UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE POSGRADO EN INGENIERÍA
MAESTRÍA EN INGENIERÍA ELÉCTRICA
TELECOMUNICACIONES

Algoritmo para la detección del ataque wormhole en una red de sensores a partir de características topológicas.

TESIS

QUE PARA OPTAR POR EL GRADO DE:
MAESTRA EN INGENIERÍA ELÉCTRICA

PRESENTA:
ANAKAREN GALVÁN BENITEZ

TUTOR
DR. LUIS FRNACISCO GARCÍA JIMENEZ
FACULTAD DE INGENIERÍA

Ciudad Universitaria, Cd. Mx. 2021

Agradecimientos

A mi familia, quienes con su cariño y apoyo incondicional me han permitido llegar a cumplir un sueño más. A mis padres especialmente por inculcar en mí el ejemplo, todo lo que soy es gracias a ustedes. A mis hermanos por creer en mí y estar siempre que los necesito. A mis abuelos que también fueron pilares en mi formación y aunque ya no estén, estoy segura de que se hubieran sentido muy orgullosos de mí.

A Gerardo por confiar en mí y acompañarme a lo largo de esta etapa, dándome consejos y palabras de aliento.

De la misma manera quiero expresar mi más grande y sincero agradecimiento al Dr. Francisco, quien con su dirección y conocimiento hizo posible esta tesis.

Agradezco a la Universidad Nacional Autónoma de México por abrirme las puertas una vez más y ser mi segundo hogar.

Finalmente doy gracias al apoyo brindado por parte del proyecto DGAPA-PAPIIT IA105520.

Índice general

Índice de figuras

Índice de tablas

Resumen

Las redes de sensores han incrementando su popularidad día con día, ya que juegan un papel muy importante en muchas aplicaciones, tales como la monitorización ambiental, detección de incendios, seguridad, diagnóstico de enfermedades en el cuerpo humano, aplicaciones militares y la protección de vida silvestre. Estas redes están formadas por un grupo de dispositivos distribuidos físicamente en un área geográfica sin un coordinador preestablecido, lo que las obliga a sentar mecanismos locales que hagan posible la comunicación. Uno de los aspectos más importantes para tomar en cuenta en este tipo de redes es la protección de la información que es transmitida entre los dispositivos, ya que si un conjunto de nodos maliciosos interceptara la información, éstos pueden ganar algún beneficio. Hoy en día se conocen una gran variedad de ataques que pueden afectar el correcto funcionamiento de una red de sensores, y como consecuencia deteriorar su rendimiento y sus recursos. Algunos de los ataques más conocidos en las redes de sensores son *black* y *gray hole*, en los cuales un nodo malicioso captura el tráfico que pasa por él con el fin de que no llegue a su destino, es decir, logra atraer hacia sí mismo todos los paquetes de sus vecinos eliminándolos total o parcialmente. El ataque *sinkhole*, por otro lado, daña la estructura de la red, ya que el nodo malicioso hace creer a sus nodos vecinos que han encontrado una mejor ruta a un dispositivo y estos envían su tráfico a través de él. Sin embargo, esta tesis se enfoca en el ataque *wormhole*, en el cual dos o más nodos maliciosos mediante un par de antenas crean túneles de alta velocidad por donde retransmiten lo escuchado, en otras palabras, reenvían paquetes de una antena a la otra, considerando un canal bidireccional. Esto afecta a los protocolos de localización, así como a lo protocolos de encaminamiento, por lo que este ataque es muy dañino en la topología de la red, y como consecuencia los protocolos que están en capas superiores no trabajarán correctamente. Actualmente existen diferentes propuestas para poder detectar el ataque *wormhole*, algunas están basadas en la sincronización de relojes y hardware especializado, lo que incrementa el costo de los dispositivos. Otras se basan en *overhearing* lo que afecta a la red debido al consumo excesivo de recursos. Algunas más se basan en detectar longitudes de las rutas más cortas y en estructuras anormales. No obstante, suelen requerir el conocimiento de al menos 4 o 5-vecindario, lo que decrece completamente la eficiencia de la red por la administración y el procesamiento de la información. Es por ello, que en esta tesis se propone un algoritmo para la detección de un ataque *wormhole* con ayuda de características topológicas, que solo requiere el conocimiento del 1-vecindario y algunos cuantos nodos con hardware especializado, específicamente deben contar con un *sistema de posicionamiento global* (GPS). El al-

goritmo propuesto en esta tesis está formado por tres etapas, la primera etapa consiste en localizar las zonas en las que se puedan encontrar los posibles atacantes mediante el uso del algoritmo distribuido *Distance Vector-Hop (DV-HOP)*. En la segunda etapa se valida si existe una conexión entre estas zonas mediante la creación de un *spanning tree*, verificando que se forme entre ellas un grafo bipartito con el conocimiento del vecindario de los nodos en cuestión; con base en esto, la tercera etapa crea un nuevo *spanning tree* el cual elimina las aristas pertenecientes al grafo bipartito que une las zonas afectadas por el ataque. Finalmente si se desea y como un proceso adicional para verificar la presencia del ataque, se realiza un estudio estadístico respecto a la distancia que presentan los nodos maliciosos antes y después de la eliminación de las aristas. Con esto se busca hacer frente al ataque y recuperar la seguridad de la red.

Capítulo 1

Introducción

Las redes inalámbricas de sensores están ganando terreno muy rápidamente, debido a que su despliegue tiene un bajo costo y un sinfín de aplicaciones, como monitorizar parámetros, tales como temperatura, sonido, vibraciones, ozono, monóxido de carbono y detección de agentes químicos en áreas hostiles; también para dar seguimiento en pacientes y recopilación de datos clínicos para el cuidado de la salud. Dentro del ámbito militar, se pueden establecer en territorios desconocidos para descubrir información que les permita estar prevenidos; en otros casos, se podría tener respuesta rápida ante una emergencia por desastres, como recopilar y difundir la información después de un huracán o terremoto, especialmente cuando se daña la infraestructura para las comunicaciones y es importante restablecerla; asimismo, se usan en la construcción para encontrar daños o puntos críticos al poderse colocar dentro de una estructura. También son usadas en las redes vehiculares para difundir información sobre el tráfico, condiciones ambientales y prevención o advertencia de accidentes; además se considera una de las principales tecnologías para llevar a cabo la implementación de *Internet of Things (IoT)*, con todo esto se busca mejorar la calidad de vida y facilitar diversas tareas.

Las redes de sensores están formadas por dispositivos autónomos, distribuidos a lo largo de un área de interés y cuyo objetivo es trabajar de manera colectiva para recolectar y enviar datos a través de la red, eligiendo una ruta óptima (entre dispositivos) para llegar a su destino. Cada elemento de la red tiene un costo relativamente bajo, adicional a eso, requieren muy poca o ninguna infraestructura física para la transmisión de información entre los nodos de la red, para las transmisiones generalmente requieren de nodos intermediarios que reenvíen la información, lo que ofrece un sistema flexible y fácil de instalar. Una de las características más llamativas es su bajo consumo energético, ya que los dispositivos suelen llevar pequeñas baterías, que pueden recargar mediante tecnologías como *energy harvesting*, prolongando la vida útil y la autonomía de la red.

Sin embargo, se tienen algunas desventajas para estas redes, como la disponibilidad, velocidad o rango de trasmisión limitados, interferencias o pérdida de paquetes, cambios de ruta por la movilidad, recursos reducidos, entre algunas otras; pero la que manifiesta un mayor reto es la seguridad, esto se debe a la vulnerabilidad en el medio de transmisión facilitando la escucha clandestina y el ser víctima de diferentes

ataques, tanto activos como pasivos, y es imposible utilizar los mecanismos convencionales, debido a los recursos limitados que presentan los dispositivos en cuanto a su alcance de transmisión, procesamiento, capacidad de almacenamiento y suministro de energía [1]. Por lo que el desarrollo de estudios que permitan incrementar la seguridad se convierte en un punto esencial dentro de esta área.

Algunos de los ataques más relevantes que se han identificado en la literatura son el espionaje mediante un nodo malicioso, alteración de parámetros vitales para el funcionamiento de la red, inundación de mensajes, análisis de tráfico, entre otros; específicamente se encuentran [2,3,4]: *negación del servicio (DoS), eavesdropping, SinkHole, Sybil, Blackhole, Greyhole y Wormhole.*

El ataque *DoS* inhabilita la disponibilidad del servicio a nivel físico, capa MAC, capa de enlace de datos o en capa de red, es decir, incapacita una aplicación o un dispositivo, con el objetivo de bloquear el servicio para el que está destinado. *Eavesdropping* es un ataque a nivel red que se encarga de capturar paquetes transmitidos en la red buscando cualquier tipo de información. Por otro lado, el ataque *Sinkhole* es creado por un nodo que se presenta como una buena elección de ruta y los mensajes que pasan por él pueden desecharse o utilizarse para ataques adicionales en los que todo el tráfico de un área en particular se desvía hacia un nodo malicioso. El ataque *Sybil*, esta conformado por un nodo malicioso que adopta múltiples identidades o localizaciones, asemejando que son nodos distintos en la red, esto lo hace para atraer información e interceptarla. En *Blackhole* el nodo desecha todo el tráfico que recibe y si el nodo es también un sumidero, el ataque es mucho más agresivo. En el ataque *Greyhole* solo un tipo concreto de paquetes se desecha por el nodo malicioso. Por otro lado, un ataque particularmente difícil de contrarrestar es el ataque *Wormhole*, este se presenta en tres diferentes maneras, estas incluyen la manipulación de paquetes, reproducción de paquetes de datos con alta potencia y un canal oculto fuera de banda, esta tesis se centra en el último caso, en el cual dos o más nodos maliciosos que se encuentran a cierta distancia pueden conspirar para intercambiar mensajes entre ellos utilizando un medio de alta velocidad, alto ancho de banda OOB (*out-of-bound*) y baja latencia, para vulnerar la seguridad de la red e interceptar información, ya que las transmisiones inalámbricas se graban en un lado de la conexión y se reproducen en el otro, creando un enlace virtual bajo el control del atacante, lo que significa que los nodos atacados pueden recibir información de otros nodos atacados, aunque seguramente no la hayan recibido anteriormente. Este ataque atrae el flujo de información de los nodos legítimos, dándoles la impresión de que es 1-vecino de un nodo que en realidad está a varios saltos de él [4], con esto se alteran las tablas de encaminamiento y los paquetes se desechan al alcanzar el tiempo que tienen para ser entregados (*TTL (time to live)*). Éste ataque se puede dividir en dos fases: en la primera, los nodos maliciosos intentan atraer a nodos legítimos para enviar datos a otros nodos a través de ellos; en la segunda fase, los nodos maliciosos podrían explotar los datos en una variedad de formas, como intentar romper la clave de cifrado o modificar y descartar paquetes [5]. Esto hace que el ataque *Wormhole* sirva como un trampolín para muchos otros ataques más agresivos y severos [7], además también se podría tener un ataque *Sinkhole* al mismo tiempo, ya que otro nodo malicioso puede artificialmente proveer una "mejor" ruta y de este modo todo el tráfico del área será enviado por él y no por

otras rutas menos atractivas, pero más seguras. Debido a que el ataque *Wormhole* no afecta drásticamente la comunicación se vuelve complicado detectarlo, pero puede dañar la estructura de rutas, interferir en la transmisión de datos y especialmente afectar a los algoritmos de localización.

En la figura 1.1 se ejemplifica el ataque wormhole. En esta red de sensores los nodos maliciosos son A y B, entre ellos se forma un canal para intercambiar información, mostrado por la línea punteada en color rojo. Los nodos 1,2,3 y 4 son 1-vecino del nodo A y los nodos 5,6,7,8 y 9 son 1-vecino del nodo B, esto quiere decir que se encuentran a un salto respectivamente, debido a la implementación del ataque los nodos 1,2,3 y 4, pueden ver al nodo B como si estuviera a dos saltos de ellos y a los vecinos a un salto de B como 3-vecindario, cuando en realidad se encuentran más lejanos, los nodos 5,6,7,8 y 9 tienen el mismo comportamiento con A y su 1-vecindario. Por ese motivo las tablas de encaminamiento de los nodos 1,2,3,4,5,6,7,8 y 9 se verán alteradas.

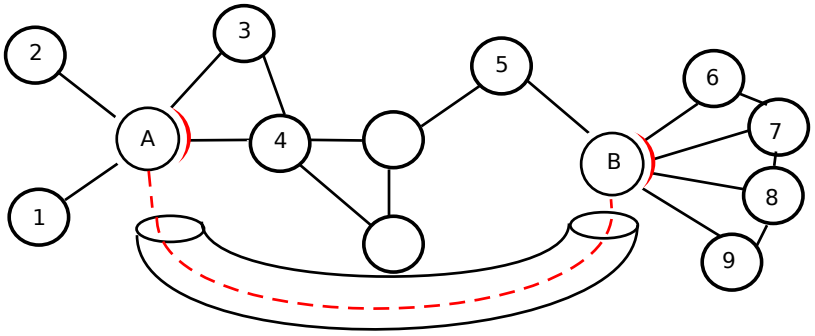


Figura 1.1: Ataque Wormhole.

1.1. Definición del problema

El ataque *Wormhole* no tiene ningún efecto sobre la integridad de la comunicación, lo que lo hace difícil de detectar. Además, puede dañar la estructura de rutas, interferir en la transmisión de datos y especialmente provocar fallas en los algoritmos de localización. Aunque se han propuesto varias formas de contrarrestarlo, muchas de ellas requieren el conocimiento de la topología de la red, ya sea parcial o totalmente. En otras se requiere de hardware muy especializado como relojes precisos o antenas direccionales que elevan el costo de los dispositivos y van en contra del consumo energético típico de estas redes. Otras propuestas presuponen que todos los nodos tienen el mismo radio de comunicación o que se conoce un umbral que les permite saber si existe o no el ataque, sin embargo este umbral debe ser previamente calculado y cambiará para diferentes topologías de red. Otros trabajos suponen un despliegue homogéneo en los sensores, por lo que no funcionarían para topologías no homogéneas. También existen propuestas que utilizan el algoritmo *DV-Hop* como base [10,11,12,13,14,15]. Sin embargo, en esta tesis se propone un mecanismo basado en dos algoritmos muy competentes *DV-Hop* y *spanning tree*, con los cuales se afronta el ataque sin afectar el rendimiento de la red, y en el que solo se requiere conocer el 1-vecindario de cada nodo.

1.2. Hipótesis

El uso de características topológicas combinado con hardware de posicionamiento global permite detectar y eliminar el ataque *wormhole* conociendo únicamente el 1-vecindario.

1.3. Aproximación al problema

Una de las amenazas más peligrosas para las redes de sensores es el ataque *wormhole*, esto se debe a la capacidad que presenta para manipular datos en tiempo real y causar daños importantes en la red. Como contramedida a estos efectos, en esta tesis se utilizan dos algoritmos principales *DV-Hop* y *spanning tree*, además de que el mecanismo propuesto está dividido en 3 fases y una adicional para reafirmar la presencia del ataque; en la primera fase se seleccionan a los posibles nodos atacantes con ayuda del algoritmo *DV-Hop*, en la segunda fase el nodo que inicia el *spanning tree* solicita a los nodos dentro de las zonas de peligro anunciar su 1-vecindario para validar si existe una unión entre ellos, dentro de la tercera fase el mismo nodo que inicializa el *spanning tree* crea un segundo árbol que permite suprimir el ataque. Finalmente, si se quisiera ratificar la existencia del ataque, se puede realizar un estudio estadístico basado en la media y la desviación estándar de saltos entre los nodos atacados.

1.4. Metas

1.4.1. Meta general

Proponer un algoritmo capaz de detectar un ataque *wormhole* en una red de sensores haciendo uso de estudios topológicos, estadísticos y hardware adicional poco costoso (*GPS*), sin la necesidad de implementar algoritmos adicionales que provoquen un alto consumo de recursos y procesamiento.

