, *Research on the node location algorithm in the wireless sensor network*, 2011 International Conference on Electrical and Control Engineering, Yichang, 2011, pp. 350-353, doi: 10.1109/ICECENG. 2011.6057023.

[19] Liu, J. , Z. Wu and Z. Yin, *An improved location algorithm in Wireless Sensor Network*,2013 International Conference on Optoelectronics and Microelectronics (ICOM), Harbin, 2013, pp. 255-258, doi: 10.1109/ICoOM. 2013.6626543.

[20] Shi, X. , J. Su, Z. Ye, F. Chen, P. Zhang and F. Lang,, *A Wireless Sensor Network Node Location Method Based on Salp Swarm Algorithm*, 2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Metz, France, 2019, pp. 357-361, doi:10.1109/IDAACS.2019.8924394.

[21] Farmer and Justin R., *A comparison of power harvesting techniques and related energy storage issues*, M.S Thesis. Dept. Mech. Eng., Virginia Polytechnic Institute and State Univ. Blacksburg, VA; 2007

[22] Beedy, Stephen and White, Neil, *Energy Harvesting for Autonomous Systems*,Artech House Series: 201060

[23] Essa Q. Shahra; Tarek R. Sheltami; Elhadi Shakshuk., *A Comparative Study of Range-Free and Range-Based Localization Protocols for Wireless Sensor Network: Using COOJA Simulator* January 2017. International Journal of Distributed Systems and Technologies 8(2017):16. DOI: 10.4018/IJDST.2017010101