1.4.2. Metas particulares

- Validar el funcionamiento del algoritmo.
- Crear múltiples escenarios mediante un simulador propio programado en el lenguaje de alto nivel Python, y observar el comportamiento de la red con y sin la presencia del ataque *wormhole*.
- Estudiar múltiples escenarios y compararlos para concluir la eficacia del algoritmo.

1.5. Metodología

Para lograr el objetivo principal de este trabajo se toman en cuenta dos etapas. En la primera etapa, a partir del análisis de los algoritmos utilizados (*DV-Hop* y *spanning tree*), se realizan pruebas tomando en cuenta varios escenarios, además se realizan cálculos estadísticos para corroborar la existencia del ataque, todo esto de forma teórica. En la segunda etapa se realizan simulaciones en el lenguaje interpretado de alto nivel Python con el fin de localizar a los posibles nodos maliciosos y las conexiones entre ellos, para poder eliminarlas y deshacer el canal que forma el ataque *wormhole*, también se obtienen estadísticas antes y después de eliminar el ataque para poder analizar los resultados y concluir la fiabilidad del algoritmo propuesto.

1.6. Contribución

Se propone un algoritmo capaz de detectar y eliminar el ataque *wormhole* en una red de sensores mediante el uso de características topológicas que solo requieren el conocimiento del 1-vecindario.

1.7. Descripción del contenido

Con el objetivo de alcanzar las metas planteadas en este trabajo de investigación, se propone la siguiente estructura:

En el capítulo 2, se presentan los trabajos más relevantes para prevenir el ataque *wormhole*, así como algunas propuestas relacionadas con la utilización del algoritmo Distance Vector-Hop como base para afrontarlo.

En el capítulo 3, se explican los algoritmos propuestos en este trabajo, además se ejemplifican ciertos escenarios y se muestran algunas de las simulaciones de la técnica planteada.

En el capítulo 4, se presenta el análisis de resultados y se muestran algunos cálculos estadísticos para conocer la efectividad del mecanismo propuesto.

En el capítulo 5, se presentan las conclusiones generales, la verificación de la hipótesis y las perspectivas de investigación.

Capítulo 2

Antecedentes

En este capítulo se presentan los trabajos más relevantes para contrarrestar el ataque wormhole de forma general y se resaltan aquellas investigaciones que utilizan el algoritmo Distance Vector-Hop como base para hacer frente a este ataque.

2.1. Estado del arte para el ataque wormhole

Algunas de las propuestas más relevantes para contrarrestar el peligro que genera un ataque wormhole, se presentan en [5,7,9], las cuales demandan un alto grado de procesamiento en la red para poder efectuarse, otras [6,9] trabajan bajo protocolos específicos (AODV, DSR) o es necesario contar con información adicional como el modelo de comunicación, también se puede requerir uso de relojes [6], que no siempre se pueden sincronizar correctamente, asimismo existe la suposición de tener un solo ataque [8], por lo que en el caso de que en la misma red se tengan dos o más ataques wormhole trabajando al mismo tiempo se vuelve imposible detectarlos.

En [5] se propone una contramedida llamada *TrueLink* en la cual un nodo puede verificar la existencia de un enlace directo a un vecino aparente operando en dos fases, la de encuentro y la de autenticación, una vez establecida la adyacencia de un vecino, un protocolo de encaminamiento que utiliza esto puede usar la información para controlar el envío de paquetes. Sin embargo, en caso de que la topología cambie se debe verificar nuevamente la adyacencia de los nodos.

En [6] se propone un mecanismo basado en el tiempo de transmisión *(TTM)* para detectar este ataque durante el procedimiento de configuración de ruta. En este trabajo se calcula el tiempo de transmisión entre nodos adyacentes a lo largo de la ruta establecida, el tiempo de transmisión entre dos vecinos falsos creados por el ataque es considerablemente mayor que el que existe entre dos vecinos reales que están dentro del alcance. TTM tiene buen rendimiento y poca sobrecarga, pero está diseñado específicamente para el protocolo de encaminamiento AODV.

En [7] se propone un método que se basa en la eliminación de los bordes del ataque, lo que causa cambios considerables en la longitud de algunas de las rutas más cortas; la complicación que presenta es que para monitorizar los cambios hacen uso de un nodo raíz, lo que incrementa los cálculos y la demanda de recursos.

En [8] se explora el impacto de éste tipo de ataque en las topologías de conectividad de red y se desarrolla un método distribuido simple para detectarlo, llamado *WormCircle*, el cual proviene de un fenómeno físico, la propagación de la onda en el agua, lo que resulta en la difracción debido al ataque wormhole. Esta propuesta depende únicamente de la información de conectividad local y sin ningún requisito adicional.

En [9] se propone un algoritmo que busca por cada nodo una estructura prohibida en su vecindario, pero en caso de no saber el modelo de comunicación es necesario realizar un trabajo extra para saberlo, porque con esto se obtiene un parámetro fundamental para el algoritmo, lo que afecta al procesamiento, además de que no garantiza la detección del ataque en todos los casos.

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

En esta sección se mencionan los trabajos mayormente relacionados a la propuesta de esta tesis, la detección del ataque wormhole usando el algoritmo DV-Hop. Algunas de las propuestas detectan que ocurre una anomalía pero no son capaces de localizar la zona en la que ésta ocurre, [10], otras asumen que los nodos se distribuyen de manera uniforme en un área rectangular [10,12] o que todos los nodos tienen el mismo radio de cobertura para llevar a cabo la comunicación en la red [11,13,15], también llegan a suponer que los nodos ancla se encuentran dentro del ataque [11,12] o que el canal del ataque es pequeño [11]; por otra parte, hay investigaciones en las que es necesario realizar cálculos adicionales para la detección del ataque [12,14,15] o es necesario que cuenten con características de la red para rastrearlo, como son la potencia de la señal recibida ($RRSI$) y el tiempo que dura el mensaje enviado por la red antes de ser descartado (TTL) [14]. Además la mayoría de ellas suponen que en la etapa de inicio no hay ataque, por lo que si éste ya existía será imposible detectarlo al comenzar el mecanismo.

En [10] se presenta un mecanismo llamado *DWDV*, el cual busca que el mayor número de nodos tenga el menor error en su posición bajo un ataque wormhole, una ventaja con la que cuenta es que proporciona un enfoque sencillo para comprobar el ataque en varios esquemas. El algoritmo funciona haciendo una comparación entre saltos, para ello los autores proponen una medida llamada "salto normal", si entre dos nodos vecinos el salto es mayor que el salto normal se considera la existencia del ataque. No todos los nodos conocen su posición; por lo que no pueden comprobar directamente el movimiento. Pero si recibe dos saltos sin ataque, puede verificar el salto siguiente usando la misma política. El problema con esta propuesta es que no toma en cuenta las fluctuaciones típicas de una señal inalámbrica y además suponen que todos los nodos tienen el mismo radio de comunicación.

En [11] proponen un esquema de localización seguro basado en el proceso básico de localización *DV-Hop* y en etiquetas para defenderse del ataque. La idea principal de este mecanismo es generar una lista de pseudo-vecinos para cada nodo ancla, se usan todas las listas recibidas para clasificar a los nodos atacados en diferentes grupos, y

luego etiquetar a todos los nodos vecinos. Según las etiquetas cada nodo prohíbe las comunicaciones con sus pseudo-vecinos, los cuales están siendo atacados.

En [12] se propone un algoritmo llamado *AWDV-hop* que consta de los siguientes procesos, primero a través de las listas de la relación entre vecinos *NNRL*, los nodos ancla sospechosos se pueden encontrar mediante una comparación entre el número de vecinos. Luego, estos mismos calculan las distancias desde otros nodos ancla para determinar si realmente están siendo atacados, marcándose con 1 o 2, indicando que son afectados por diferentes nodos de ataque. Los nodos marcados con 1 desconectan el enlace de comunicación a los nodos marcados con 2 para eliminar la influencia del ataque.

En [13] se presenta un algoritmo que consta de 4 pasos llamado *WRL-Wormhole-Resistent Localization*, el primer paso consiste en que los nodos ancla transmitan sus ubicaciones por toda la red, en el segundo paso se lleva un proceso de detección del ataque, en el cual los nodos ancla toman la decisión de confianza en el conteo de saltos, después de que un nodo ancla obtiene el recuento de saltos a los demás nodos ancla y toma la decisión de confiabilidad entre ellos, estima un tamaño promedio de salto, por último se manda esta información en la red teniendo inundaciones controladas.

En [14] la idea principal es conectar una contramedida proactiva al esquema básico de *DV-Hop* llamada *Infection prevention*, el algoritmo propuesto es llamado *Wormhole-Free DVHop (WFDV)* el cual incluye dos fases: prevención de infecciones y localización segura basada en *DV-Hop*, la primera fase se realiza para evitar la contaminación del ataque, cada nodo crea la lista de vecinos e intenta encontrar los enlaces sospechosos; así, mientras cada nodo elimina los enlaces falsos de su lista de vecinos, en el segundo paso, el procedimiento de localización puede ser realizado con éxito.

En [15] se propone un algoritmo de seguridad para resistir el *Multi-Wormhole-Node-Link (MWNL)*, en el cual se establece una lista de vecinos en la fase de inicialización para encontrar a los nodos ancla sospechosos. Los nodos ancla atacados generan y transmiten un mensaje para distinguir las diferentes áreas del ataque, al mismo tiempo los nodos desconocidos se marcan en dos rondas, en caso contrario, se dice que están semiaislados.

A diferencia de los trabajos previamente mencionados, este trabajo de tesis no asume que todos los nodos tienen el mismo rango de transmisión como es el caso de [12,13,14,15], lo que limita su uso a sensores homogéneos. Tampoco requiere el cálculo previo de constantes que permiten identificar el ataque como es el caso de [12], no basa su detección en parámetros como RSSI y TTL que tienden a ser muy variables y poco confiables para tomarse como mediciones principales, ni presupone que la red en un principio no es atacada como es el caso de [14]. Tampoco presupone que los nodos ancla deben estar dentro de las áreas de cobertura de los nodos maliciosos. Por el contrario, este trabajo de tesis solo requiere el conocimiento del 1-vecindario y presupone que los nodos pueden o no tener el mismo radio de cobertura, además no es necesario que estos estén distribuidos de manera uniforme. Adicional a eso, requiere de muy poco procesamiento a nivel local con la creación de un spanning tree, lo que ayuda a conservar recursos como la energía.

Capítulo 3

Desarrollo

En este capítulo se presentan los algoritmos que componen la propuesta de tesis y se muestra su funcionamiento en conjunto.

3.1. Distance Vector Hop (DV-Hop)

Muchas de las aplicaciones en las redes de sensores requieren que los datos recolectados estén relacionados con una posición geográfica, por lo que es indispensable que los sensores conozcan su ubicación. Para ello, se han propuesto múltiples técnicas de localización que permiten estimar la posición de los sensores [16][17][18][19]. Estas técnicas han sido clasificadas en métodos basados en rango (*range-based*) y métodos no basados en rango (*range-free*). Por un lado, los esquemas range-free utilizan métricas como proximidad o conectividad. Mientras que los esquemas range-based, utilizan técnicas como son AoA, ToA y TDoA para estimar la distancia entre dos nodos. Sin embargo, estos últimos requieren de hardware especializado y costoso. Para el desarrollo de esta tesis se hace uso del algoritmo range-free DV-Hop como parte fundamental para detectar candidatos que pudieran estar bajo la influencia del ataque wormhole. A continuación se explica el funcionamiento del algoritmo range-free DV-Hop.

3.1.1. Funcionamiento del algoritmo *DV-Hop*

El algoritmo *DV-Hop* utiliza la técnica de multilateración para estimar la posición de los sensores a través de un conteo de "saltos". DV-Hop define dos tipos de nodos: los nodos denominados ancla (*hook points*) cuya posición es conocida debido a que cuentan con *GPS*, y el resto de nodos denominados desconocidos, que tratarán de estimar su posición con ayuda de los nodos ancla. Cabe mencionar que por cuestiones de costo es complicado instalar *GPS* en todos los nodos de la red, por lo que únicamente se tienen algunos nodos con esta característica. El funcionamiento de DV-Hop se divide en tres fases:

- Primera fase: dentro de ésta etapa se lleva a cabo un proceso llamado "inundación" o "flooding", el cual sirve para estimar el mínimo número de saltos entre

los nodos desconocidos y los nodos ancla. Para ello, cada nodo ancla transmite un mensaje de datos que contiene el número de identificación (ID), las coordenadas del nodo ancla (x,y) y la información del número de saltos (al inicio todos los nodos anclan tienen en su campo saltos un valor de cero, y los nodos desconocidos un valor de infinito). El mensaje de datos se puede ver como {ID, x, y, saltos}. Cuando algún nodo (ancla o desconocido) recibe un mensaje, el número de saltos se aumenta en 1 y se compara con su campo saltos, si éste es menor, entonces se actualiza. Además el nodo guarda la información del nodo ancla en un registro de memoria. Posteriormente el mensaje se reenvía a los nodos vecinos. Este proceso permite que todos los nodos obtengan la mínima distancia a los nodos ancla; en términos de saltos, considerando a este número como la cantidad de nodos que separan a un nodo desconocido con respecto a un nodo ancla.

- Segunda fase: en esta fase, los nodos ancla después de recibir la información de los demás nodos ancla {ID, x, y, saltos}, calculan la distancia media por salto *average hopsize*, la cual se obtiene promediando las distancias a los demás nodos ancla sobre los conteos de saltos. La distancia promedio de cada salto o también llamado "hopsize" se puede calcular como:

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

  donde (x_i, y_i) y (x_j, y_j) son las coordenadas de los nodos ancla X_i y X_j, respectivamente. $hops_{ij}$ es el número de saltos entre X_i y X_j. Finalmente el hopsize estimado por los nodos ancla es enviado por la red para conocimiento de los nodos desconocidos.

- Tercera fase: en esta fase se calculan las coordenadas del nodo desconocido utilizando la información obtenida de las dos primeras fases, es decir, cuando un nodo desconocido obtiene tres o más distancias de los nodos ancla (al multiplicar el hopzise por el número de saltos a un nodo ancla), sus coordenadas se pueden estimar utilizando el método de multilateración al resolver un sistema de ecuaciones.

Para fines de esta tesis no es importante resolver el sistema de ecuaciones en la tercera fase, basta con que haya un área de intersección entre todas las circunferencias del método de multilateración (dependiendo del número de nodos ancla presentes en la red) para poder localizar a un nodo desconocido (ver figura 3.1, donde el nodo desconocido es coloreado en rojo).

Existen casos en los que la localización del nodo desconocido puede tener errores, como el que no haya suficientes nodos ancla o que no haya intersección entre las circunferencias pertenecientes a cada nodo ancla (ver figura 3.2, donde el nodo desconocido es coloreado en rojo).

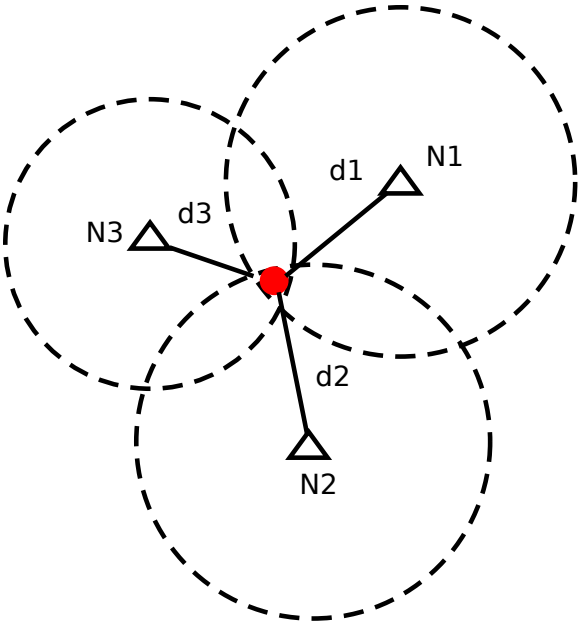


Figura 3.1: Método de trilateración.

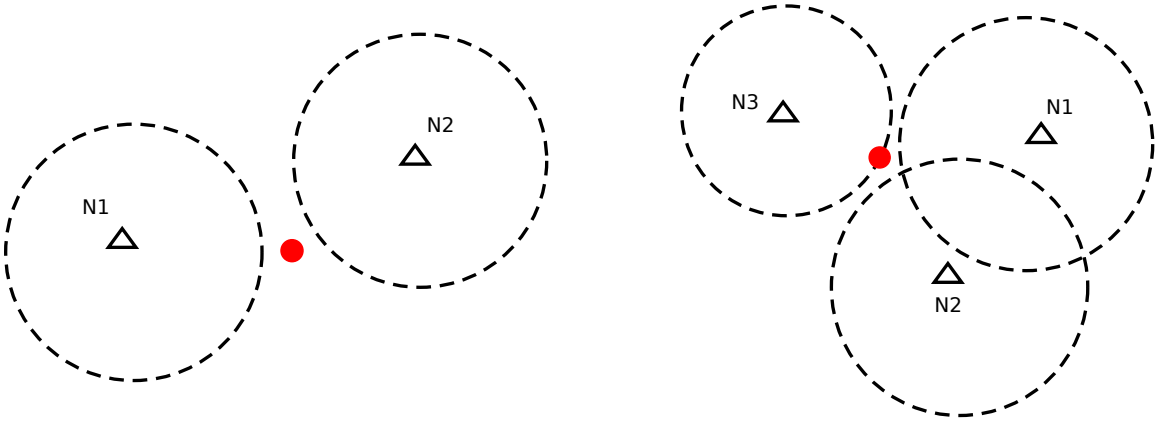


Figura 3.2: Algunos errores en la localización del nodo desconocido.

La figura 3.3 muestra un ejemplo de cómo opera el algoritmo DV-Hop en una red libre de ataques, mientras que la figura 3.4 presenta un ejemplo de cómo esa misma red es afectada por el ataque *wormhole*. Los nodos ancla para ambas figuras son coloreados en blanco (1, 2 y 3), mientras que los nodos desconocidos son coloreados en negro. Para ambas figuras se tienen dos nodos intentando auto-localizarse con ayuda de los nodos ancla, representados en color azul y magenta. En la figura 3.3 se puede observar que ninguno de los nodos tiene problemas para auto-localizarse, el nodo azul, por ejemplo se encuentra a dos saltos de cada uno de los nodos ancla, mientras que el nodo magenta se encuentra a uno, tres y cuatro saltos de lo nodos ancla 1, 2 y 3, respectivamente. Por lo que para ambos nodos será posible aplicar el método de trilateración. Por otro lado, en la figura 3.4, el nodo azul mantiene el mismo número de saltos hacia los nodos ancla, debido a que se encuentra fuera del rango del ataque y su cuenta de saltos no es afectada. Sin embargo, el nodo en color magenta, se encuentra dentro de la cobertura del ataque (representado por elipses en color rojo), por lo que al momento de realizar el conteo de saltos presentará errores; para llegar a los nodos ancla marcados con los números 1 y 2 se mantiene el número de saltos, pero para llegar al nodo ancla 3, el número de saltos cambia de cuatro a dos, esto provocará que el nodo magenta no pueda auto-localizarse debido a que el nodo ancla marcado con el número 3 lo ve como 2-vecino, cuando en realidad se encuentra más alejado de él, esto también afectará la intersección de las circunferencias en el método de trilateración. Claramente podemos visualizar que el ataque puede provocar una anomalía en el algoritmo de localización DV-Hop.

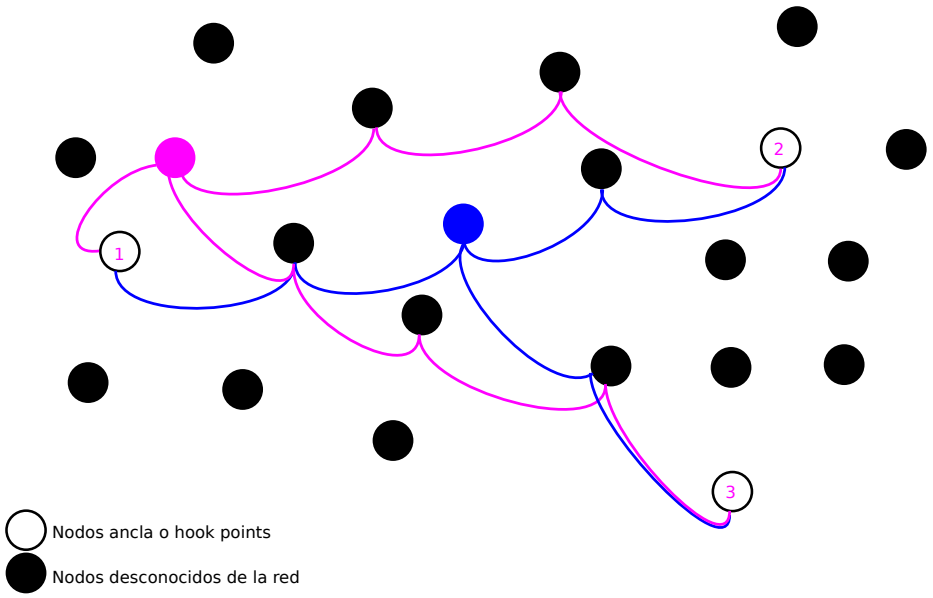


Figura 3.3: Ejemplo de Distance Vector-Hop sin ataque.

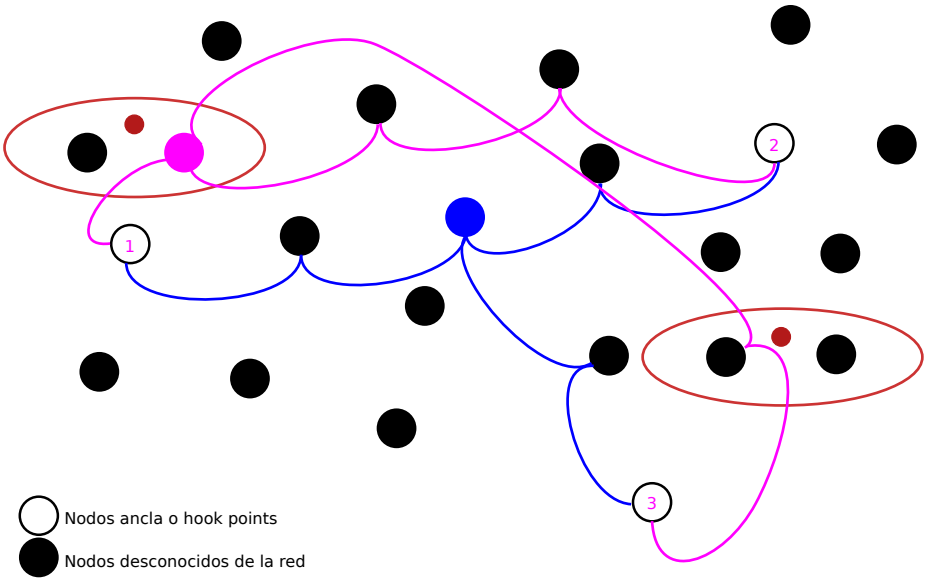


Figura 3.4: Ejemplo de Distance Vector-Hop con ataque.

La figura 3.5 muestra otro ejemplo de cómo el ataque *wormhole* puede afectar al algoritmo de localización. En una red sin ataque, el nodo ancla A se encuentra a 7 saltos de distancia del nodo ancla B pero en la existencia de un *wormhole*, el recuento de saltos entre ellos cambia a 3, lo que lleva a que A y B hagan una estimación falsa en el tamaño medio del salto (hopsize). De la misma forma, los nodos cercanos a A supondrán que hay un menor número de saltos a B y viceversa, por lo que el método de trilateración fallará en la estimación.

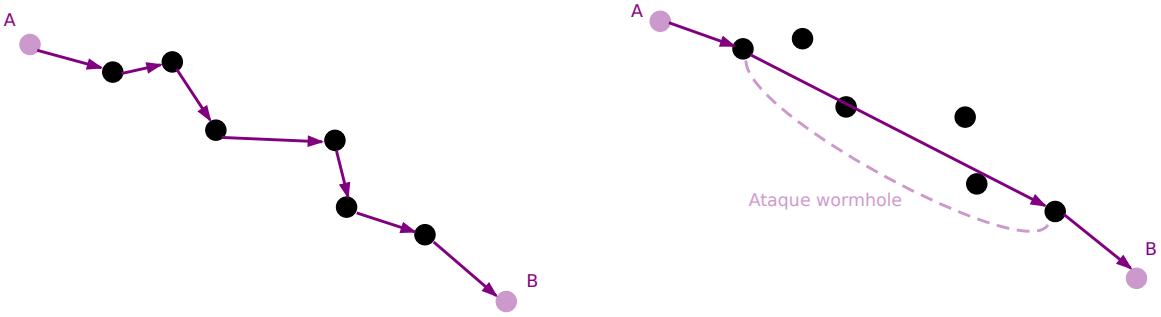


Figura 3.5: Ejemplo de Distance Vector-Hop con/sin ataque.

El objetivo por el que se eligió este algoritmo fue el hecho de que permite identificar las zonas de ataque, ya que los nodos desconocidos cercanos a los nodos maliciosos fallarán en su localización, lo que da una pista de dónde buscar a los posibles nodos que componen el ataque *wormhole*. Además de que es simple y tiene un bajo costo computacional.

3.2. Spanning tree

Un árbol de expansión o *spanning tree* es un sub-grafo de un grafo conectado no dirigido, que incluye todos los vértices del grafo con el mínimo número de aristas (ver figura 3.6). El número total de árboles de expansión posibles que se pueden crear a partir de un grafo con n vértices es igual a:

$$n^{(n-2)} \tag{3.2}$$

Un árbol de expansión siempre debe cumplir con la siguiente propiedad: si hay n vértices en un grafo, entonces el árbol de expansión debe tener $n-1$ aristas.

Si un árbol de expansión presenta el mínimo peso en todas sus aristas, se dice que es un árbol de expansión mínimo. Este peso puede representar la distancia entre dos nodos, el ancho de banda en el canal o algún otro valor que represente una métrica de costo ente dos vértices. Todo grafo conectado y no dirigido tiene al menos un árbol de expansión.

Figura 3.6: Gráfico no dirigido y su árbol de expansión.

Propiedades generales del árbol de expansión

- 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 un arista del árbol de expansión hará que el grafo se desconecte.
- Agregar un arista al árbol de expansión creará un ciclo, por esto se dice que el árbol de expansión es máximamente acíclico.

A continuación se muestra como se puede calcular el árbol de expansión o *spanning tree* de manera distribuida, es decir, de manera local sin el conocimiento del grafo entero. Para ello, se deben tener en cuenta dos conjuntos:

- El conjunto A contiene a todos los nodos que forman parte del 1-vecindario del nodo con el que se está trabajando, es decir, los nodos que tiene conectados directamente.

- El conjunto B del nodo analizado, es aquel que contiene el nodo por el cual es descubierto, el cual se llama "padre" y los vecinos directos del padre.

Para el nodo que inicia la creación del árbol de expansión, su conjunto B siempre es vacío.

Para crear el árbol de expansión, cada nodo en la red debe enviar un mensaje compuesto por el resultado de $A - B$, el cual contiene aquellos elementos que están en A que no están en B. Si la resta de estos conjuntos resulta en un conjunto vacío, implica que el nodo no transmite el mensaje. Para esta tesis, a este conjunto de nodos se llama "breakpoint (BP)" u hoja del árbol; esto quiere decir que es un nodo en el cual se corta la construcción del *spanning tree*.

La figura 3.7 muestra algunos escenarios que ilustran las propiedades antes mencionadas. Se muestra un grafo con 7 nodos, lo que implica que se pueden formar $7^{(7-2)}$ posibles árboles de expansión, es decir, 16807 árboles, todos ellos formados por 6 aristas. Por cuestiones de ilustración solo se muestran 4 árboles de expansión, los cuales son el resultado del mecanismo realizado bajo diferentes condiciones, en ellos se muestran todas las conexiones existentes entre los nodos A, B, C, D, E, F y G. Las líneas continuas representan las aristas que forman el árbol de expansión, mientras las líneas punteadas indican las aristas que se eliminaron. En los primeros dos árboles (parte superior) se inicia el algoritmo en el nodo A, pero se descubren a los demás nodos en diferente orden. Para los siguientes árboles (parte inferior), se inicia el *spanning tree* en el nodo B y C, respectivamente.

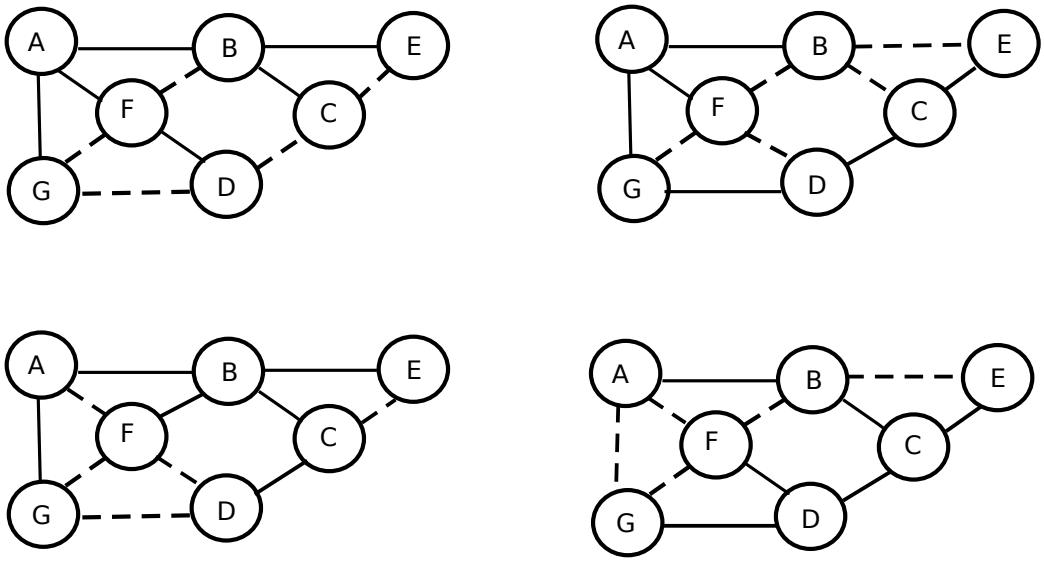


Figura 3.7: Spanning Tree bajo diferentes condiciones.

Árbol 1 (parte superior izquierda):
Inicio: A; Orden: A, B, F, G, D, C, E.
Árbol 2 (parte superior derecha):
Inicio: A; Orden: A, G, F, D, C, E, B.
Árbol 3 (parte inferior izquierda):
Inicio: B; Orden: B, F, G, A, E, C, D.
Árbol 4 (parte inferior derecha):
Inicio: C; Orden: C, E, D, B, F, G, A.

3.2.1. Ejemplo de construcción del algoritmo *spanning tree*

La figura 3.8 muestra una red conformada por 10 nodos, los cuales son marcados con números del 0 al 9. El árbol de expansión estará formado por 9 aristas, y existirán un total de 10^8 posibles árboles. Es importante mencionar que dentro de este ejemplo no se esta tomando en cuenta la existencia del ataque *wormhole*.

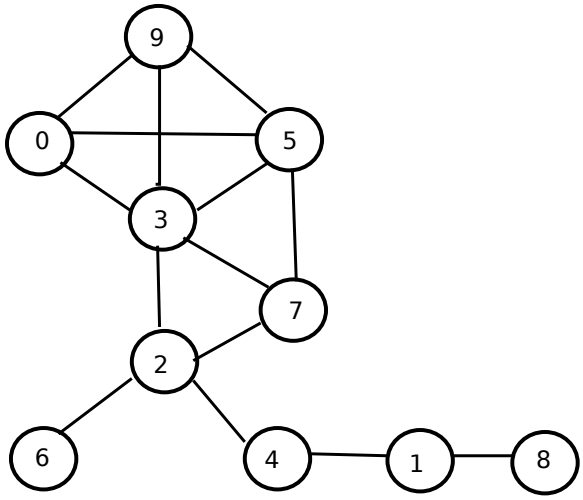


Figura 3.8: Red inicial.

Se toma al nodo 9 como iniciador del algoritmo y posteriormente, se sigue un orden elegido aleatoriamente, este orden es: $9 \rightarrow 0 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 1 \rightarrow 8$.

Nodo 9:

A $=\{0, 5, 3\}$ B $=\emptyset$ A-B$=\{0, 5, 3\}$

En este caso el conjunto B es vacío, debido a que el nodo 9 está iniciando el árbol de expansión y no fue descubierto por alguien más. El resultado de la resta indica el conjunto de nodos a los que se les enviará un mensaje multicast y ahora forman parte del árbol, como se muestra en la figura 3.9.

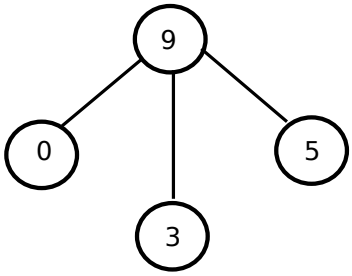


Figura 3.9: Inicializando árbol.

Nodo 0:

Padre= 9 A =$\{9, 5, 3\}$ B =$\{9, 0, 5, 3\}$ A-B=$\emptyset \rightarrow BP$

Nodo 3:

Padre= 9 A =$\{0, 9, 5, 7, 2\}$ B =$\{9, 0, 5, 3\}$ A-B=$\{7, 2\}$

El nodo 0 se convierte en un BP, mientras el nodo 3 agrega nuevas ramas, como se muestra en la figura 3.10.

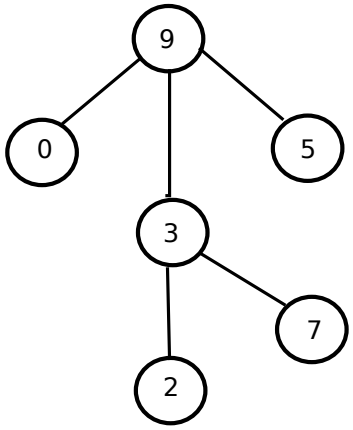


Figura 3.10: Continuación del árbol.

Nodo 5:

Padre= 9 A =$\{9, 3, 7\}$ B =$\{9, 0, 5, 3\}$ A-B=$\{7\}$

Para este caso, el nodo 7 ha sido descubierto por 2 padres (3,5), por lo que se debe eliminar la arista (5-7) para que no forme un ciclo.

Nodo 7:

Este nodo tiene doble padre, ya que fue descubierto por dos nodos, por lo que ambos se deben poner como elementos del conjunto B.

Padres= 3 y 5 A =$\{5, 3, 2\}$ B =$\{3, 7, 2, 5, 7\}$ A-B=$\emptyset \rightarrow$ BP

Al igual que el nodo 0, el nodo 7 es una hoja del árbol, debido a que en él se detiene la expansión, esto quiere decir que no se agregan ramas al *spanning tree*.

Nodo 2:
Padre= 3 A =$\{3, 7, 6, 4\}$ B =$\{3, 7, 2\}$ A-B=$\{4, 6\}$
El árbol actualizado se muestra en la Figura 3.11.

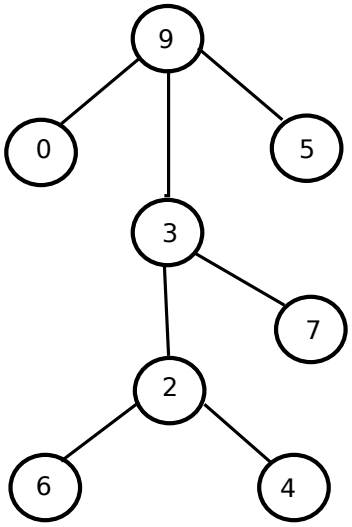


Figura 3.11: Actualización del árbol de expansión.

Nodo 4:
Padre= 2 A =$\{2, 1\}$ B =$\{2, 4, 6\}$ A-B=$\{1\}$

Hasta este punto el árbol de expansión se muestra en la Figura 3.12.

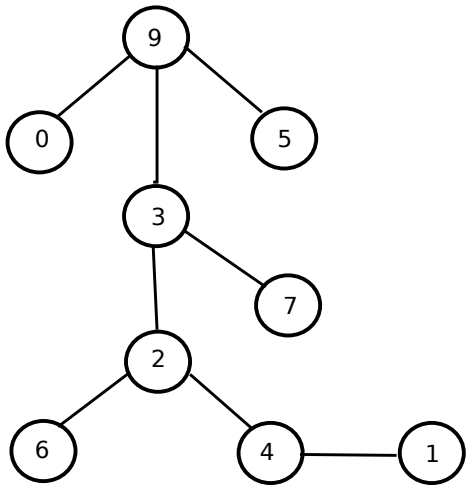


Figura 3.12: Árbol actualizado.

Nodo 6:
Padre= 2 A =$\{2\}$ B =$\{2, 4, 6\}$ A-B=$\emptyset \rightarrow$ BP
Nodo 1:
Padre= 2 A =$\{4, 8\}$ B =$\{4, 1\}$ A-B=$\{8\}$
Para este nodo se agrega una arista que apunta hacia el nodo 8.
Nodo 8:
Padre= 2 A =$\{1\}$ B =$\{1, 8\}$ A-B=$\emptyset \rightarrow$ BP

Al tener una hoja en el último nodo, el árbol final queda como se muestra en la figura 3.13.

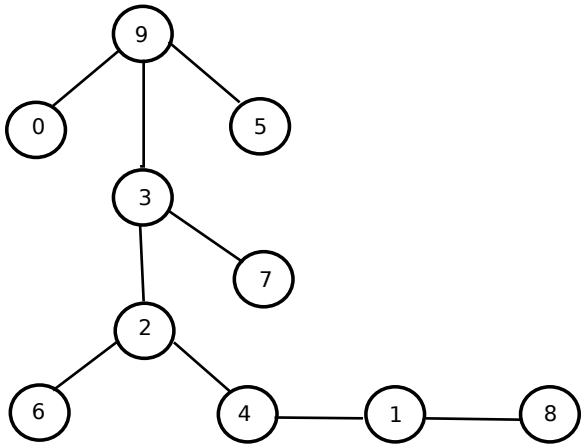


Figura 3.13: Árbol final.

Ejemplo de spanning tree con ataque *wormhole*

A continuación se presenta un ejemplo para construir un árbol de expansión, pero en esta ocasión se considera la existencia del ataque *wormhole*. La cobertura del *wormhole* se presenta por elipses en color negro, donde los puntos negros son las antenas que conforman el ataque (ver figura 3.14). Los nodos afectados por el ataque son 1, 8, 5, y 9. Se toma en cuenta el mismo orden de descubrimiento de nodos que en el ejemplo anterior.

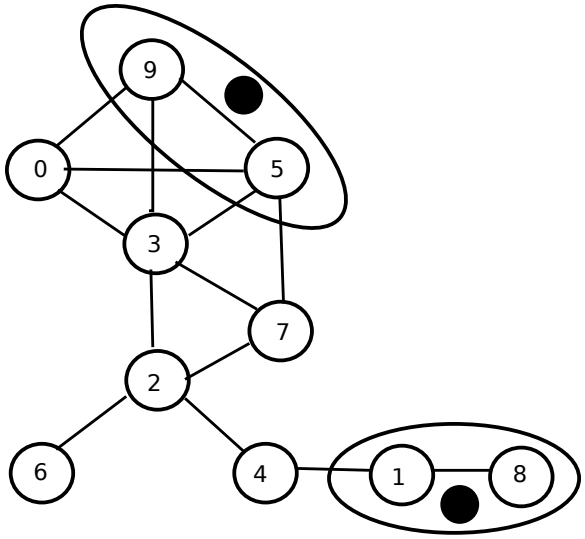


Figura 3.14: Red inicial con ataque wormhole.

Nodo 9:

A $=\{0, 5, 3, 1, 8\}$ B $=\emptyset$ A-B$=\{0, 5, 3, 1, 8\}$

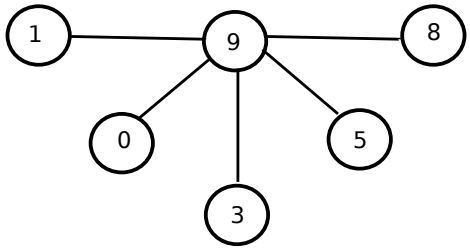


Figura 3.15: Árbol inicial con ataque.

Nodo 0:

Padre= 9 A $=\{0, 5, 3\}$ B $=\{9, 0, 5, 3, 1, 8\}$ A-B$=\emptyset \rightarrow$ BP

Nodo 3:

Padre= 9 A $=\{0, 9, 5, 7, 2\}$ B $=\{9, 0, 5, 3, 1, 8\}$ A-B$=\{7, 2\}$

El nuevo árbol se muestra en la figura 3.16.

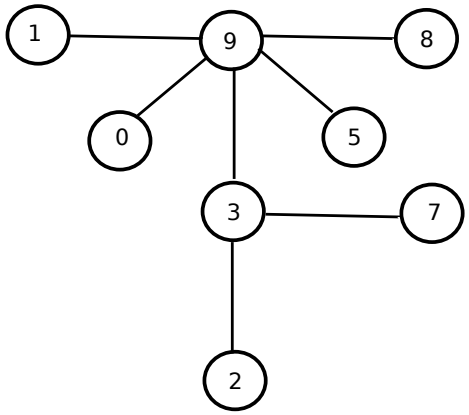


Figura 3.16: Actualización del árbol con ataque.

Nodo 5:
Padre= 9 A $=\{9, 0, 3, 7, 1, 8\}$ B $=\{9, 0, 5, 3, 1, 8\}$ A-B$=\{7\}$
Se descarta la arista debido a que existe un segundo padre.
Nodo 7:
Padres= 3 y 5 $A=\{5, 3, 2\}$ $B=\{3, 7, 2, 5, 7\}$ A-B$=\emptyset \rightarrow$ BP
Nodo 2:
Padre= 3 $A=\{3, 7, 6, 4\}$ $B=\{3, 7, 2\}$ A-B$=\{4, 6\}$
El árbol de expansión con los cambios hasta este momento se muestra en la figura 3.17.

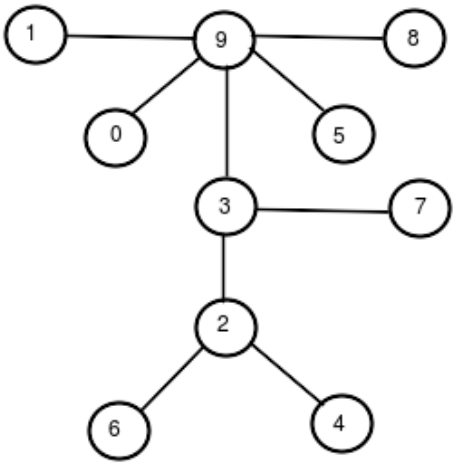


Figura 3.17: Actualización del árbol de expansión con ataque wormhole.

Nodo 4:
Padre= 2 $A=\{2, 1\}$ $B=\{2, 4, 6\}$ A-B$=\{1\}$
Se descarta porque ya existe una arista que conecta con el nodo 1.

Nodo 6:
Padre= 2 A $=\{2\}$ B $=\{2,4,6\}$ A-B$=\emptyset \rightarrow$ BP
Nodo 1:
Padres= 9 y 4 A $=\{4,8,9,5\}$ B$=\{9,0,3,5,1,8,4,1\}$ A-B$=\emptyset \rightarrow$ BP
Nodo 8:
Padre= 9 A $=\{1,9,5\}$ B $=\{9,0,3,5,1,8\}$ A-B$=\emptyset \rightarrow$ BP

Para el escenario anterior se tienen dos árboles, con ataque (derecha) y sin ataque (izquierda), como se muestra en la figura 3.18.

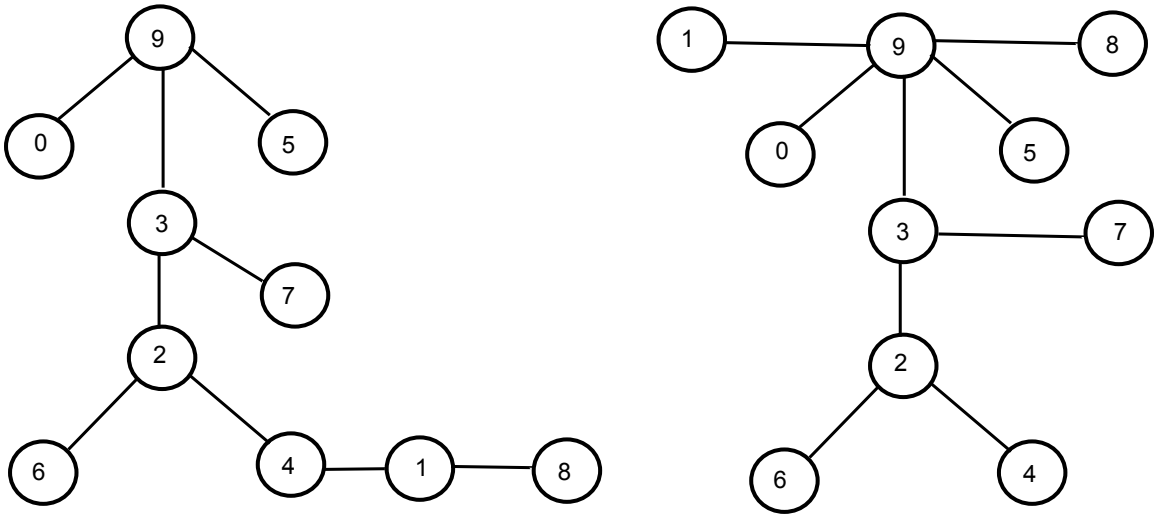


Figura 3.18: Árboles de expansión.

Es importante hacer notar que durante la creación del árbol de expansión, en la presencia de un ataque, se crea una arista entre dos nodos afectados por el ataque. Para este ejemplo es la arista (1-9). Esto se debe a que en la construcción del árbol el nodo 9 es vecino aparente del nodo 1 y al estar físicamente separados no existe un vecino del nodo 9 fuera del ataque que haya podido descubrir al nodo 1 previamente, en otras palabras, el conjunto B que descubre a 9 no puede descubrir a los nodos 1 y 8 previamente. Sin embargo, 9 lo descubre como 1-vecindario debido al ataque. También es importante hacer notar que un *wormhole*, se convierte en un grafo bipartito en términos topológicos.

Definición de grafo bipartito

Sea un grafo G expresado como la unión de dos subconjuntos de vértices V_1 y V_2, de forma que cada arista de G une un vértice de V_1 con otro de V_2, entonces se dice que G es un grafo bipartito. Se cumple que $V_1 \cap V_2 = 0$; $V_1 \cup V_2 = V$.

Un grafo bipartito en el cual todos los elementos de V_1 (rojo) están unidos con todos los elementos de V_2 (azul) se denomina grafo bipartito completo (figura 3.19).

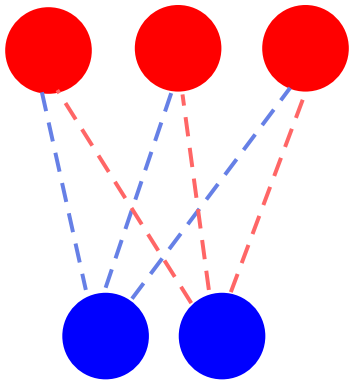


Figura 3.19: Grafo bipartito completo.

3.2.2. Contramedida del ataque wormhole

En esta tesis, el algoritmo de spanning tree tiene el propósito de identificar alguna conexión entre los posibles nodos maliciosos previamente localizados con *DV-Hop*. Para ello, un nodo elegido al azar inicializa el árbol de expansión, conforme el árbol se va expandiendo, los nodos cuya auto-localización fallaron en la primera etapa (DV-Hop) envían al nodo que inició el *spanninig tree* la información relativa a su 1-vecindario, con esta información éste verifica la existencia de un grafo bipartito entre los posibles nodos que no pudieron auto-localizarse, dicho de otra manera, valida una conexión bipartita entre los nodos inscritos en las zonas que presentan mayor probabilidad de influencia del ataque. Si éste nodo encuentra un grafo bipartito, en la tercera fase inicia un nuevo *spanning tree* avisando a los nodos que son afectados por el ataque, esto con la intención de que no tomen en cuenta las aristas formadas por el grafo bipartito. De esta manera se suprime el ataque *wormhole*. Para ambos árboles de expansión, los nodos *break-points* regresan al nodo que inició el árbol, la parte que ellos conocen con el fin de que éste reconstruya el *spanning tree*.

La figura 3.20 muestra un ejemplo de la presencia de un grafo bipartito. Suponiendo que las áreas dudosas son las mostradas por las elipses en color rojo, marcadas como **zona de peligro** 1 y **zona de peligro** 2, donde la zona de peligro 1 está conformada por tres nodos de color negro, azul y lila, mientras que la zona de peligro 2 está compuesta por dos nodos de color cian y magenta. El nodo que inicia el *spanning tree* solicita a los cinco nodos inscritos en estas zonas que indiquen su 1-vecindario, debido a que en el árbol de expansión principal no es posible ver esta anomalía (ver figura 3.21). Con está información el nodo que inicia el árbol es capaz de buscar grafos bipartitos entre los posibles candidatos, y de detectarlo requerirá de la construcción de un nuevo árbol de expansión, en el cual le dirá a los nodos atacados que no tomen en cuenta las aristas pertenecientes al grafo bipartito.

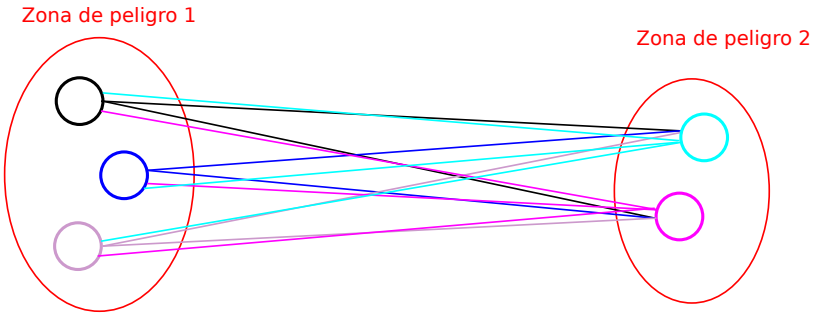


Figura 3.20: Grafo bipartito completo en el escenario planteado.

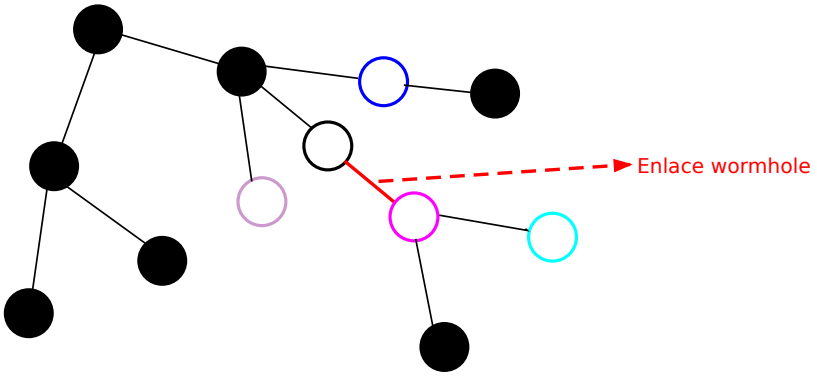


Figura 3.21: Árbol principal con ataque aún no detectado.

Posteriormente, se puede reafirmar la presencia de un ataque al hacer un estudio estadístico entre la media y la desviación estándar de saltos. Para este estudio se deben considerar los dos árboles de expansión 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. Retomando la figura 3.18, los nodos que se encuentran dentro del ataque son 9, 5, 8 y 1. La arista creada por el árbol con ataque fue 9-1, donde el nodo que inicializa el árbol detecto un grafo bipartito. El 1-vecindario del nodo 1 en el árbol sin ataque son 4 y 8. Mientras que el 1-vecindario del nodo 9 son los nodos 0, 3 y 5. A continuación se muestra la estadística:

Análisis estadístico sin ataque:

Nodo 9:
$1 \rightarrow 9 = 4$ saltos $\quad 4 \rightarrow 9 = 3$ saltos
$8 \rightarrow 9 = 5$ saltos
Media de saltos$\rightarrow E(X) = \frac{12}{3} = 4.0$ saltos

desviación estándar:
$\sigma^2 = \frac{\sum(X-E(X))^2}{n-1} = 0.6666 \qquad \sigma = \sqrt{0.6666} = 0.8164$

Nodo 1:
$9 \rightarrow 1 = 4$ saltos $\quad 0 \rightarrow 1 = 5$ saltos
$5 \rightarrow 1 = 5$ saltos $\quad 3 \rightarrow 1 = 3$ saltos

Media de saltos$\rightarrow E(X) = 4.25$ saltos

$\sigma^2 = \frac{\sum(X-E(X))^2}{n-1} = 0.6875 \qquad \sigma = \sqrt{0.6875} = 0.8291$

Análisis estadístico con ataque:

Nodo 9:
$1 \rightarrow 9 = 1$ saltos $\quad 8 \rightarrow 9 = 1$ saltos
$4 \rightarrow 9 = 3$ saltos

Media de saltos$\rightarrow E(X) = 1.6$ saltos

$\sigma^2 = \frac{\sum(X-E(X))^2}{n-1} = 0.888 \qquad \sigma = 0.9428$

Nodo 1:
$9 \rightarrow 1 = 1$ salto $\quad 0 \rightarrow 1 = 2$ salto
$3 \rightarrow 1 = 2$ salto $\quad 5 \rightarrow 1 = 2$ salto

Media de saltos$\rightarrow E(X) = 1.75$ salto

$\sigma^2 = \frac{\sum(X-E(X))^2}{n-1} = 0.187 \qquad \sigma = 0.43$

Para este ejemplo se puede ver como la media de saltos aumenta cuando se elimina el ataque tanto para el nodo 9 como para el nodo 1. Esto es debido a que el *wormhole* disminuye las métricas de saltos. Por otro lado se puede ver como para el caso del nodo 9, la desviación estándar va de 0.9428 con ataque a 0.8164 sin ataque. Esto se debe al hecho de que cuando existe un ataque existen nodos que usan el *wormhole* y otros que no lo usan, por lo que sus rutas serán de tamaño distinto provocando cambios en la desviación estándar. Sin embargo, también está el caso donde todos los nodos vecinos usan el *wormhole* y por lo tanto su desviación estándar será pequeña, este es el caso del nodo 1.

3.3. Simulación

En esta sección se presentan algunos ejemplos del funcionamiento del algoritmo en una serie de escenarios simulados en el lenguaje de programación de alto nivel Python. Para poder ejecutar la simulación del ataque es necesario verificar que se tiene instalada la versión 2.7, y las siguientes bibliotecas (ver código en Apéndice A):

- Numpy: Es una biblioteca de Python que significa "Numerical Python", su objetivo es permitir la computación numérica. Proporciona potentes estructuras de datos, las cuales garantizan cálculos eficientes.

- Matplotlib: Es una biblioteca de trazado utilizada para crear gráficos 2D y 3D, visualizaciones estáticas, animadas e interactivas en el lenguaje de programación Python. Fue creada tomando como base a Matlab, es multiplataforma y puede ser usada desde scripts o desde la consola.

- Shapely: Es un paquete de Python para el análisis teórico de conjuntos y la manipulación de características planas usando funciones de la biblioteca GEOS (Geometry Engine - Open Source) lo que proporciona una gran funcionalidad espacial.

- Python-tk: Tk ha sido durante mucho tiempo una parte integral de Python. Proporciona un conjunto de herramientas robustas e independientes de la plataforma para administrar ventanas. Además suministra el acceso a un intérprete de Tcl subyacente con el kit de herramientas Tk, que en sí mismo es una biblioteca de interfaz de usuario gráfica multiplataforma.

Los parámetros más importantes dentro del código (ver ápendice A) son:

- Tamaño de la red (size): indica el área en metros cuadrados en la que se distribuyen los nodos aleatoriamente (coordenadas (X_n,Y_n)).

- Número de nodos (N): es el total de nodos que formarán parte de la red.

- Nodo que inicia el *spanning tree* (start): es el nodo en el cual se comienza la construcción del árbol de expansión, recordando que el orden de descubrimiento de los demás nodos depende de su capa MAC.

- Radio de cobertura (coverage): se refiere al área geográfica que abarca el intercambio de información.

- Cobertura del ataque *wormhole* (wormCoverage): es el área afectada por el ataque.

- Nodos ancla (hook): son aquellos que cuentan con *GPS* para la realización del algoritmo *DV-Hop*. Estos pueden mantenerse fijos o pueden escogerse aleatoriamente.

- Nodos *wormhole* (wormnodes): son aquellos que forman parte del ataque, dependiendo del área de cobertura, son las coordenadas en las que se despliega el canal inseguro.

El significado de los colores para los nodos resultantes en la parte gráfica de la simulación son:

- Gris: nodos ancla.

- Amarillo: nodos desconocidos.

- Verde: nodos afectados por el ataque.

- Cian: nodos antena del ataque *wormhole*.

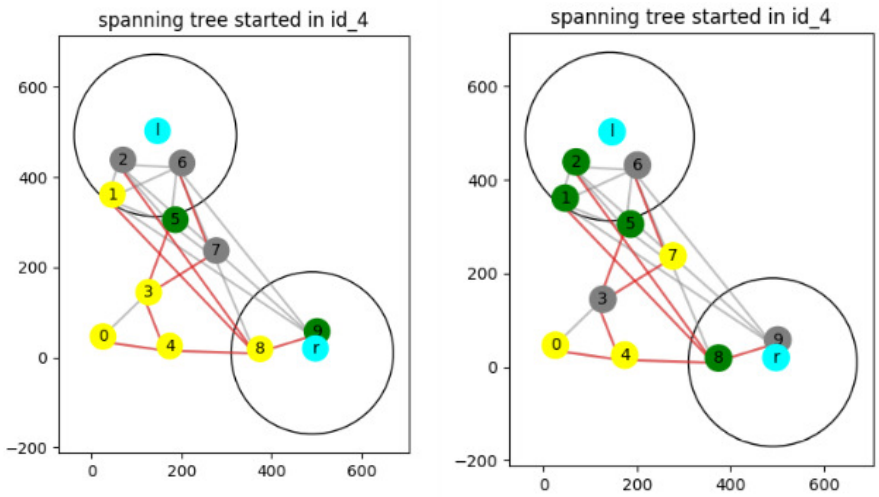
A continuación se muestra un conjunto de escenarios simulados, para los cuales se mantiene el tamaño de la red en $500 \times 500m$, el radio de cobertura de los nodos es de $230m$ y el radio del ataque *wormhole* es de $180m$; cuyas antenas se encuentran en las coordenadas (142,493) y (490,10). La figura 3.22(a) muestra el *spanning tree* (lineas rojas) que inicia en el nodo 4. Para este escenario se utilizaron 3 nodos ancla (2, 6 y 7), se puede observar que los nodos afectados por el ataque son 5 y 9 (aquellos que no se pudieron auto-localizar). Por otro lado, en la figura 3.22(b), los nodos ancla son 3, 6 y 9 y los posibles nodos afectados son 1, 2, 5 y 8. En ambos escenarios, el nodo 4 es capaz de detectar el ataque dado que existe un grafo bipartito en el vecindario de los nodos en color verde. La figura 3.23 muestra el mismo escenario que en la figura 3.22, pero esta vez con 4 nodos ancla: 1, 5, 8 y 9. Los nodos afectados por el ataque son 2, 6 y 4, los primeros dos nodos se encuentran dentro de la cobertura del ataque, pero el último no. Sin embargo, el algoritmo es capaz de detectar el ataque debido a que los nodos 2 y 6 son parte de un grafo bipartito y al hacer el conteo de saltos y eliminar el ataque las medias de salto varían en un gran porcentaje. Efecto que no sucede en el nodo 4, donde a pesar de hay un nodo de color verde, no hay ninguna conexión con otro nodo afectado por el ataque, es decir, el nodo 4 está aislado.

No obstante, existen casos en los no se puede detectar el ataque debido a la ubicación aleatoria de los nodos ancla. Este es el caso de la figura 3.24, donde los nodos dentro del ataque son capaces de auto-localizarse.

La figura 3.25 muestra dos ejecuciones del mismo escenario con 3 nodos ancla. En la figura 3.25(a) se tienen como hook points los nodos 0, 5 y 13, mientras que en la figura 3.25(b) se tienen los nodos 2, 0 y 9. Se puede observar que en la figura 3.25(a) el algoritmo es capaz de detectar el ataque, sin embargo en la figura 3.25(b) no lo es. Cabe resaltar que en estas pruebas se conservaron las métricas del la figura 3.24, pero en esta ocasión se tienen 15 nodos en lugar de 10.

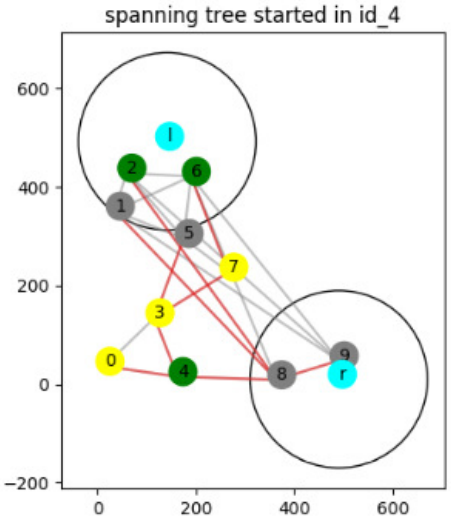
La figura 3.26 muestra cómo la disposición de los nodos ancla presentan una alteración en la detección del ataque. Por ejemplo, para la figura 3.26(a), los nodos ancla (8, 10 y 11) son cuasi-colineales; un caso similar se presenta en el figura 3.26(b), donde además de que los nodos ancla son cuasi-colineales, se encuentran muy cercanos, lo que afecta la detección del ataque.

Las figuras 3.27(a), (b) y (c) muestran una red formada por 25 nodos, considerando tres escenarios con 3,4 y 5 hook points, respectivamente. El nodo que inicializa el árbol es 6. Para la figura 3.27(a), los nodos ancla son 7, 15 y 23; para este escenario la disposición de los nodos ancla permite detectar el ataque. En la figura 3.27(b) los cuatro hook points 5, 7, 15 y 22 descubren a seis nodos posiblemente afectados por el wormhole, todos ellos se encuentran inscritos dentro de las circunferencias de la cobertura del ataque, por lo que el nodo 6 es capaz de detectarlo. En la figura 3.27(c) los nodos ancla 1, 5, 7, 14 y 15 detectan a dos nodos como posibles candidatos a formar parte de un ataque, sin embargo estos no se encuentran dentro de la cobertura de éste, lo que no permite encontrar un grafo bipartito.



(a) Escenario 1 con hook points fijos. (b) Escenario 2 con hook points aleatorios.

Figura 3.22: Escenarios con 10 nodos y 3 hook points.



(a) Escenario 1 con hook points fijos.

Figura 3.23: Escenarios con 10 nodos y 4 hook points.

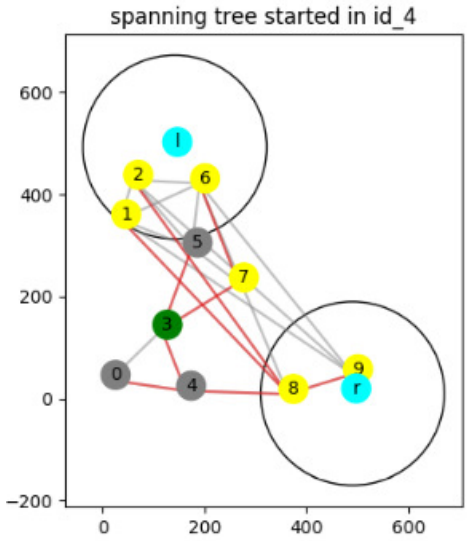
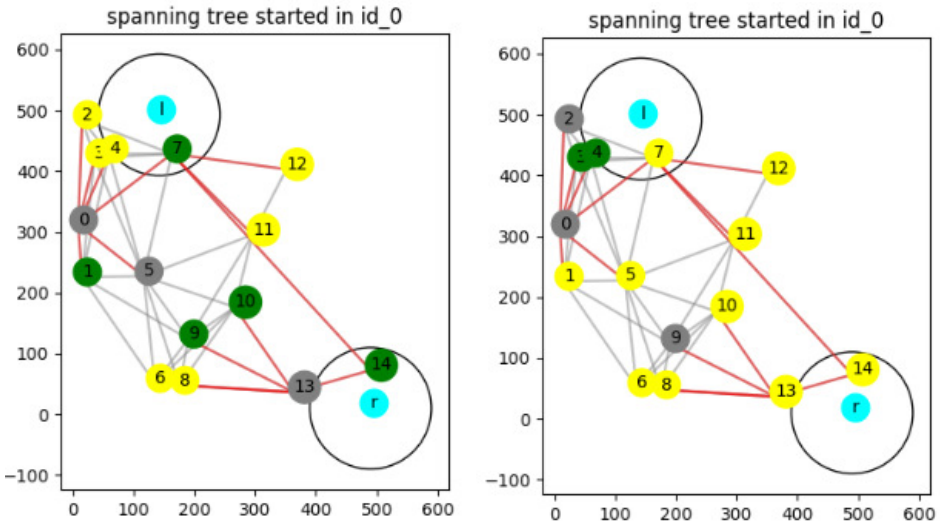
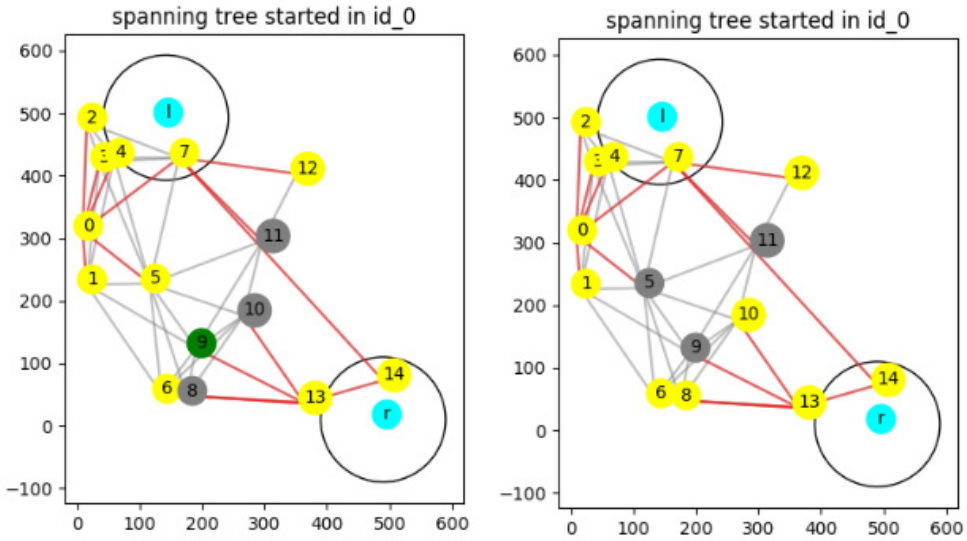


Figura 3.24: Error en la localización del ataque.



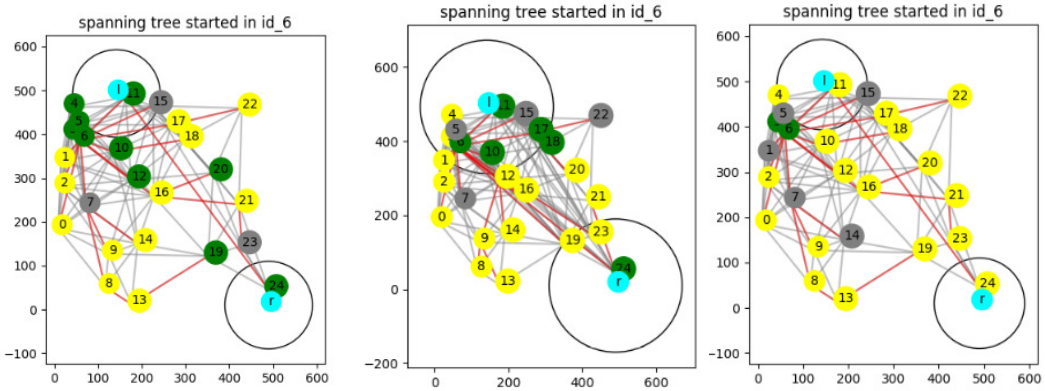
(a) Escenario 1 con hook points aleato-(b) Escenario 2 con hook points aleatorios.
rios.

Figura 3.25: Escenarios con 15 nodos y 3 hook points.



(a) Escenario 1 con error. (b) Escenario 2 con error.

Figura 3.26: Escenarios erróneos.



(a) Escenario 1 con 3 hook points. (b) Escenario 2 con 4 hook points. (c) Escenario 3 con 5 hook points.

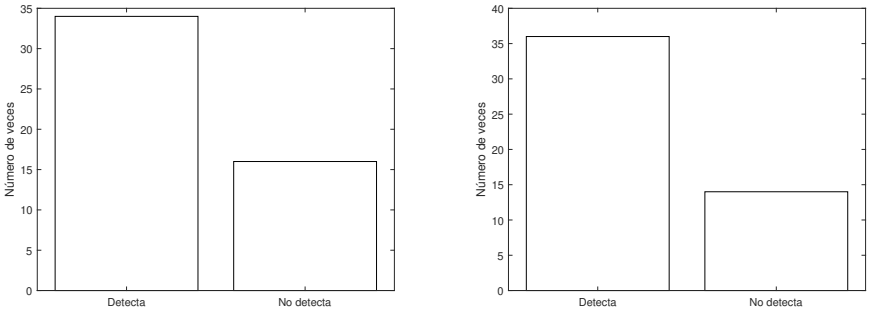
Figura 3.27: Escenarios para 25 nodos.

Capítulo 4

Resultados

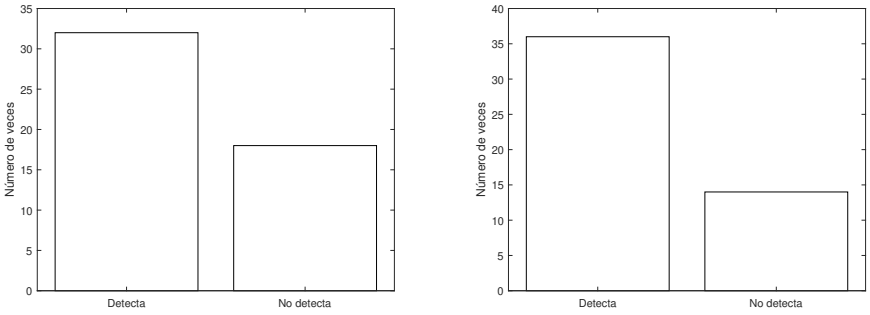
Este capítulo presenta los resultados estadísticos que permiten observar el desempeño del algoritmo. También muestra los casos en los que la propuesta falla. Se llevaron a cabo 50 repeticiones para obtener resultados estadísticos y se tomó un radio de cobertura del *wormhole* de $120m$. En la figura 4.1 se tienen 10 nodos que conforman la red, del total de ejecuciones, 34 detectaron el ataque con 3 nodos ancla, mientras que para 4 nodos ancla fueron 36. Para la figura 4.2 la red está conformada por 20 nodos, de las cuales 32 pruebas detectaron el ataque para 3 hook points y en 36 ocasiones para 4 hook points. La figura 4.3 corresponde a una red formada por 50 nodos, para este caso se realizó el mismo número de ejecuciones para 3,4 y 5 nodos ancla, el ataque fue detectado en 33, 31 y 38 ocasiones, respectivamente. En la figura 4.4 se muestra un escenario formado por 100 nodos; para esta prueba se llevó a cabo la simulación con 4,5 y 10 nodos ancla. En este caso, el ataque fue detectado en 43, 45 y 47 casos, respectivamente.

Las figuras 4,5 y 4.6 muestran el porcentaje de efectividad del algoritmo, tomando en cuenta todos los escenarios planteados anteriormente para cada caso (10,20,50 y 100 nodos), es decir, se generalizó que el ataque wormhole fuera o no detectado independientemente del número de hook points.



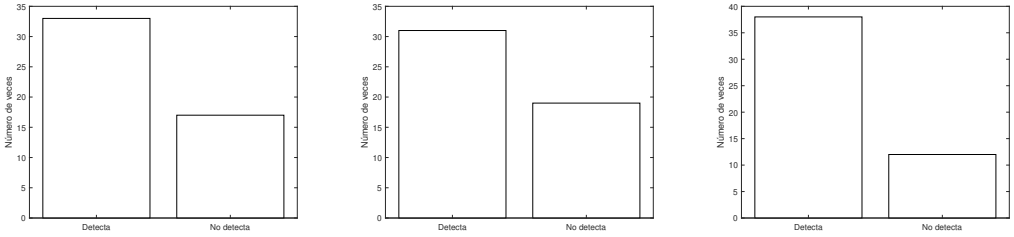
(a) Escenario con 3 hook points. (b) Escenario con 4 hook points.

Figura 4.1: Detecciones para 10 nodos.



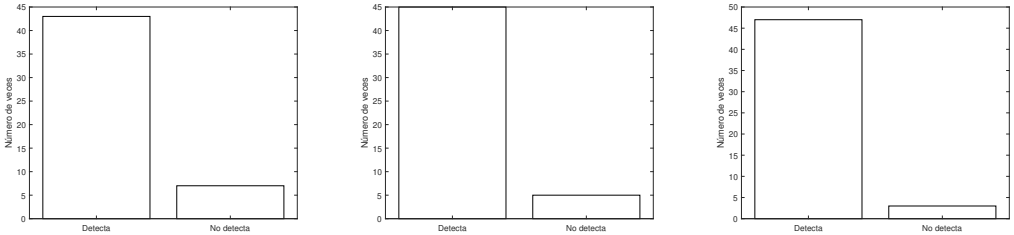
(a) Escenario con 3 hook points. (b) Escenario con 4 hook points.

Figura 4.2: Detecciones para 20 nodos.



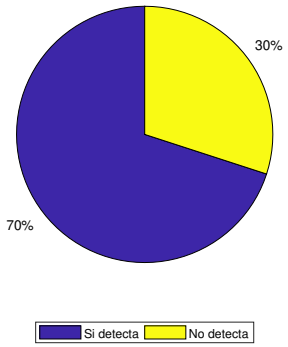
(a) Escenario con 3 hook points. (b) Escenario con 4 hook points. (c) Escenario con 5 hook points.

Figura 4.3: Detecciones para 50 nodos.

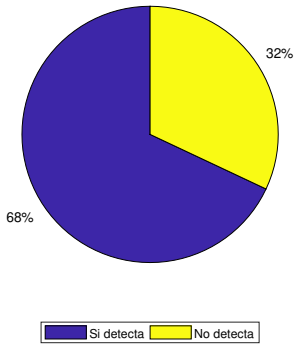


(a) Escenario con 4 hook points. (b) Escenario con 5 hook points. (c) Escenario con 10 hook points.

Figura 4.4: Detecciones para 100 nodos.

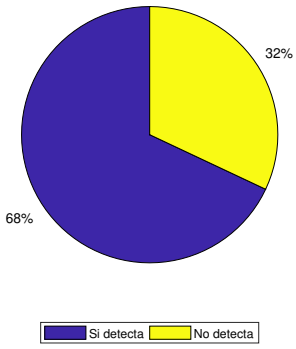


(a) Para 10 nodos.

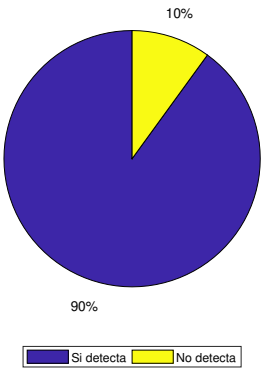


(b) Para 20 nodos.

Figura 4.5: Porcentaje de efectividad.



(a) Para 50 nodos



(b) Para 100 nodos

Figura 4.6: Porcentaje de efectividad.

Para esta sección de pruebas se desarrollaron cuatro escenarios. Cada escenario consta de dos pruebas, donde se varía el número de hook points. El primer y segundo escenario constó de 10 y 20 nodos, respectivamente. Para ambos escenarios se tomaron 3 y 4 hook points. El tercer escenario constó de 50 nodos con 3 y 5 hook points. Finalmente, el último escenario tiene 100 nodos con 5 y 10 nodos ancla. Para cada prueba se graficó el número de veces que el algoritmo detecta el ataque con respecto a la relación longitud del canal (distancia de antena a antena del ataque) entre longitud del hopsize. Se crearon canales de 198, 278, 335, 390, 429, 475, 523, 595, 647 y 678m y se ejecutó 100 veces cada escenario para obtener promedios.

La tabla 4.1 muestra las coordenadas del wormhole.

Longitud del ataque [m]	Coordenadas $(X_1, Y_1)(X_2, Y_2)$
198	(50,350)(200,480)
278	(100,220)(200,480)
335	(15,200)(200,480)
390	(50,120)(200,480)
429	(50,150)(470,60)
475	(10,120)(480,50)
523	(15,480)(300,41)
595	(142,493)(490,10)
647	(30,480)(490,25)
678	(10,490)(490,10)

Tabla 4.1: Coordenadas de los canales del ataque.

Las tablas 4.2, 4.3, 4.4 y 4.5 contienen la longitud del hopsize con respecto a la longitud del canal que forma el ataque wormhole, para 10, 20, 50 y 100 nodos, respectivamente. La relación 'longitud_ataque/hopsize', se muestran en las tablas 4.6, 4.7, 4,8 y 4.9, para 10, 20, 50 y 100 nodos, respectivamente. La figura 4.7(a) muestra el porcentaje de veces que se detecta el ataque con respecto a la relación longitud del ataque entre el hopsize. Para este caso se tiene una red formada por 10 nodos, con 3 hook points. Se detectó un total de 61 % del total de ejecuciones, recordando que esta prueba contiene 10 longitudes del wormhole. Los casos en los que el ataque no fue detectado se debieron a que los nodos ancla estaban muy cercanos unos con otros o como se ve en la figura, la relación longitud/hopsize es muy pequeña. Esto se puede ver claramente para los canales con longitudes de 198, 278 y 335m, cuya relación es 1.43, 2.01 y 2.43, respectivamente. La figura 4.7(b) muestra el mismo escenario para 4 nodos ancla, en este caso fue posible detectar el ataque el 71 % de las veces, al igual que en la figura 4.7(a) se tuvieron problemas de detección en canales con longitudes cortas, tal es el caso de 198 y 335m, en los cuales solo alcanza una detección del 60 % de las ejecuciones.

La figura 4.8(a) se tiene una red formada por 20 nodos, para el caso de 3 hook points se logró detectar el 78 % de las ejecuciones, los escenarios en los que se presentó un menor número de detecciones fueron con canales de 647 y 678m, esto se debe a que el canal tiene una longitud extensa y al tener pocos hook points estos no logran

cubrir el total del área que forma la red. Para la figura 4.8(b) se tiene el caso para 4 hook points, en el cual se tuvo un porcentaje de efectividad del 83 % y el único escenario en el cual se tuvo un porcentaje de detección menor al 50 % fue para un canal de 678m de longitud.

La figura 4.9(a) muestra un escenario formado por 50 nodos con 3 hook points. Para este caso se tuvo un porcentaje de detección del 66 % como consecuencia del tamaño de la red y el pequeño número de nodos ancla. Los casos en los que se presentó un menor número de detecciones fueron para canales de 429, 595 y 647m, el peor fue el último caso, ya que solo se pudo localizar el ataque en el 40 % del total de ejecuciones. Para la figura 4.9(b) incrementó el porcentaje de efectividad a un 82 % puesto que también se presentó un aumento en el número de nodos ancla, ahora al tener 5 nodos era más probable que la detección del ataque fuera correcta; el único escenario que tuvo el 50 % del total de detecciones fue el de 429m.

En la figura 4.10 se tomó en cuenta una red con 100 nodos y en esta dos casos de 5 y 10 hook points. Para el caso de 5 hook points (figura 4.10(a)), se tuvo un porcentaje del 94 % de detecciones, mientras que para el segundo caso (figura 4.10(b)) se presentó un porcentaje del 97 %.

Longitud del ataque [m]	Hopsize [m]	Longitud del ataque [m]	Hopsize [m]
198	143.1	198	154.1
278	138.5	278	131.8
335	137.8	335	152.8
390	127.7	390	134.5
429	155.8	429	137.6
475	124.8	475	126.1
523	134.4	523	135.4
595	137.8	595	140.5
647	136.5	647	135.8
678	136.3	678	135.4

Tabla 4.2: Escenario formado por 10 nodos con 3 (izquierda) y 4 (derecha) hook points, respectivamente.

Longitud del ataque [m]	Hopsize [m]	Longitud del ataque [m]	Hopsize [m]
198	136.5	198	132.8
278	137.6	278	136.1
335	117.2	335	127.4
390	136.1	390	132.8
429	129	429	136.1
475	134	475	132
523	124.4	523	115.3
595	137.1	595	151.8
647	136.5	647	135.8
678	155.9	678	156.9

Tabla 4.3: Escenario formado por 20 nodos con 3 (izquierda) y 4 (derecha) hook points, respectivamente.

Longitud del ataque [m]	Hopsize [m]	Longitud del ataque [m]	Hopsize [m]
198	119.8	198	114.4
278	116.8	278	106.3
335	122.5	335	123.3
390	114.6	390	127.7
429	137.6	429	129.5
475	136.3	475	135.6
523	126.3	523	125.7
595	132.8	595	136.6
647	129.5	647	128
678	139.8	678	136.4

Tabla 4.4: Escenario formado por 50 nodos con 3 (izquierda) y 5 (derecha) hook points, respectivamente.

Longitud del ataque [m]	Hopsize [m]	Longitud del ataque [m]	Hopsize [m]
198	115.4	198	119.8
278	104.5	278	115.4
335	109.1	335	118.4
390	110.8	390	118.2
429	105.4	429	102.8
475	116.1	475	106.2
523	110.3	523	102.3
595	122.9	595	106.3
647	122	647	120.9
678	126.9	678	120.6

Tabla 4.5: Escenario formado por 100 nodos con 5 (izquierda) y 10 (derecha) hook points, respectivamente.

Longitud del ataque [m]	Relación
198	1.43
278	2.01
335	2.43
390	2.82
429	3.11
475	3.44
523	3.79
595	4.31
647	4.69
678	4.91

Tabla 4.6: Relaciones para el escenario formado por 10 nodos.

Longitud del ataque [m]	Relación
198	1.46
278	2.05
335	2.48
390	2.88
429	3.17
475	3.51
523	3.87
595	4.40
647	4.79
678	5.01

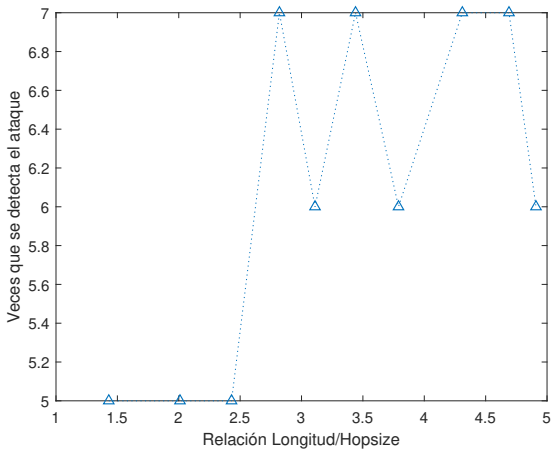
Tabla 4.7: Relaciones para el escenario formado por 20 nodos.

Longitud del ataque [m]	Relación
198	1.55
278	2.18
335	2.63
390	3.07
429	3.37
475	3.74
523	4.11
595	4.68
647	5.09
678	5.33

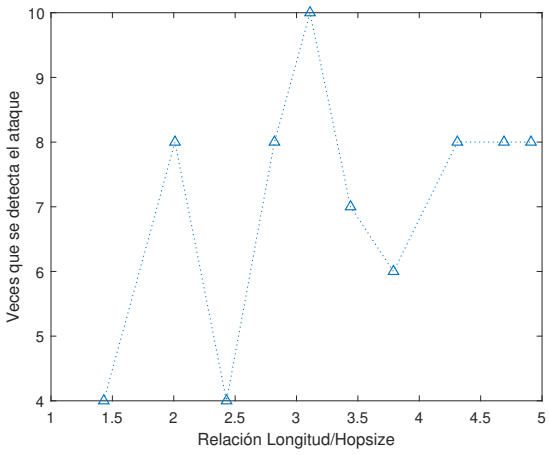
Tabla 4.8: Relaciones para el escenario formado por 50 nodos.

Longitud del ataque [m]	Relación
198	1.74
278	2.44
335	2.94
390	3.42
429	3.77
475	4.17
523	4.59
595	5.23
647	5.68
678	5.96

Tabla 4.9: Relaciones para el escenario formado por 100 nodos.

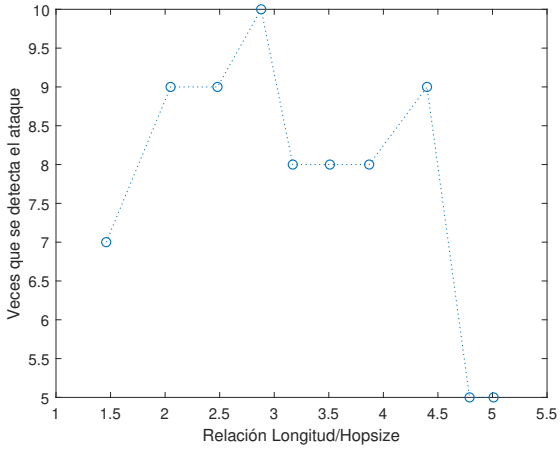


(a) Escenario con 3 hook points.

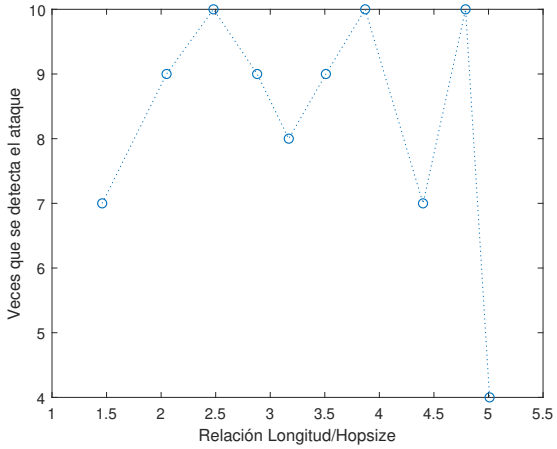


(b) Escenario con 4 hook points.

Figura 4.7: Escenarios detectados para 10 nodos.

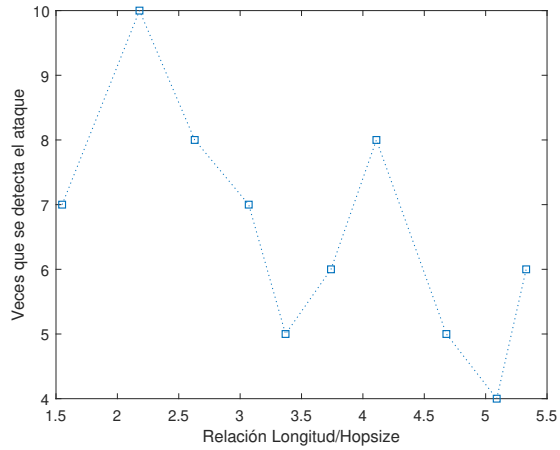


(a) Escenario con 3 hook points.

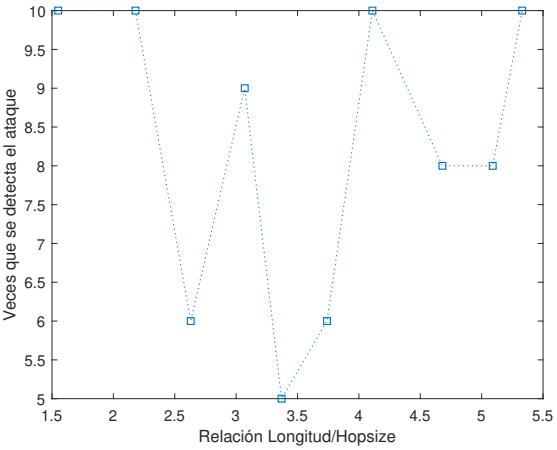


(b) Escenario con 4 hook points.

Figura 4.8: Escenarios detectados para 20 nodos.

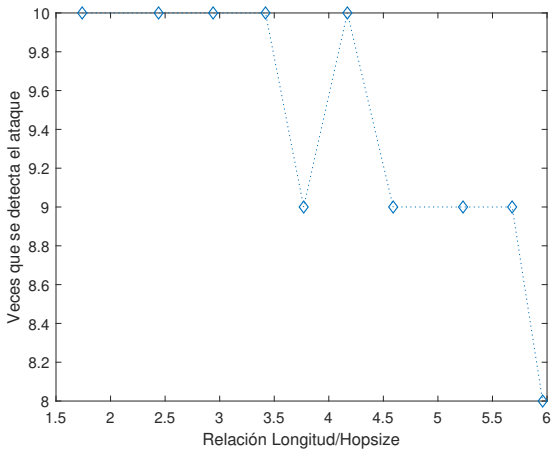


(a) Escenario con 3 hook points.

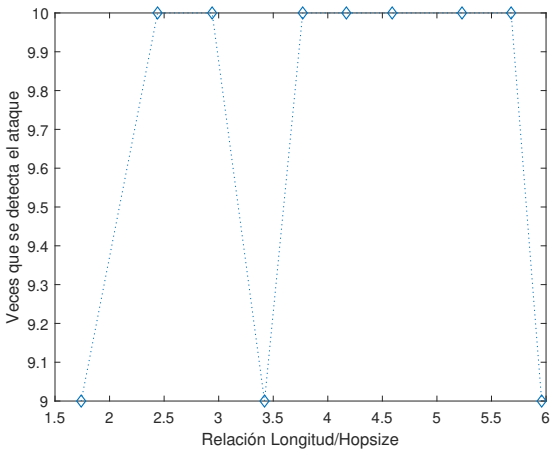


(b) Escenario con 5 hook points.

Figura 4.9: Escenarios detectados para 50 nodos.



(a) Escenario con 5 hook points.



(b) Escenario con 10 hook points.

Figura 4.10: Escenarios detectados para 100 nodos.

Capítulo 5

Conclusiones

En este capítulo, se presentan las conclusiones generales de esta tesis, así como la verificación de la hipótesis con base en los resultados obtenidos para el mecanismo propuesto. Finalmente se presentan las perspectivas de investigación.

5.1. Conclusiones generales

Las redes inalámbricas forman parte de nuestro día a día debido a las múltiples aplicaciones que brindan; además suponen un gran ahorro en la mayoría de los escenarios donde operan, sin embargo este tipo de redes cuentan con varias desventajas, y una de las principales es que suelen ser inseguras en términos de ocultar la información que procesan, por ello se han planteado diferentes formas de proteger la privacidad en este tipo de redes. Especialmente porque suelen funcionar en entornos inhóspitos. El ataque *wormhole* puede ser lanzado fácilmente por dos atacantes coludidos sin autorización del sistema. Este ataque es particularmente desafiante, ya que el adversario no necesita comprometer ningún nodo ni tener acceso a ninguna clave criptográfica, por lo que es importante detectar y eliminar sus influencias. Los trabajos relacionados mostrados en esta tesis proponen diferentes métodos para contrarrestarlo, sin embargo algunos son costosos o difíciles de implementar, demandan demasiados recursos o necesitan información adicional, algunos otros realizan suposiciones discordantes o necesitan el conocimiento de toda la red. Esta tesis hace uso de dos algoritmos robustos que trabajan conjuntamente (*DV-Hop y Spanning tree*) para lograr el objetivo planteado: detectar y localizar el ataque conociendo únicamente el 1-vecindario y haciendo uso de los cambios topológicos. En el capítulo 4 se muestran diferentes escenarios donde se puede ver la efectividad del mecanismo propuesto, se puede observar que la propuesta de esta tesis es capaz de detectar el ataque entre un 61 y 97 % de veces. Por ejemplo, en la figura 4.7, podemos ver que para una red de 10 nodos y 3 hook points se tiene el 61 % de efectividad y para el mismo escenario incrementando el número de nodos ancla a 4 se tiene un total del 71 % de detecciones, como podemos ver en este caso aumentar un nodo ancla incrementa el 10 % de detecciones. En la figura 4.8 para la red de 20 nodos se tienen 78 % y 83 % de detecciones para escenarios con 3 y 4 hook points respectivamente, en este caso la diferencia no

es significativa en cuestión de detección de ataques, únicamente se tienen 5 % más de detecciones. En la figura 4.9 se puede apreciar que para una red formada por 50 nodos se detectó el ataque en un 65 % de las veces teniendo solo 3 hook points, mientras que para 5 hook points se tiene un porcentaje de efectividad del 82 %, aquí el incremento en porcentaje es del 17 %, pero ahora la diferencia de nodos ancla es de 2, por lo que el costo de hardware sería mayor. Por último, en la figura 4.10 se observa que para un escenario formado por 100 nodos se tuvo 94 % de efectividad cuando se tienen 5 nodos ancla y 97 % para el caso de 10 hook points, en este caso la diferencia no es tan significativa en cuestión de detección, pero tomando en cuenta el costo de hardware adicional incrementaría al doble, puesto que el número de nodos ancla pasa de 5 a 10.

En todos los casos, al tener un número relativamente bajo de nodos ancla respecto al número de nodos de la red es posible tener un porcentaje de efectividad mayor al 60 %. Se puede ver que la detección correcta del ataque depende de la posición de los nodos ancla respecto al tamaño de la red y también del tamaño del canal, es decir, si el canal tiene una longitud corta es más difícil detectarlo. Sin embargo, este caso no es común, ya que un nodo malicioso busca hacer el mayor daño posible y un canal corto no altera en gran porcentaje la topología de la red. Lo que se puede observar durante todas las pruebas es que es más importante la distribución correcta de los nodos ancla, más que el número de nodos ancla, ya que es posible tener un alto porcentaje de detecciones si los nodos ancla están separados y distribuidos sobre la red. Es decir, no se requiere de un número alto de hook points, además que esto afecta el costo y el número de mensajes en la primera etapa del algoritmo (DV-Hop).

5.2. Verificación de la hipótesis

Retomando la hipótesis presentada en la sección 1.2:

> "*El uso de características topológicas combinado con hardware de posicionamiento global permite detectar y eliminar el ataque wormhole conociendo únicamente el 1-vecindario.*"

Para verificar la validez de la hipótesis previamente mostrada, esta tesis se desarrollo en dos etapas. En la primera etapa, a partir del análisis de los algoritmos utilizados (DV-Hop y spanning tree), se realizaron pruebas teóricas tomando en cuenta varios escenarios (Sección 3.1.1, 3.2.1 y estudios estadísticos). En la segunda etapa se simularon múltiples escenarios en el lenguaje interpretado de alto nivel Python (Sección 3.3). Como se muestra en el figuras 4.5 y 4.6 y las figuras 4.7, 4.8, 4,9 y 4.10 se puede ver que el algoritmo es capaz de detectar un alto número de ataques bajo diversas condiciones. Por ejemplo, se puede observar que el algoritmo es capaz de detectar hasta el 94 % de los ataques con pocos nodos ancla, tal es el caso de una red de 100 nodos con 5 hook points. Tambien estas pruebas muestran que el algoritmo propuesto es capaz de detectar ataques cuando la longitud del ataque es pequeña con más del 70 % de las veces (para la mayoría de los escenarios) y hasta 100 % de las veces con longitudes de 5 veces la relación del largo del canal con respecto al hopsize. Con esto podemos concluir que la hipótesis es verdadera debido a que es posible localizar la

zona en la que se está dando el ataque en el mayor número de escenarios y con ello suprimir el ataque; para esto únicamente se necesita el conocimiento de 1-vecindario.

5.3. Perspectivas de la investigación

Se implementará el mecanismo propuesto en esta tesis dentro de un simulador especializado en redes de sensores con el fin de hacer un análisis más detallado de los resultados, ya que en esta tesis no se consideraron variaciones en las señales inalámbricas, pérdidas de paquetes y mecanismos complejos de acceso al medio. Esto permitirá tener resultados más cercano a la realidad y poder obtener un porcentaje de efectividad de esta contramedida para el ataque *wormhole* en una red de sensores.

Apéndice A

Código principal para el ataque en Python versión 2.7

```
#!/usr/bin/env python
#
#    Wormhole detection
#    Copyright (C) 2020, A. Galvan
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <https://www.gnu.org/licenses/>.
#

import numpy as np
from kdTree import *
from HopsCircle import *
import random
########################
## MAIN
#####################################

def main():
    #seed = 2380438460
    #seed = 3539892598
    seed = 3632731615
```

```
    size = 500
    N = 10 # number of vertices
    start = 'id_4'
    coverage = 230 # coverage radius
    wormCoverage = 180 # coverage of wormhole
    np.random.seed(seed)
    x = np.random.uniform(0.0,float(size),N)
    y = np.random.uniform(0.0,float(size),N)
    points = zip(x, y)

    # creates the kdtree

    kdTree = buildKdtree(points, points)
    # creates a graph G
    G = {}
    inOrderTraversal(kdTree, G)
    for i in G.values():
        searchNeighbors(i, kdTree, coverage) # last argument means
  the coverage area of a node
    ####### create random hooks ################################
    #hook = ['id_124', 'id_56', 'id_89', 'id_4']
    hook = random.sample(G.keys(), k=3)
    ####### wormHole attack  ##########################
    wormnodes = wormhole((142,493),(490,10),kdTree,G, wormCoverage)
    #wormnodes = []
    ######### DV-hop     ##############################
    dvHop(G, hook)
    nodes = G.keys() # set of vertices
    for n in hook:
        G[n].autoLoc = 2 # this means hook point
        nodes.remove(n) # remove hook points from vertices since
  they know its location
    hopsizeMean = []  # to print hopsize mean (not necesarily)
    for p in nodes: # auto locatization process
        if (autoLocalization(G,hook, p, hopsizeMean) == True):
            G[p].autoLoc = 1
        else:
            G[p].autoLoc = 0
    print "hopzise mean is {}".format(np.mean(hopsizeMean))
    #np.random.seed()
    #  send multicast to build the spanning tree
    spanning = []
    appear = [start]
    while len(appear)>0:
        index = appear.pop(0)
        G[index].multicast(G,spanning,appear)
        #print " ====== SPANNING EDGES ======"
        #for item in spanning:
        #    print item
        #print " ====== Appear  ======="
        #print appear

    print "spanning's len {}".format(len(spanning))
    #  printing the graph
```

```
84      printGraph(G,size,start, spanning, wormnodes, wormCoverage)
85
86  if __name__ == "__main__":
87      main()
```

Apéndice B

Código para KdTree

```
#! /usr/bin/env python

#######################################################################
#
#       This class implements kd-tree. Given a point and range, this class looks for
#        a set of nearest neighbor.  A. Galvan
#
#
#######################################################################
import matplotlib.pyplot as plt
import numpy as np
from packet import *
import math
import sys
infinity = sys.maxint-1
#########################################################


class Vertex:
    def __init__(self, name, pt):
        self.id = 'id_{}'.format(name)
        self.neighbors = {}
        self.pi = []
        self.hook = {}
        self.autoLoc = infinity
        self.key = infinity
        self.memory = []
        self.breakPoint = 0 # 0 means normal node, 1 breakPoint due to empty set, 2 breakPoint due to have multiple fathers
        self.leftChild = None
        self.rightChild = None
        self.location = pt
        self.axis = None

    def multicast(self, graph, spanning, order):
        #print "run on node {}".format(self.id)
        # set A
```

```
37          set_a = set(self.neighbors.keys())
38          #print "set A {}".format(set_a)
39          # set B
40          b = []
41          if len(self.memory)>0:
42              for pkt in self.memory:
43                  b.append(pkt.src)
44                  b = b+pkt.dst
45          set_b = set(b)
46          #print "set B {}".format(set_b)
47
48          dsts = list(set_a - set_b) # multicast destination (A-B)
49          if dsts == []: # breakPoint due to empty set
50              self.breakPoint = 1
51          ## create order list ###
52          for j in dsts:
53              if j not in order:
54                  order.append(j)
55          np.random.shuffle(order) # visiting nodes in random
    selection
56          ## send pkt to dsts
57          for i in dsts:
58              graph[i].send(packet(self.id, dsts, "spanning-tree"))
59              ### create spanning tree ###
60              spanning.append(sorted((self.id,i)))
61          ###  seek for more multiple fathers (loops) #####
62          if len(self.pi) > 1:
63              for item in self.pi[1:]:
64                  spanning.remove(sorted((self.id, item)))
65                  self.breakPoint = 2 # breakpoint due to multiple
    fathers
66                  #print "remove edge {}".format((self.id,item))
67
68
69      def send(self, pkt):
70          self.memory.append(pkt)
71          self.pi.append(pkt.src)
72
73
74  def wormhole(x1,x2, kd, graph, radii):
75      wormLeft = Vertex('id_l', x1)
76      wormRight = Vertex('id_r', x2)
77      searchNeighbors(wormLeft, kd, radii)
78      print "wormLeft is {}".format(wormLeft.neighbors.keys())
79      searchNeighbors(wormRight, kd, radii)
80      print "wormRight is {}".format(wormRight.neighbors.keys())
81      for i in wormLeft.neighbors.keys():
82          for j in wormRight.neighbors.keys():
83              graph[i].neighbors[j] = distance(graph[i].location,
    wormLeft.location)
84              graph[j].neighbors[i] = distance(graph[j].location,
    wormRight.location)
85      print "Lenght of wormhole is {} meters".format(distance(wormLeft
    .location, wormRight.location))
```

```
86      return [wormLeft, wormRight]
87
88  def buildKdtree(pointList, entirelist, depth=0):
89      if not pointList:
90          return
91
92      #Select the axis based on depht.
93      k = len(pointList[0])
94      axis = depth % k
95
96      #Sort points depended on axis and choose the midpoint
97      pointList.sort(key=lambda point:point[axis])
98      median = len(pointList) // 2
99
100     node = Vertex(entirelist.index(pointList[median]), pointList[
        median])
101     node.axis = axis
102     node.leftChild = buildKdtree(pointList[0:median], entirelist,
        depth+1)
103     node.rightChild = buildKdtree(pointList[median+1:], entirelist,
        depth+1)
104     return node
105
106 def searchNeighbors(q, node, radius):
107     if (node == None):
108         return
109
110     if (isInside(q.location, node.location, radius) == 1):
111         if q.id != node.id:
112             q.neighbors[node.id] = distance(q.location, node.
        location)
113
114     if (node.location[node.axis] < q.location[node.axis]+radius):
115         searchNeighbors(q, node.rightChild, radius)
116
117     if (node.location[node.axis] > q.location[node.axis]-radius):
118         searchNeighbors(q, node.leftChild, radius)
119
120
121 def isInside(pointA, pointB, dist):
122
123     if (((pointB[0] - pointA[0])**2 + (pointB[1] - pointA[1])**2) <=
        dist**2):
124         return 1
125     else:
126         return 0
127
128 def distance(pointA, pointB):
129     return math.floor(math.sqrt((pointB[0] - pointA[0])**2 + (pointB
        [1] - pointA[1])**2))
130
131 def inOrderTraversal(node, graph):
132     "Inorder traversal"
133     graph[node.id] = node
```

```
134     inOrderHelper(node, graph)
135 
136 def inOrderHelper(node, graph):
137     "Inorder traversal helper function"
138 
139     if node is not None:
140   inOrderHelper(node.leftChild, graph)
141   graph[node.id]= node
142   inOrderHelper(node.rightChild, graph)
143 
144 def printVertices(vertex,col,ax):
145     ax.plot(vertex.location[0],vertex.location[1], 'o', color='black
    ',markersize=3)
146     ax.text(vertex.location[0],vertex.location[1],vertex.id.replace(
    'id_','),bbox={"boxstyle" : "circle","color": col})
147 
148 def printGraph(graph, size, start, spanning, wormnodes, coverage):
149     vertices = graph.values()
150     edges = []
151     for i in vertices:
152         for j in i.neighbors.keys():
153             temp = sorted((i.id,j))
154             if temp not in edges:
155                 edges.append(temp)
156 
157     ####### plot ########
158     ax = plt.gca()
159     ax.cla() # clear things for fresh plot
160     ## change default range so that new circles will work
161     ax.set_xlim((0,size))
162     ax.set_ylim((0,size))
163     ax.set_title('spanning tree started in {}'.format(start))
164     fig = plt.gcf()
165 
166     ###### print vertices  ####################
167     for item in vertices:
168         if item.autoLoc == 1:
169             printVertices(item, 'yellow',ax)
170         if item.autoLoc == 0:
171             printVertices(item, 'green',ax)
172         if item.autoLoc == 2:
173             printVertices(item, 'gray',ax)
174     for item in wormnodes:  ### print wormholes
175         printVertices(item, 'cyan', ax)
176     if len(wormnodes) > 0:
177         circle1 = plt.Circle((wormnodes[0].location[0], wormnodes
    [0].location[1]), coverage, color='black', fill=False)
178         circle2 = plt.Circle((wormnodes[1].location[0], wormnodes
    [1].location[1]),coverage , color='black', fill= False)
179         ax.add_patch(circle1)
180         ax.add_patch(circle2)
181     plt.axis('scaled')
182 
183     for edge in edges:
```

```
184         ax.plot(np.array([graph[edge[0]].location[0],graph[edge[1]].
    location[0]]),np.array([graph[edge[0]].location[1],graph[edge
    [1]].location[1]]),color='gray',alpha=0.5)
185
186
187     for edge in spanning:
188         ax.plot(np.array([graph[edge[0]].location[0],graph[edge[1]].
    location[0]]),np.array([graph[edge[0]].location[1],graph[edge
    [1]].location[1]]),color='red',alpha=0.5)
189
190     plt.show()
```

Dentro de este código se menciona una clase en la que se define el envío de paquetes entre los nodos:

```
1 #! /usr/bin/env python
2
3
4 class packet:  # class packet
5     def __init__(self, src, dst, msg):
6         self.src = src
7         self.dst = dst
8         self.msg = msg
```

Apéndice C

Programa para el algoritmo DV-HOP

```
##############################################
#     Dijkstra class (python 2.7)            #
##############################################
import sys
infinity = sys.maxint-1
import numpy as np
from shapely.geometry import Point


def shortestPath(Gr,root):
    Q = Gr.values() # Queue
    Gr[root].key = 0 # start path
    Gr[root].hook[root] = None # pi's of hook is none

    while Q:
        Q.sort(key=lambda x : x.key)
        u = Q.pop(0)

        for v in u.neighbors.keys():
            if Gr[v].key > u.key + u.neighbors[v] :
                Gr[v].key = u.key + u.neighbors[v]
                Gr[v].hook[root] = u

def printPath(Gr, end, root):
    l = []
    current = Gr[end]
    while current != None :
        l.append(current.id)
        current = current.hook[root]
    l.reverse()
    print l


def autoLocalization(Gr, hooks, vertex, mean):
    dist = []
    hops = []
    for item in hooks:
        current = Gr[vertex]
```

```
        h = 0
        while current.id != Gr[item].id: # measures the distance between vertex and hook point
            dist.append(current.neighbors[current.hook[item].id])# distance between current node and his father
            current = current.hook[item]
            h = h+1  # measures the number of hops
        hops.append(h)
    #print "dist {}".format(dist)
    #print "hops {}".format(hops)
    ### create boxes ##########
    hopsize = np.sum(dist)/np.sum(hops)
    mean.append(hopsize)
    circles = []
    for couter, value in enumerate(hooks):
        radius = hops[couter]*hopsize
        point = Point(Gr[value].location[0],Gr[value].location[1])
        circles.append(point.buffer(radius))
    ## intersect boxes #####
    cic1cic2 = circles[:2]
    if (cic1cic2[0].intersects(cic1cic2[1])) == False:
        return False
    for b in circles[2:]:
        for k in cic1cic2:
            if (k.intersects(b)) == False:
                return False
        cic1cic2.append(k)
    return True

def dvHop(Gr, hooks):
    for hook in hooks:
        shortestPath(Gr,hook)
        #for item in Gr.keys():
        #    printPath(Gr,item, hook)
        resetKeys(Gr)
    print "################################"

def resetKeys(Gr):
    for node in Gr.values():
        node.key = infinity
```

Bibliografía

[1] Johan S. Rueda R. and Jesús M. Talavera. P., 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.

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

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

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

[5] Eriksson and 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

[6] P. V. Tran and L. X. Hung and Y. Lee and 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] K. Harsányi and 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

[8] D. Dong and M. Li and 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

[9] R. Maheshwari and 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

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

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

[12] 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.

[13] He Ronghui, 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] Nabila Labraoui, Mourad Gueroui and Makhlouf Aliouat, Secure DV-Hop localization scheme against wormhole attacks in wireless sensor networks, December 2011, DOI: 10.1002/ett.1532.

[15] 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.

[16] X. Liu, 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.

[17] B. Liu, 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.

[18] Jinlong Liu, Zhilu Wu and Zhendong 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.

[19] X. Shi, 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.